File Transfer Software SC3

Gavin McCance – JRA1 Data Management Cluster

LCG Storage Management Workshop
April 6, 2005, CERN

Outline

- Requirements and status
- Design and components
- How to plumb your existing frameworks onto it

Reliable File Transfer Requirements

- **LCG created a set of requirements based on the Robust Data Transfer Service Challenge**

- **LCG and gLite teams translated this into a detailed architecture and design document for the software and the service**
  - A prototype (radiant) was created to test out the architecture and was used in SC1 and SC2
    - Architecture and design have worked well for SC2
  - gLite FTS (“File Transfer Service”) is an instantiation of the same architecture and design, and is the candidate for use in SC3
    - Current version of FTS and SC2 radiant software are interoperable

Requirements Summary

- **From the LCG requirements document: https://edms.cern.ch/file/490347/3**

Functional	R1.1	R1.2	R1.3	R1.4	R1.5	R1.6	R1.7	R1.8	R1.9	R1.10
	YES	YES	YES	(NO)	YES	YES	YES	YES	(NO)	PARTIAL

Security	R2.1	R2.2	R2.3	***Scalability***	R3.1	R3.2	R3.3	R3.4
	YES	YES	YES		To demonstrate			

Manageability	R4.1	R4.2	R4.3	R4.4	R4.5	R4.6	R4.7
	YES	To demonstrate		YES	PARTIAL	PARTIAL	YES

Monitoring	R5.1	R5.2	R5.3	R5.4	R5.5
	NO	NO	YES	PARTIAL	YES

Scheduling	R6.1	R6.2	R6.3	R6.4	***User interface***	R7.1	R7.2	R7.3
	YES	(NO)	(NO)	(NO)		YES	YES	PARTIAL

- **Core functionality mostly done: some improvements for SC3**
- **Our remaining focus is on monitoring, manageability and service stability**

Notes on architecture

- **The FTS is a service which provides point to point movement of SURLs**
  - Aims to provide reliable file transfer, and that's it!
  - Does not do 'routing' (e.g like Phedex)
  - Does not deal with GUID, LFN, Dataset, Collections
- **These are all higher level concepts and can be dealt with by higher level software**
- **It does provides a pluggable agent-based architecture where some of this higher level work could be done, if required**
  - VOs could provide their own agents to do VO specific things (e.g cataloging, different retry policies)
  - As an example of this, gLite will aim to provide basic agents that integrate the FTS with the rest of the gLite software stack (especially the catalogs)
    - this is what people mean when they talk about the gLite FPS "File Placement Service".

To be clear…

- The substance of this presentation is about the FTS (i.e. the basic SURL->SURL transfer system) and how to plumb your own frameworks onto it
- FTS code is now stabilised, we're getting good experience running with it
- The example extra agents that plumb the FTS into the rest of the gLite framework (the "FPS" agents) need some work and are not tested

Software components

- **Decoupled stateless agents:**
  - Allocator agents
  - Transfer agents
  - Retry agents to decide what to do with failed files (depending on the failure class).
- **Database: stores all the state**
  - Currently runs on the CERN Oracle service
- **Web-service:**
  - wrapper over database, secure job submissions and status querying via the interface, channel service management
  - decoupled from the agents that actually do the file movement
- **Command line:**
  - command line clients (job management and service management)
- **GUI:**
  - job monitoring GUI and service monitoring GUI running against web-service or DB directly

Service management

- **“Channels” are dedicated network pipes for production use, e.g. CERN->RAL distribution**
  - The service can be configured to assign work to and service work on a number of channels
- **Management functions to control the state of channels (without losing work assigned to that channel)**
  - Change channel state (e.g. Active, Inactive, Drain)
  - Throttle network usage
  - Manual intervention functions to recover jobs that have been running on a problematic channel

Testing status

- **Presented in more detail at the Lyon LCG service challenge meeting…**

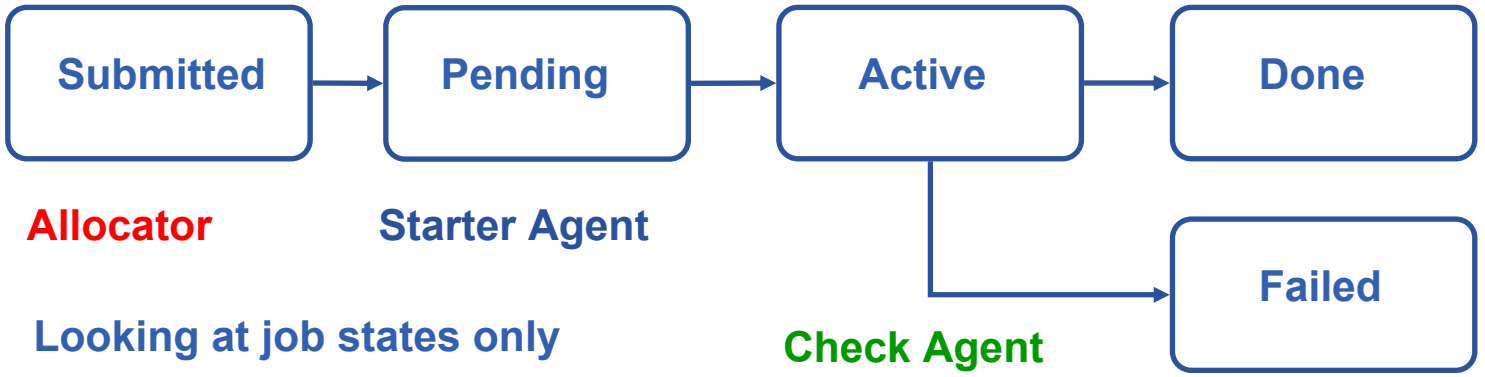
- **Validation tests have proved ~positive**
  - Trying to run as close to SC2 mode as possible to make sure that we catch issues early – 24/7 running on (small) subset of SC2 setup, running in parallel. Best effort problem resolution.
  - We plan to continue this testing and increase the scale of it as we head to SC3
  - Issues already found and understood / fixed.
  - Maintaining a close interaction with LCG service challenge team to ensure the experience of SC1 and SC2 feed into the software

