The gLite File Transfer Service

Peter Kunszt Paolo Badino Ricardo Brito da Rocha James Casey Ákos Frohner
Gavin McCance

CERN, IT Department
1211 Geneva 23, Switzerland

Abstract

Transferring data reliably and in a robust manner is one of the most basic requirements in distributed systems. In large scientific Grid environments, the transfer services have to be not only scalable but also very flexible in adapting to the individual Grid site's policies. Also the policies of the virtual organizations making use of the Grid resources have to be taken into account such that the service remains easy to use and to manage. In this paper we describe the architecture and design of the EGEE project's gLite data movement services, i.e. the File Transfer Service FTS and the Data Scheduler DS. The FTS is a low level data movement service, responsible for moving sets of files from one site to another while allowing participating sites to control the network resource usage. This control includes the enforcement of site and usage policies like fair-share mechanisms on dedicated network links. The Data Scheduler is a high-level service that manages the individual transfers on the FTS links, organizing the FTS overlay network. This allows for additional management and customization of the topology on the level of each virtual organization while taking into account and enforcing the underlying policies of the network resource owners. This work is funded by Enabling Grids for E-sciencE (EGEE), a project of the European Commission (contract number INFSO–508833).

1. Introduction and Related Work

The reliable transport of data is one of the cornerstones for distributed systems. The transport mechanisms have to be scalable and efficient, making optimal usage of the available network and storage bandwidth. In production grids the most important requirement is robustness, meaning that the current high performance transfer protocols need to be run over extended periods of time with little supervision. The transfer middleware has to be able to apply policies for failure, adapting parameters dynamically or raising alerts where necessary. In large Grids, we have the additional complication of having to support multiple administrative domains while enforcing local site policies. At the same time the Grid application needs to be given uniform interface semantics independent of the site local policies, as is the promise of ubiquitous computing.

The EU-funded Enabling Grids for E-SciencE (EGEE) project's gLite middleware [**?**] has been designed such that these requirements can be addressed [**?**]. The service decomposition is according to the Grid layered architecture of services [**?**]. In gLite, each service offers a Web Service interface that is described with a standard WSDL document, adhering to the WS-I recommendation [**?**] wherever possible (due to legacy constraints this is often not an easy task). The emerging Open Grid Service Architecture OGSA interfaces and WSRF [**?**] are not being used yet as they have not been finalized and available at the time when the gLite FTS was already in use. Once the standards are final and well supported, there is no reason why the gLite interfaces could not be updated to be OGSA-compliant. A very important aspect in the design of the FTS was interoperability and exchangeability with other existing file placement services, including OGSA-enabled ones.

There are several file transfer mechanisms in use today in Data Grids. The most commonly used one is GridFTP [**?**, **?**], providing a highly performant secure transfer service. Of course mechanisms for transfer like `http(s)`, `(s)ftp` and `scp` are also common and in use in Data Grids. There exist also some enhanced variants of these 'classical' protocols which some communities may prefer like `bbftp` [**?**]. The Grid middleware makes use of these protocols for the file transfers while providing additional reliability and management features. The Globus Reliable File Transfer service RFT is fully OGSA compliant and built on top of GridFTP [**?**]. It has retry mechanisms to address especially network problems, greatly improving the reliability of vanilla GridFTP. The SDSC Storage Resource Broker SRB [**?**] provides a uniform interface to data in a distributed environment, storing a large set of data attributes in its metadata

catalog (MCAT). The SRB can also make use of several different transfer protocols to actually perform the transfer. The Condor Stork data placement service [**?**] acts as a planner and scheduler for data transfer jobs. It has the capability to also translate between different protocols if the source and target data stores have no matching protocol support. The Storage Resource Manager SRM interface, which is being standardized through the Global Grid Forum [**?**], also provides a data movement facility called `srm-copy`, which in most implementations makes use of GridFTP to perform the transfer on the user's behalf. Its advantage is the detailed knowledge of the local storage availability and bandwidth.

The gLite FTS in its functionality is similar to Globus RFT and the SRM copy services since it manages only point-to-point movement of files. Condor Stork is also sometimes used in this mode of operation. Indeed, the FTS can make use of the RFT and SRM copy facilities instead of direct GridFTP, or it may also call upon Stork or SRB to perform the operation. Or the other way round, SRB or Stork might use the FTS. This flexibility in layering allows the FTS to interoperate with other Grid file transfer services in a straightforward way.

What distinguishes the FTS from these related services is its design for policy management. The FTS acts as the resource manager's policy enforcement tool for a dedicated network link between two sites since it is capable of managing the policies of the resource owner as well as of the users (the VOs). The FTS has dedicated interfaces to manage these policies. An interesting comparison has been reported in [**?**].

