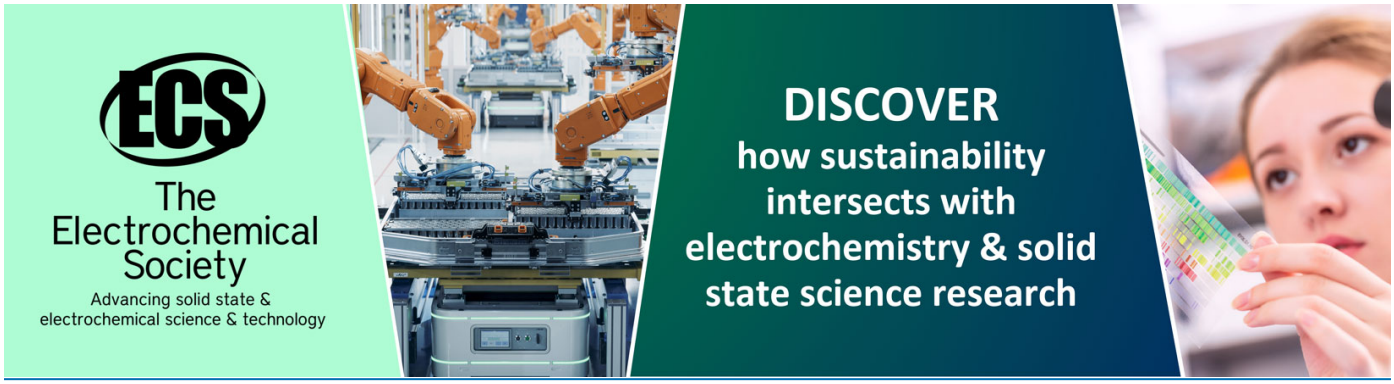
FTS3: New Data Movement Service For WLCG



FTS3: New Data Movement Service For WLCG

A A Ayllon, M Salichos, M K Simon, O Keeble

CERN, European Organization for Nuclear Research

E-mail: alejandro.alvarez.ayllon@cern.ch, Michail.Salichos@cern.ch, Michal.Simon@cern.ch, Oliver.Keeble@cern.ch

Abstract. The File Transfer Service (FTS) is the service responsible for distributing the majority of LHC data across the WLCG infrastructure. We present the current status and features of the new File Transfer Service (FTS3), which addresses the problems that the previous FTS version faced : static channel model, configuration and scalability problems, new protocols support, more database back-ends support, etc. We present the solution we implemented and the design of the new tools as well the reliability, stability, scalability and performance requirements of a data movement middle-ware in the grid environment. The ultimate goal has been to deliver a service which scales horizontally, is easy to install and configure, supports protocols via a plug-in based mechanism (GFAL2) and can perform multi GB/s data transfer on the full mesh of its tiered centres.

1. Service Architecture

FTS3 (File Transfer Service, version 3.0) is one of the projects of critical importance for data management at CERN [1]. It has been designed in a modular and extensible way to allow good scalability. The components of the FTS (shown in Figure 1) are: CLI clients, a daemon process responsible for transfer submission, status retrieval and general VO (Virtual Organization) and service configuration, another daemon process responsible for bulk stage-in of files from archive using the SRM [2] protocol (BringOnline operation) and, finally, the database back-end. The service scales very well horizontally by adding more resources with identical configuration into an FTS3 cluster, since the configuration is stored into the database and is being read during start-up of the service. A single configuration file is used only to store the database credentials.

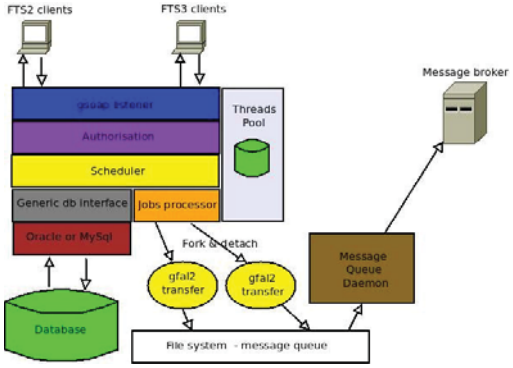


Figure 1. FTS3 service architecture

2. Main features

FTS3 offers features and functionality that have been requested by the experiments and sites following this initial usage of FTS2 [3, 4] and the needs that the new FTS3 service should address. Most of these have already been implemented or are in the process of being finalized. For example:

