GRID DATA MANAGEMENT: RELIABLE FILE TRANSFER SERVICES' PERFORMACE

G.A. Stewart, University of Glasgow, Glasgow, UK
G. McCance, CERN, Geneva, Switzerland

Abstract

Data management has proved to be one of the hardest jobs to do in a the grid environment. In particular, file replication has suffered problems of transport failures, client disconnections, duplication of current transfers and resultant server saturation.

To address these problems the globus and gLite grid middlewares offer new services which improve the resiliency and robustness of file replication on the grid. gLite has the File Transfer Service (FTS) and Globus offers Reliable File Transfer (RFT). Both of these middleware components offer clients a web service interface to which they can submit a request to copy a file from one grid storage element to another. Clients can then return to the web service to query the status of their requested transfer, while the services can schedule, load balance and retry failures between the received requests.

In this paper we compare these two services, examining,

1. Architecture and features offered to clients and grid infrastructure providers.
2. Robustness under load: e.g., when large numbers of clients attempt to connect in a short time or large numbers of transfers are scheduled at once.
3. Behaviour under common failure conditions - loss of network connectivity, failure of backend database, sudden client disconnections.

Lessons learned in the deployment of gLite FTS during LCG Service Challenge 3 are also discussed.

Finally, further development of higher level data management services, including interaction with catalogs in gLite File Placement Service and Globus Data Replication Service is considered.

INTRODUCTION

Initial efforts to manage data on the grid concentrated primarily on data movement protocols. These protocols, such as `gridftp`[7] and `http+modgridsite`[5], are low level protocols focused on actually shifting data between machines in a grid environment.

However, even though these data transport protocols have evolved and are now quite robust, there is a problem when a user's client computer itself initiates the transfer. When this happens the client itself holds important information about the state of the transfer. This means that the client cannot disconnect from the network – which may be inconvenient for the user – and that if it does the transfer fails[1].

More importantly though, when data transfers are initiated at the client level, there is no notion of the global state of data transfers between two sites. This can lead problems for sites[2] and clients[3]:

- Storage elements can be overwhelmed with transfer requests.
- Clients can attempt to replicate the same file simultaneously onto the same storage element.
- Sites have little control over the use of their network resources.

To address these problems there is a clear need for a data transfer *service*, shown in Figure 1. Such a service should provide clients with a stateless interface, so which they can connect to submit a transfers, monitor status or cancel a transfer request. It should also store the state of transfers robustly, allowing for breaks in connectivity and restarts of the service. Finally, it should offer some ability for sites to manage data flow in and out of their storage.

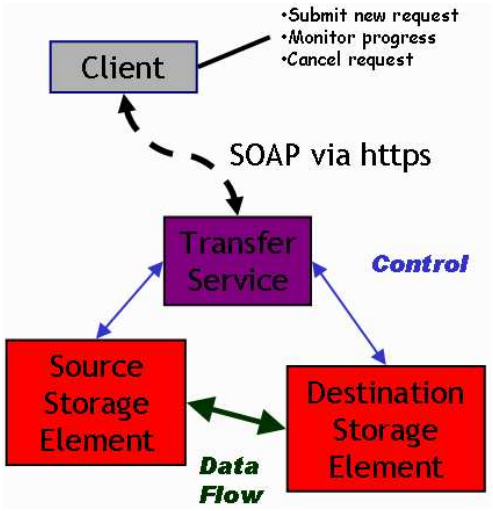

Figure 1: Architecture of a transfer service.

SERVICE ARCHITECTURE

Globus RFT

The Globus Reliable File Transfer[9] (RFT) service presents a stateless web service to clients, which can run

in either a tomcat5 container or inside the globus WS container. Underlying the stateless web service, RFT stores transfer status in a database (currently PostgreSQL or MySQL), ensuring robustness and service continuity.

A few parameters are controlled at the RFT instance level. Of particular importance is the maximum number of concurrent transfers which will be attempted. Unfortunately this feature is currently not working, so no global concurrency limit can be set in RFT at the moment.