Although it is beyond the scope of this paper, it is worth mentioning that the FTS is extensible such that upon certain events user-definable functions can be executed. The VOs may make use of this extensibility point to call upon other services when transfers complete (e.g. register data in catalogs) or to change the policies for certain error handling operations (e.g. the retry policy).

2. Addressing Requirements

In this section we summarize the most important challenges and requirements for a Grid data transfer service and how they are addressed by the FTS.

2.1. Policy Enforcement

The virtual organizations and the resource owners need to enforce many different (and sometimes conflicting) policies. Security aspects are very important as well as fair share mechanisms, accounting and auditing. The requirements may differ considerably; not only between VOs and resources but also from one VO to the next and between the different resource owners. The policies also need to be enforced properly. The resource owner needs to have maximal flexibility to declare how the resource is to be used by the different VOs for the particular resource in question, while the VOs need to be able to define their specific policies independently of each other. In large scale Grids there are several VOs sharing the same resources, expecting to get their share of the resource on which they agreed on.

If we are to scale to support several VOs for the same network and storage resource, then it is impractical to have to start a new service for every new VO that gains access to the resource. The service has to scale in terms of deployment support, meaning that at a certain Grid site, the site administrator must not have additional work because of the support of a new VO. The FTS was designed to allow VO management to be lightweight. Many other Grid transfer services operate best for single-VO deployments, or need to have new instances started for every VO while lacking the ability to coordinate between those instances.

2.2. Heterogeinity

The Grid is heterogeneous by definition. There are different storage systems and different site policies allowing the usage of the resources. In the future there may be also several mechanisms to provide network bandwidth through dynamic reservation and allocation, and the Grid middleware would be expected to make use of them all and to their best ability to maximize performance. In addition, multinational projects operating on many continents are making use of Grid middleware coming from a variety of middleware providers that do not always make use of standard interfaces.

Of course for the FTS we had to choose which technologies to support and interoperate with, but the architecture is extensible so that additional support for new technologies can easily be added. The FTS is also simple enough so that it can be used by other high-level services.

2.3. Robustness

Any distributed system is inherently fragile. Many things can go wrong in many different ways. It is important that the relevant error conditions are exposed. However, due to the heterogeneous nature of the Grid it is often difficult to trace particular errors back to their source. If several services from different middleware stacks are participating in a particular failed operation, it is unlikely that all of them use the same error reporting mechanism allowing the user to retrieve a full trace of the problem.

The FTS has extensive monitoring and error reporting mechanisms that may be used to retrieve logs or detailed errors by higher-level services.

2.4. Security

The network as a resource needs to be secured so that only authorized users are able to make use of it. Any transfer service needs to account for its usage by the user community that has authorization to do so.

In addition to standard authentication and accounting of usage, the transfer of data may also mandate the transfer of system metadata like data access control lists from one site to the other. The FTS does not transfer any such security metadata but the hooks are available to extend it such that it could. The semantics of secure data replication may be different from one VO to the next, so this is an extensibility point for the service that can be individualized for each VO.

3. Service Components

3.1. Channels

The FTS performs file transfers on channels (network links). An FTS instance serves a configurable set of channels. Every channel is unidirectional, i.e. it is intentional that different FTS instances may serve the two directions of a network link between two sites. Usually the receiving site configures the corresponding channel, as it is shown on the image to the right. In short, we define channel as a logicl unit of management the represent the specific (perhaps dedicated) directed network pipe for transferring files between two sites. We distinguish between *Dedicated* and *Non-dedicated* channels.

Dedicated channels correspond to real point-to-point network links. In the best case these are network links where the FTS is set up to have full control over the reserved bandwidth and the capacity is not shared with any other service. If some sharing exists, the parameters can still be tuned empirically to achieve an average high bandwidth but it is of course more difficult to maximize bandwidth usage at all times in this case. There can be many channels on the same dedicated link in which case the FTS can provide full control on the sharing of the link among the channels being set up. Therefore, the operational purpose of dedicated channels is to be able to set parameters like the number of concurrent transfers, the TCP buffer size and the number of TCP streams, in order to perform all the transfers with the configuration that can guarantee a certain throughput between two sites. The percentage of current transfers of each virtual organization having access to dedicated channels usually depends on service level agreements with the network managers in question. Dedicated channels only accept transfer jobs where all source and destination pairs in a transfer job are from the channel's source to its destination (see below for the definition of transfer jobs). Usually the FTS runs two instances one for each direction of a dedicated channel.