- transfer auto-tuning/ adaptive optimization;
- endpoint-centric VO configuration;
- transfer multi-hop;
- VO activity shares;
- multiple replica support;
- REST-style interface for transfer submission and status retrieval;
- retry failed transfers mechanism;
- staging files from archive;
- support for Oracle and MySQL database back-ends;
- transfer and access protocols support on top of gfal2 plug-in mechanism (SRM, GridFTP [5], HTTP, xroot);
- session/connection reuse (GridFTP, SSL, etc), which is ideal for many small file size transfer jobs.

In order to be a credible long-term platform for data transfer, FTS3 has been designed to exploit upcoming developments in networking, such as integrating monitoring data from perfSonar for further transfer optimization, resource management and monitoring network state.

2.1. Adaptive optimization

Adaptive optimization is the mechanism introduced in FTS3 to address the shortcomings or restrictions of its predecessor (FTS2). FTS2 require file transfers to be performed on channels which must be defined and configured explicitly. The FTS3 auto-tuning algorithm and channel-less model allows moving from a structured topology to a full-mesh (as shown in Figure 2) for networks links and storage elements without any configuration and, at the same time, optimizing transfers based on information retrieved from the database.

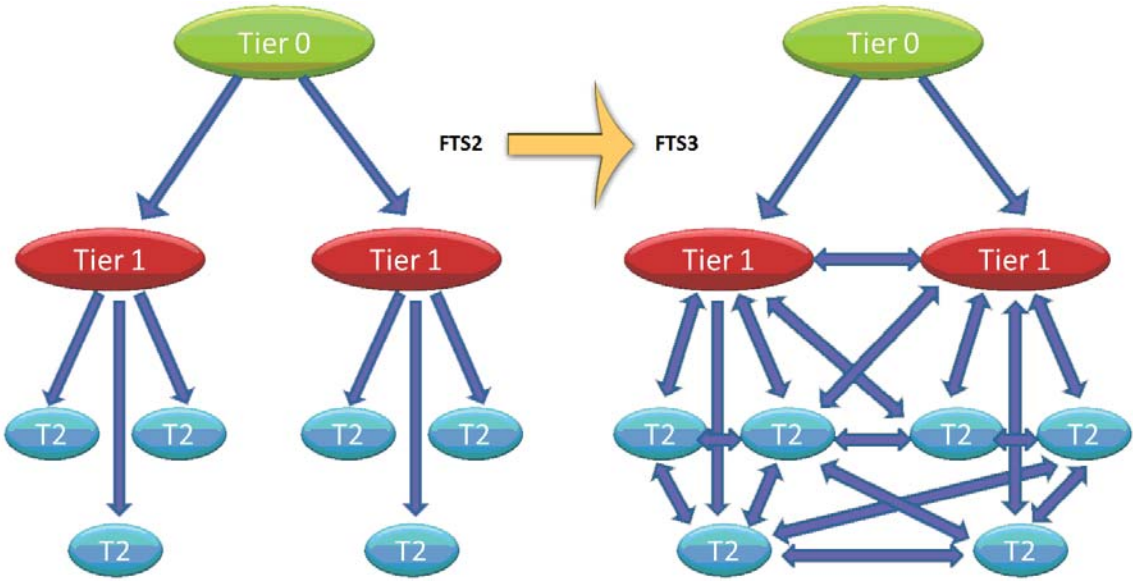


Figure 2. move from hierarchical topology to full mesh using FTS3 (zero-config)

The plot in Figure 3 demonstrates how the adaptive optimization algorithm is influenced by the achieved throughput and success rate of distinct links, and dynamically adjusts the number of concurrent file transfers based on this information. A sample is taken every 30secs and the decisions of the optimization are stored into the database for persistency and service monitoring.