To submit a transfer request to RFT, clients use SOAP over https, giving a list of source and destination `gsiftp://` URLs. Certain transfer parameters (gridftp streams, concurrency, tcp buffer sizes) can be specified, as well as a maximum lifetime for the transfer. Jobs consisting of recursive copies can be submitted. An "all or nothing" flag can be set, which indicates whether partial success of a transfer is acceptable (if not, files which did transfer successfully will be deleted).

Clients can query the transfer state later (polling) or register for status updates with the RFT service.

It is possible to submit mass deletion jobs to RFT.

RFT is currently capable of transferring between storage elements offering gridftp interfaces. No integration with Storage Resource Manager[8] interfaces on SEs is available.

gLite FTS

The gLite File Transfer Service[6] (FTS), like the RFT, presents a web service interface (running in a tomcat5 container) to its clients. The web service interface stores transfer state in a backend database (currently Oracle and MySQL are supported), which acts as a the single point of persistence for the service.

When clients submit file transfer requests they submit a list of source URLs (SURLs).and destination SURLs, expressed as `srm://` URLs. Other parameters, such as gridftp options, can also be given.

Once a transfer has been submitted clients may poll for its status or cancel a request.

FTS can presently transfer files between grid storage elements offering an SRM interface. It is intended that storage elements with only gridftp interfaces will also be supported, but currently the parsing of `gsiftp://` URLs is broken.

FTS has two other important architectural concepts: channels and agents, together with a supporting authorisation infrastructure for different roles[4].

Channels Channels are an abstract FTS concept, which map from the storage elements of one site to that of another (N.B., channels are *unidirectional*). Each channel can be managed independently of all others (stopped, started, drained, etc.) and can have specific transfer parameters set for it, e.g., number of concurrent transfers. The "share" of a channel given to a specific VO can also be set, allowing priorities to be shifted between VOs.

Clearly a channel between dedicated sites will usually map to a specific production network pipe in HEP. Channels can however take a wildcard, $*$, meaning "any site", and channels can be configured as "$*$ to `mysite`" or even "$*$ to $*$" providing a final default channel.

FTS Agents FTS agents are daemons responsible for managing certain parts of the management of transfer requests.

FTS supports two types of agents, channel agents and VO agents. The VO agent is responsible for managing requests for a particular VO. At its most basic this consists of authorising a transfer request and then assigning it to an FTS channel.

However, hooks exist allowing VO agents to interact with catalogs, assign transfer priorities or perform any other VO specific tasks.

Once jobs have been assigned to a channel, the channel agent will start and monitor the transfers (applying a retry policy if necessary). The channel agent also balances requests on a channel between different VOs.

It should be noted that channel and VO agents can run on different machines from the FTS web service.

ROBUSTNESS TESTS

Hardware

For the purposes of robustness testing both services were setup on the same type of hardware, a single processor 2.80GHz P4 machines with 512MB of memory and a single IDE disk. The operating system was Scientific Linux 305. The servers were both setup to use MySQL backend databases (v 4.1.11) co-hosted with the transfer service.

Service Restart

Robust file transfer services should be able to cope with being restarted and carry on transfers from the last checkpointed state.

Both services were initialised with set of transfers, then shutdown. The behaviour of the services was examined and is summarised in Table 1.

Table 1: Robustness tests over service restart

Test	**FTS**	**RFT**
New transfer	PASS	PASS
Incomplete transfer	PASS	FAIL
Pending transfer	PASS	UNTESTABLE
New transfer after restart	PASS	PASS

Unfortunately, because of the global concurrency bug in RFT a test of the behaviour of a pending transfer was unable to be done.

After restart RFT failed to restart incomplete transfers.

Database Connection Loss

A test of the statelessness and robustness of transfer services can be examined by simulating loss of connectivity with the underlying database. The was achieved by using `iptables` to break the TCP connection from the service to the database.

As before, the services were seeded with set of transfers, then the connection to the database was broken. After a time the `iptables` rule was removed, to allow the service to reconnect to the database. The behaviour of the services was examined and is summarised in Table 2.