Non-dedicated or "catch-all" channels may be defined to use the FTS as a simple collector of transfer jobs, to be executed on the network links as defined in the channel configuration. They may not be coupled to a physical network connection at all. In this mode the FTS operates exactly like RFT or Stork, where its additional bonus are simply its extensibility and interoperability features.

The FTS takes the channel as its unit of network to manage. If the channel is defined to be the open network then this will be the management unit. Using "catch-all" channels allows you to limit the number of channels you need to manage, but also limits the degree of control you have over what is coming into your site (although it still provides the other advantages like queueing, policy enforcement and error recovery). Even if the channel definitions are usually coupled with the network layout, it's worth to mention that the FTS doesn't force routing, that is always provided by the underlying network.

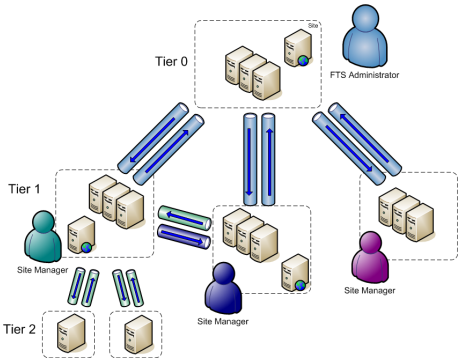

Figure 1. The dedicated channels layout in a multi-tier hierarchy

3.2. Transfer and File States

The transfers managed by the FTS are all asynchronous, i.e. the client submits a *transfer job*, which may contain a list of files (with proper source and destination qualifiers) to be transferred. The FTS assigns a unique string identifier to the job if it is accepted. The states of the whole job and the states of the individual files (that can be tracked using this ID) are not the same: See Figure 2 for the job states and Figure 3 for the individual file states. The reason for the difference is that there may be many files to be transferred in a single job.

The clients see only the FTS Web Service interface and its associated API and convenience command line tools. In gLite we also provide a web browser interface to the FTS.

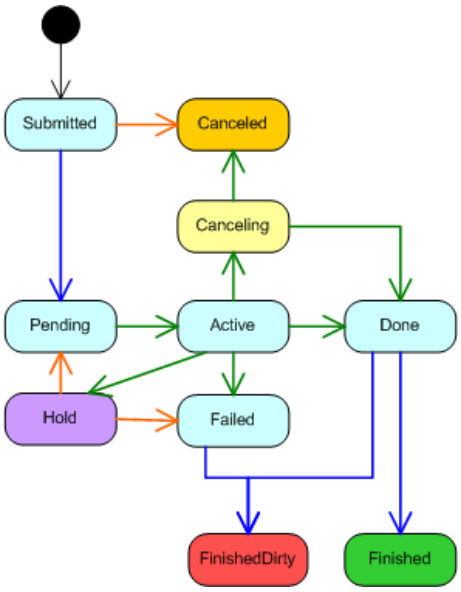

Figure 2. The states that are possible for a file transfer job. A job may have many files.

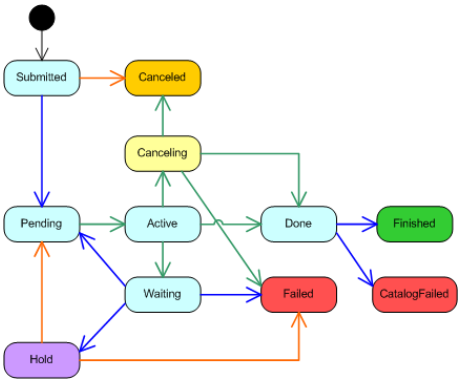

Figure 3. The states that are possible for a file that is being transferred.

4. Service Architecture

The gLite middleware architecture is based on principles of Service Oriented Architecture [**?**], with an emphasis on manageability and extensibility.

The gLite service decomposition for data movement services is shown in Figure 4. The components are described in more detail below.

FTS Web Service interface The Web Service interface is the component to which the FTS clients actually connect. It receives the transfer requests from the user directly; or indirectly via the Data Scheduler. The deployment of Web Service interfaces depends on what the needs of the VO are. It is also possible to have a service at each site, connecting to the same Transfer Queue.

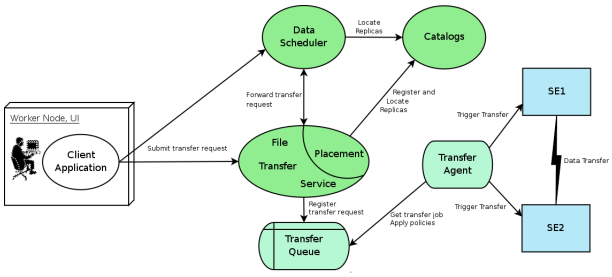

Figure 4. Data Scheduler (DS) and FTS service components.