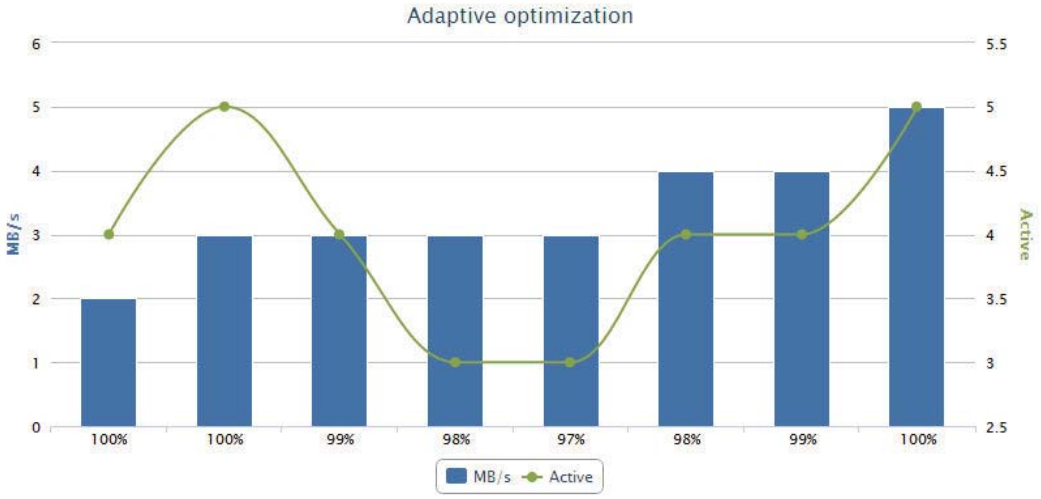


Figure 3. Adaptive optimization plot

2.2. Managing resources

In the FTS3 world resource management is Storage Element (SE) centric and it can be done at three different levels: sharing the resources between Virtual Organizations, specifying the number of concurrent transfer-jobs and defining custom protocol parameters (TCP buffer size, number of open sockets per transfer-job, etc.). The motivation for the SE-centricness was the change in computing model from a strict hierarchy to a mesh [6]. While in the case of a channel based policy the number of configurations required for full-mesh connectivity grows quadratically with the number of SEs, in case of a SE centric policy the growth is linear (for example, a setup of 6 SEs requires either 15 channel configurations or 6 SE-centric configurations). There are four types of configurations (all in JSON format) available in FTS3:

- The standalone SE configuration type has been created so the user can address the problem of overloaded SEs by limiting the total number of outgoing and incoming transfers. Those limits can be imposed per VO in order to create so called 'VO-shares'. While transfer-jobs are carried out between two SEs with standalone configurations, the limits for both the source and the destination are respected. On the other hand, their protocol configurations are merged resulting in consistent settings for the given pair (e. g. FTS3 will pick the smaller timeout value).

```
{
  "se" : " srm://se.cern.ch",
.."active" : true,
.."in" : {
    "share" : [{"cms" : 5}, {"atlas" : 5}, {"public" : 5}],
    "protocol" : [{"nostreams" : 10}, {"urlcopy_tx_to" : 3000}]
  },
  "out" : {
    "share" : [{"cms" : 6}, {"atlas" : 5}, {"public" : 4}],
```

```
    "protocol":[{"nostreams" : 12}, {"urlcopy_tx_to" : 3600}]
  }
}
```

- The SE pair configuration allows configuring explicitly the link between two SEs. This type of configuration should be used only in case when the standalone configurations do not apply to this particular pair (e.g. the destination SE is in an unusual location and requires much bigger timeout than other SEs). The SE pair configuration overwrites all the settings from the respective standalone configurations.

```
{
  "symbolic_name" : "se-link",
  "source_se" : " srm://se1.cern.ch",
  "destination_se" : " srm://se2.cern.ch",
  "share" : [{"cms" : 1}, {"atlas" : 2}, {"public" : “auto”}],
  "protocol" : “auto”,
  "active":true
}
```