Table 2: Robustness tests over database loss

Test	FTS	RFT
New transfer	PASS	PASS
Incomplete transfer	PASS	FAIL
Pending transfer	PASS	UNTESTABLE
New transfer after reconnect	PASS	FAIL

As before, the global concurrency bug in RFT meant a test of pending transfers was unable to be done.

RFT was observed not to reconnect to the database, which meant it failed to restart incomplete transfers or accept new transfers. The RFT service had to be restarted in order for it to function again.

Mass Client Submission

To test the performance of FTS and RFT under load a mass client submission was tested. This was achieved by writing a small python harness to prepare and fork a large number of clients at the same time.

gLite FTS A number of gLite FTS command line clients (`glite-transfer-submit`) were invoked on a testing host simultaneously, each submitting a single 100MB file transfer request. As the FTS clients are written in C, and thus lightweight, only a single testing client was needed to be able to overload the FTS server. The fraction of clients which failed in their submission, and the time taken for all clients to exit (successfully or not) is plotted in Figure 2.

As can be seen, on the low end hardware used for FTS simultaneous submission up to 50 clients is possible. It should be noted that all the failures were clean and the FTS server itself did not crash. (Although the error messages reported by FTS clients are not for the faint hearted.)

FTS is limited in accepting connections by the tomcat server being unable to fork threads fast enough to deal with the connection.

Globus RFT The Globus RFT client supplied with GT4.0.1 is java based. As with most java tools running against the Sun JVM it suffers from long startup time and occupies a great deal of system resources. This made testing the Globus RFT difficult as it was hard to invoke a sufficient number of clients simultaneously, indeed on a single client host with 1GB of memory it was only possible to invoke about 10 clients.

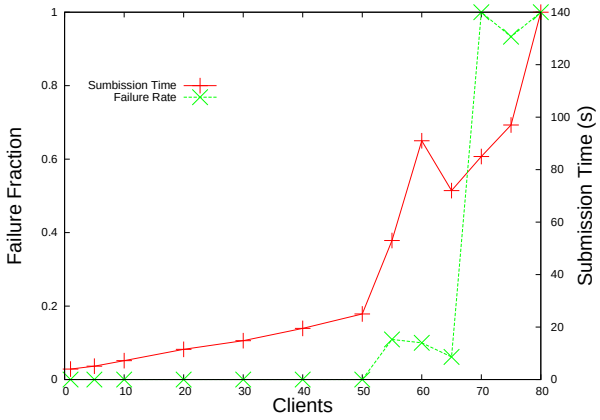

Figure 2: FTS with mass client submission.

However, by parallelising requests across a number of clients we could test up to 80 clients, but the JVM startup meant that even though these were invoked at the same time, they were considerably spread in time once they tried to contact the RFT service (over periods of up to 30s).

This made the RFT test quite different from the FTS one, where it was easy to overload the service with the lightweight C client, consequently the number of failures with RFT was always low (Figure 3).

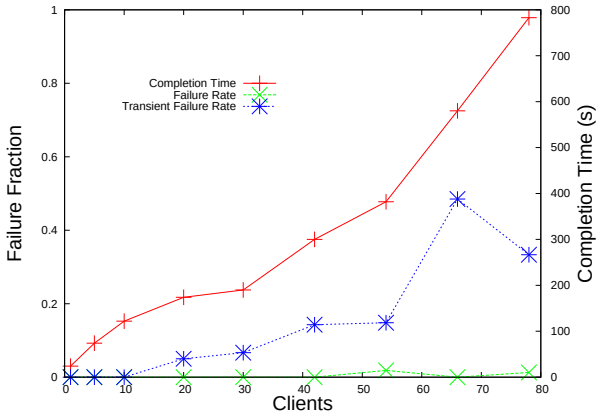

Figure 3: RFT with mass client submission.

However, when making large submissions, due to the concurrency bug in RFT, transient gridftp errors are common as the gridftp server used becomes overloaded (when 60 clients submit a minimum of 60 gridftp sessions are started), which is also seen clearly in (Figure 3).

Because the RFT client subscribes for status updates and only exits when the transfer has completed the time shown cannot be compared to that in Figure 2.