FTS Transfer Queue The transfer queue is persistent and is not tied to a site, but rather to a (set of) connections, wide area network channels (links). It keeps track of all ongoing transfers and may also be used to keep a transfer history, for logging and accounting purposes. The queue stores all persistent information of the FTS, including the file and job states.

FTS Transfer Agents The agents are responsible for actually triggering the transfers and performing all the actions necessary based on the states stored in the Transfer Queue. The transfer agents may be associated to the network resource (i.e. the channels) and/or the VOs. Agents may only simply get the next transfer job out of the queue, but may also reorder the queue according to site and VO policies, perform retries, etc. The Agent provides the actual FTS functionality.

4.1 Data Scheduler

The Data Scheduler (DS) is the high-level service that keeps track of most ongoing WAN transfers inside a VO. From the VO's point of view it is a single central service. It may actually be distributed and there may be several of them, but that depends on the implementation. The DS schedules data movement based on user requests.

Users and other services may request the DS in order to move data in a scalable, coordinated and controlled fashion between two SEs. The DS has the following components:

VO Policies Each VO can apply policies with respect to data scheduling. These may include priority information, preferred sites, recovery modes and security

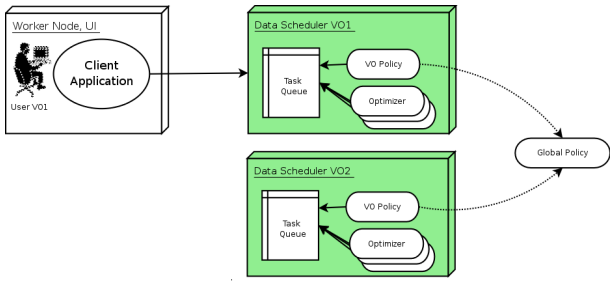

Figure 5. The Data Scheduler

considerations. There may be also be a global policy which the VO, by its policy, may choose to apply. Global policy is usually fetched from some configurable place. The same global policy may apply for many VOs.

Data Scheduler Interface The actual service interface that the clients connect to, exposing the service logic. All operations that the clients can use are exposed through this service.

DS Task Queue The queue holding the list of transfers to be done. The queue is persistent and holds the complete state of the DS, so that operations can be recovered if the DS is shut down and restarted. This queue is the heart of the DS, all services and service processes operate on this persistent queue.

Optimisers A set of modules to fill in missing information (like the source to copy from), or modifying the details of a request based on some algorithm (such as the protocol or the source replica to be used). It also is responsible for choosing the optimal routing.

4.2 File Transfer Service

The FTS receive requests either through their Web Service interface directly from the clients or indirectly through the Data Scheduler. If the request comes from the user directly and cannot be managed by the local service instance, it may be forwarded to a known Data Scheduler for further processing.

Once a request is accepted, it is simply put into the associated Transfer Queue. If the request contains logical names to be resolved, a catalog may be contacted and only the resolved names are put into the persistent queue. File placement requests also involve catalog updates after a successful transfer operation.

4.3 Transfer Queue

The transfer queue is just a set of tables, kept in a relational database. The queue in itself does not provide an interface or a service. The queue is manipulated directly by the Transfer Agents and the FTS Web interface.

4.4 Transfer Agents

The Agents are very modular, built as a container for many *Actors* that can be executed periodically. Each Actor will operate on the associated Transfer Queue, executing a simple operation.

The Transfer Queue provides different views for the Agents to operate on. There are two kinds of views: channel and VO views. The channel view will expose all transfers relevant for a given channel, the VO view will show all transfers associated with a certain VO.

Standard Actors will resolve names, apply VO policy, channel policy, monitor the state, trigger and monitor the actual file transfer and change the state in the queue if the state of the transfer has changed. For the discussion below, it is important to define at least two Actors: The *Channel Allocator Actor* will assign a file to be transferred through a certain channel. The *Transfer Trigger Actor* will actually work its way through triggering the transfers based on the parameters set for the channel configuration (number of parallel transfers, etc).