- The SE group configuration is meant to address a particular use case when a group of SEs shares a common resource (e. g. a common network link) that may limit their aggregated performance. Hence, the SE group configuration allows imposing a limit on the total number of incoming and outgoing transfer-jobs for a whole group of SEs. If a SE has a standalone configuration and is also a member of a group both limits are respected: the one imposed by the standalone configuration and the one imposed by the group configuration.

```
{
  "group" : "gr1",
  "members" : [" srm://se1.cern.ch", " srm://se2.cern.ch"],
  "active" : true,
  "in" : {
    "share" : [{"cms" : “auto”}, {"atlas" : 12}],
    "protocol" : [{"nostreams" : “auto”}, {"urlcopy_tx_to" : 3600}]
  },"out" : {
    "share" : [{"cms" : 10}, {"atlas" : “auto”}],
    "protocol" : [{"nostreams" : 10}, {"urlcopy_tx_to" : “auto”}]
  }
}
```

- The SE group pair configuration (analogous to the SE pair configuration) allows to overwrite the group configuration settings for a pair of two particular SE groups.

```
{
  "symbolic_name" : "group-link",
  "source_group" : "gr1",
  "destination_group" : "gr2",
  "share" : [{"public" : 80}],
  "protocol" : [{"nostreams" : "auto"}],
  "active":true
}
```

Each of the above mentioned configurations can be hybridized with the auto-tuner mechanism through replacing any value with the 'auto' keyword.

2.3. RESTful interface

FTS3 provides a REST [7] interface to submit transfers and query their state. This interface is to a great extent self-discoverable, as it provides information about how to reach each resource, and which sort of interaction they support [8]. We have tried to honour the restrictions imposed by some authors to consider a RESTful interface a proper RESTful interface [9]:

- Each resource has its own URI
  Collection of jobs:
  https://fts3.cern.ch:8446/jobs/
  For getting a given job:
  https://fts3.cern.ch:8446/jobs/083ac623-5649-4127-1234-abcdef123456
- Meaningful use of HTTP verbs
  GET <job-uri> retrieves information about the given job
  DELETE <job-uri> cancels a job
  POST <job-collection-uri> submits a new job
- Hypermedia. Although this third point is not complete – not every resource provides information about the actions that can be performed – we do provide self-discovery information on the root level of our API using the JSON Hypertext Application Language [10].

Even though the default deployment is to have all interfaces running in all the FTS3 machines, the FTS3 REST interface can be easily deployed separately – to reduce load, for instance.

It is worth noting that we also provide the JSON Schema [11] of the expected JSON message sent for submission. Using this schema, a client application can easily validate its message before sending it to the server, to check if it complies with the expected format.

In Python, for instance, it can be as easily done as

```
import jsonschema
import json
import requests

root_ca    = 'root-CA.pem'
my_proxy   = 'user-proxy.pem'
raw_schema = requests.get('https://fts3-pilot.cern.ch:8446/schema/submit',
                          verify=root_ca, cert=my_proxy).text
schema     = json.loads(raw_schema)
submission = json.loads(open('submit.json').read())
jsonschema.validate(submission, schema)

submission['params']['verify_checksum'] = 1234
jsonschema.validate(submission, schema)
Traceback...
jsonschema.ValidationError: 1234 is not of type [u'boolean', u'null']
```

2.4. Protocol support

FTS3 relies on the GFAL2 [12] library, which provides an abstraction layer over the grid complex storage system. Support for several protocols is provided by a collection of plug-ins. This plug-ins based system has three big advantages: independence, isolation and extensibility.

- Independence: The client application – in this case FTS3 – can be written independently of the protocol, or set of protocols, that will be used. Thanks to this we can avoid pulling dependencies for not needed protocols.

- Isolation: The protocol specific logic is contained inside its corresponding plug-in, so the addition of new functionality, fixes or any other changes can be done without risk of affecting the behaviour of the other protocols.
- Extensibility: Adding a new protocol is just a matter of installing a new plug-in.

As mentioned, FTS3 in principle can support any protocol that GFAL2 supports. Currently, these are DCAP, GridFTP, HTTP(S), RFIO, SRM and XROOTD. Some other protocols are supported by GFAL2 too, as LFC or direct access to a local file, but there are not of relevance for FTS3.