Additional Observations

No systematic attempt to measure the robustness of services over a long period. However, the following was observed:

- Globus RFT failed several times after 24 hours and required a service restart.
- On one occasion FTS suffered a channel agent failure, requiring the agent to be restarted.

In addition a bug was found in FTS which make it impossible to cancel transfers when the MySQL database backend was used.

CATALOG INTERACTION

Transfer services offering point to point data movement provide an important function to the grid. However, these are still quite low level services. In the grid paradigm a user may not know nor care where their data is. The location of their data will be held in a file catalog, which should be updated when new replicas of files are made.

This is achieved in different ways for FTS and RFT.

gLite FTS

For FTS a VO agent can take care of resolving file catalog entries to SURLs. After transfers have been completed new replicas are registered in the file catalog.

This added functionality turns the FTS into the gLite *File Placement Service* (FPS).

Globus RFT

For the RFT the whole service is wrapped in a higher level web service, the Data Replication Service (DRS). This interacts with the Globus Replica Location Service (RLS) catalog in much the same way as the FPS does.

FTS AND LCG SERVICE CHALLENGES

gLite FTS has been used extensively during the LCG service challenges, involving the transfer of hundreds of TB of data from CERN to various LCG Tier 1 sites, as seen in Figure 4.

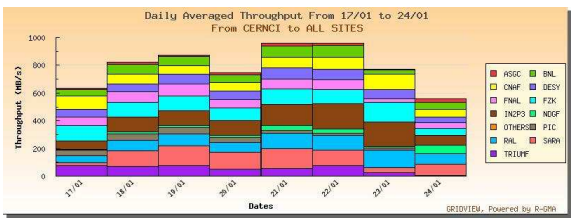

Figure 4: Data transfer rates during SC3 rerun – transfers are managed by FTS.

These tests, sustained over many days, have greatly helped improve the robustness of FTS in a production environment.

CONCLUSIONS

gLite FTS and Globus RFT both offer reliable and robust management of data movement in a grid environment, and represent a significant advance over direct client management of data transfers.

The channel architecture of FTS offers a sophisticated infrastructure for controlling data movement in a grid environment, particularly one with well defined data flows, such as LCG. Where such structured data flows do not exist the deployment of FTS channels may be less clear, but could still be useful.

This gives FTS a powerful ability to control transfers on a channel basis. For RFT concurrency can only be controlled at the global container level.

Neither service was wholly stable or bug free, however FTS had significantly less problems than RFT – probably as a consequence of stress testing during LCG service challenge 4. Error messages returned to clients need improvement and clients themselves could be made more robust.

RFT offers some useful features to clients not currently available in FTS, such as global deletions and a subscription service for updates.

Integration with data catalogs is at an early stage for both services, but will be the next important step in providing robust, transparent data movement services on the grid.

REFERENCES

[1] Allcock, William E., Ian Foster and Ravi Madduri *Reliable Data Transport: A Critical Service for the Grid*, http://www.globus.org/alliance/publications/papers/GGF11_RFTV-Final.pdf.

[2] Baud, Jean-Philippe and James Casey *Evolution of LCG-2 Data Management*, CHEP 04 Proceedings, Interlaken, Switzerland 2004.

[3] Burke, Stephen et al. *HEP Applications Experience With The European Datagrid Middleware And Testbed*, CHEP 04 Proceedings, Interlaken, Switzerland 2004.

[4] Kunst, Peter, Paulo Bandino, Gavin McCance, Ricardo Brito da Rocha *The gLite File Transfer Service: Middleware Lessons Learned from the Service Challenges*, CHEP 06 Proceedings, Mumbai, India 2006.

[5] McNab, Andrew *Web servers for bulk file transfer and storage*, CHEP 06 Proceedings, Mumbai, India 2006.

[6] http://egee-jra1-dm.web.cern.ch/egee-jra1-dm/FTS/

[7] https://forge.gridforum.org/projects/gridftp-wg

[8] http://sdm.lbl.gov/srm-wg/

[9] http://www.globus.org/toolkit/docs/4.0/data/rft/