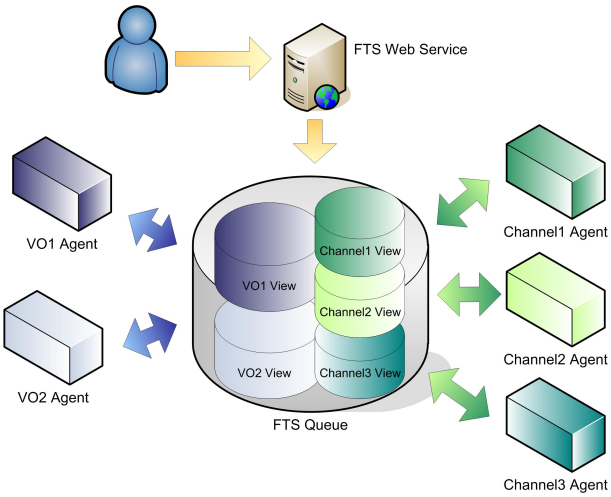

Figure 6. The File Transfer Agents

In terms of channels, each channel has its own queue in a File Transfer Queue, which is managed by at least one File Transfer Agent (see Figure 1).

The following channel states are defined (explained by referring to the two Actors defined above):

Active The Allocator is looking for work to assign to a channel *and* the Trigger is looking for work in that channel to be put on the wire.

Drain The Allocator will *not* add anything new to a channel *but* the Trigger will continue to serve jobs that have been assigned to its channel. The effect is to drain all pending transfers from the channel.

Inactive The Allocator will assign work to a channel *but* the Trigger will *not* put any more jobs on the wire. This is used by a sysadmin to empty the network. Note that jobs currently active on the wire will complete.

Stopped Neither the Trigger or Allocator will do any work. Nothing will be assigned to the channel and no work will be put on the wire. Existing jobs on the wire will complete.

Halted A serious error has been detected on the channel (e.g. there have been a certain number of 'sequential' failures on the channel in the last few minutes) and that the channel has been automatically stopped by some monitoring process. When a channel is halted an alarm should be raised to alert an operator for manual intervention. This state is designed to prevent the transfer queue draining in case of problems.

5. Experience

The gLite implementation provides the FTS with all its subcomponents (Task Queue, Agents) but no Data Scheduler (yet). The FTS has been deployed on the LHC Computing Grid's Service Challenge infrastructure, connecting CERN with all potential Tier1 sites. The Service Challenges aim to test the infrastructure available for the LHC data distribution and push its to its limits. The FTS was used to actually perform the massive amounts of sustained file transfers (see Figure 7). The FTS's monitoring and error reporting features allowed to debug the issues seen with the storage resources at some sites and also the network resources. The Service Challenges also exposed a series of weaknesses in the FTS itself that only could be seen in large scale deployments. For example there were various erroneous behaviors that only occured in very specific situations that are too rare on a test setup but have a considerable impact at a large scale. Through its configurable retry policy and the flexible configuration and administration options it is possible to tune the system such that such sustained large-scale transfers are indeed possible also over unstable and unreliable network links. For a detailed description of the Service Challenge experience, see [**?**].

The figure above shows the result achieved during the ServiceChallenge in January 2006, where more that 16,000

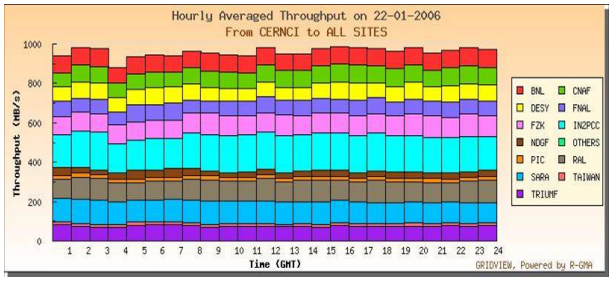

Figure 7. Throughput from CERN to several other sites on Jan. 22nd 2006.

transfer jobs have been processed by the FTS, for a total of more than 787,000 file transferred from CERN to the involved Tier1 sites.

6 Current and Future Work

The current Service Challenges (SC4) started in April 2006. They aim to achieve the same and higher throughput as with previous Service Challenges (SC3), sustained for much longer periods of time. Therefore the emphasis is put on the stability and robusteness of the involved service components. They should be able to run without interruptions, recover automatically from failures, etc. In order to achieve the targets set by SC4, the FTS monitoring needs to be improved. Error conditions need to be detected with high accuracy so that the proper policies can be invoked to deal with them. For example, if problems are encountered on a channel, the FTS might act proactively, either by throttling the transfers or signal alarms for manual interventions. In addition for SC4, support for the new version of the SRM interface (2.1.1) will be introduced. Further work includes improvements in the scheduling, retry and proxy renewal logic as well as the catalog interactions. We also plan to add possibilities to customize the retry logic and catalog plug-ins.

7 Summary

Transferring large amounts of data in a production environment sustained over an extended period of time is feasible using Grid services that are built based on the standard Grid layered architecture. The implementation of the transfer service presented here - the gLite FTS - is able to deliver the necessary performance while being easy to manage.