It is worth mentioning that, from this list, only GridFTP and SRM were supported by the previous version, FTS2.

3. Experience

The service has already undergone extensive pre-production validation and we have demonstrated the results of high volume production transfers performed on the pilot service and production-ready instances.

The plot in Figure 4 shows the usage of FTS3 instance at GridPP, used by ATLAS and CMS for production and debug transfers. It is worth mentioning that the volume shown below (first bar = 370TB), was achieved using zero configuration (adaptive optimization) on top of MySql database running on a VM.

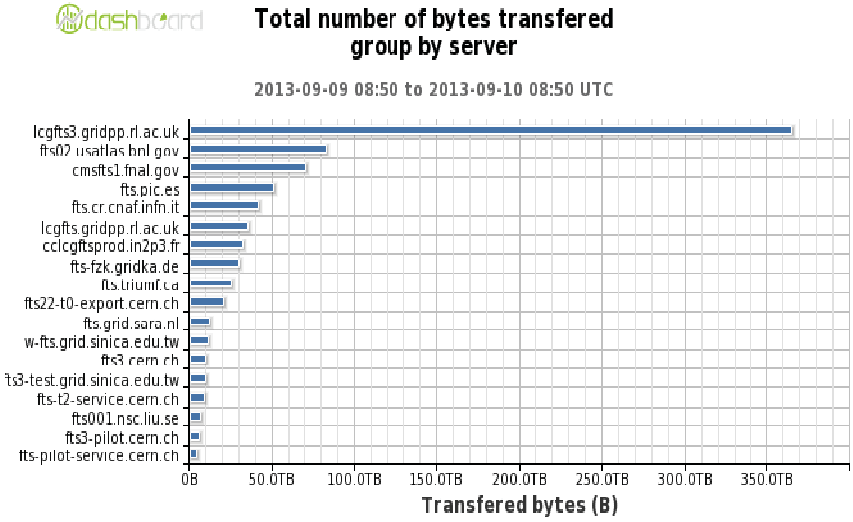


Figure 4. Transfer volume of GridPP FTS3 instance

Anticipating the upcoming data movement needs of WLCG, and building on the lessons learned during the first run, we present a new, scalable and highly-optimized data movement service, which provides a simple interface for transfer job submission, status retrieval, advanced monitoring capabilities, multiple access and transfer protocols support and simplified configuration.

Acknowledgement

We would like to gratefully and sincerely thank Mr. Andrew Lahiff from GridPP for the support and enthusiasm has been showing since the very first days of FTS3.

References

[1] Critical services in the LHC computing, A Sciabà 2010 *J. Phys.: Conf. Ser.* 219

[2] Abadie L et al 2007 24th *IEEE Conference on Mass Storage Systems and Technologies* pp.47,59, 24-27
[3] Data management in EGEE, A Frohner *et al* 2010 *J. Phys.: Conf. Ser.* 219
[4] Abadie L et al 2007 24th *IEEE Conference on Mass Storage Systems and Technologies* pp.60,71, 24-27
[5] Allcock W et al 2005 *ACM/IEEE Supercomputing* pp.54,54, 12-18
[6] The evolving role of Tier2s in ATLAS with the new Computing and Data Distribution model, S Gonzalez de la Hoz 2012 *J. Phys.: Conf. Ser.* 396
[7] Architectural Styles and the Design of Network-based Software Architectures, Roy T. Fielding *Dissertation*, 2000
[8] Principled Design of the Modern Web Architecture, Roy T. Fielding and Richard N. Taylor, *ACM Transactions on Internet Technology*, Vol. 2, No.2, May 2002, Pages 115 – 150
[9] Justice Will Take Us Millions Of Intricate Moves, Leonard Richardson*, talk at the International Software Development Conference* (QCon), 2008
[10] JSON Hypertext Application Language (Draft), M. Kelly, *IETF Draft*, October 2013
[11] JSON Schema
[12] GFAL 2.0, Adrien Devresse, *Presentation at the GDB*, November 2012