FTS Design concepts

- **System is driven by state machine and a set of 'agents' working on their own part of the state machine**
  - The state is kept in a database.
  - Agents should be decoupled
  - Agents should be as stateless as possible, and always able to recover the true state should they fail

- **Multiple file transfers grouped into a single job**
  - Overall job state ("is it done yet?") is a function of the individual states of the job's constituent files

Somewhat simplified design



e.g. three decoupled components (agents):

- Allocator: looks for jobs in '**Submitted**' - assigns a suitable channel to them (based on the files in the job) and then sets the job state to 'Pending'.
- Transfer agents:
  - looks for jobs in '**Pending**' where the channel name is one it is responsible for. If resources are available, start the copying: set to Active.
  - Checks 'Active' jobs: set to '**Done**' or '**Failed**' depending on result of job's transfers

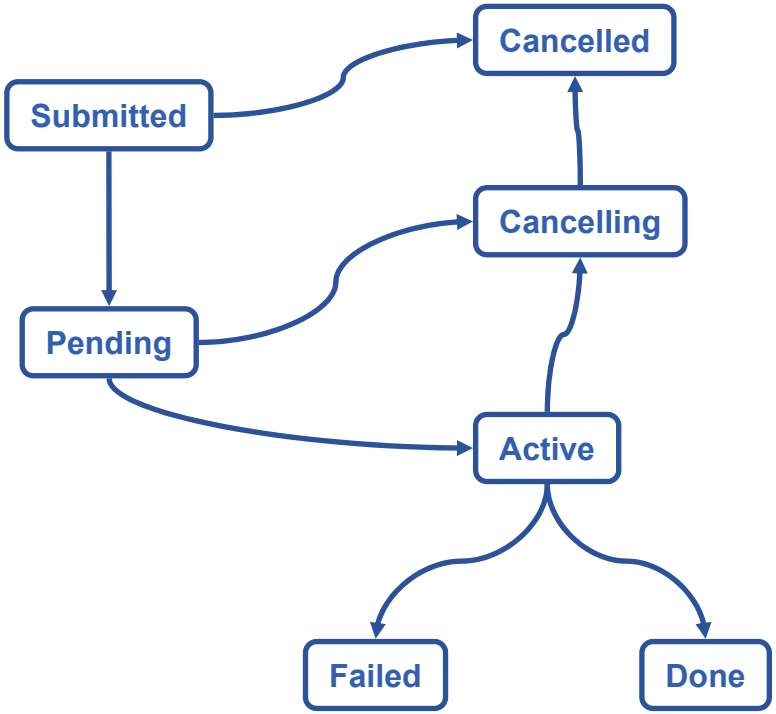
How to plug onto the FTS

- **The FTS only moves SURLs**
  - Someone has to work out which SURLs get copied
  - Someone has to update the appropriate catalogs, etc after they have been copied

- **We define two state transitions in the framework where these actions could occur**
  - Resolution (or allocation): which SURLs and which channel?
  - Post-processing: do something after the copy

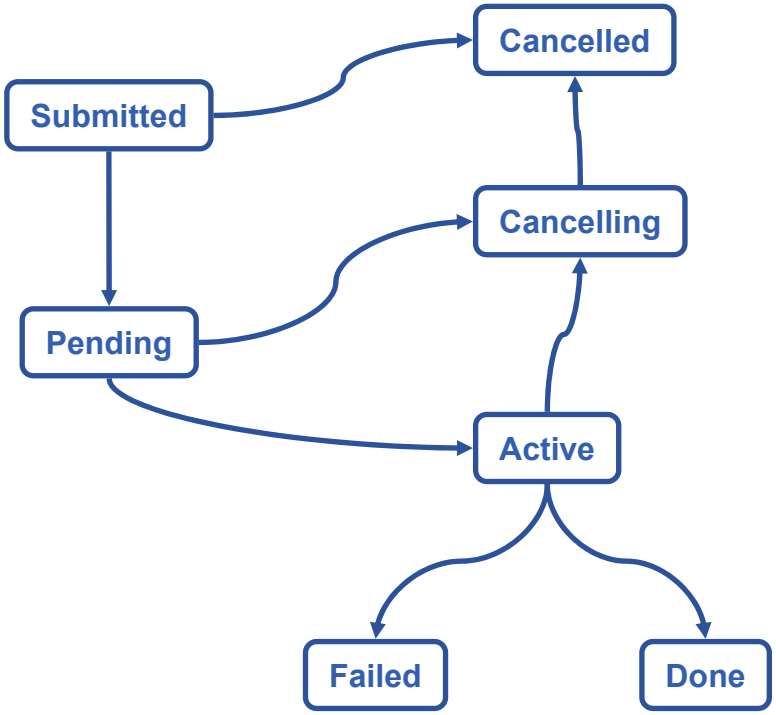
  - The details of the actions that happen are likely to be VO specific

How to plug onto the FTS



- Parts of state diagram handed by the File Transfer Service agents.
- FTS: “Allocation” is just a basic channel assignment
- Illustrative only: looking at the aggregate job states only. Need to consider file states as well.
- Wait and Hold states are omitted for clarity

How to plug onto the FTS



- NB: Job states only.
- Some failure modes omitted for clarity

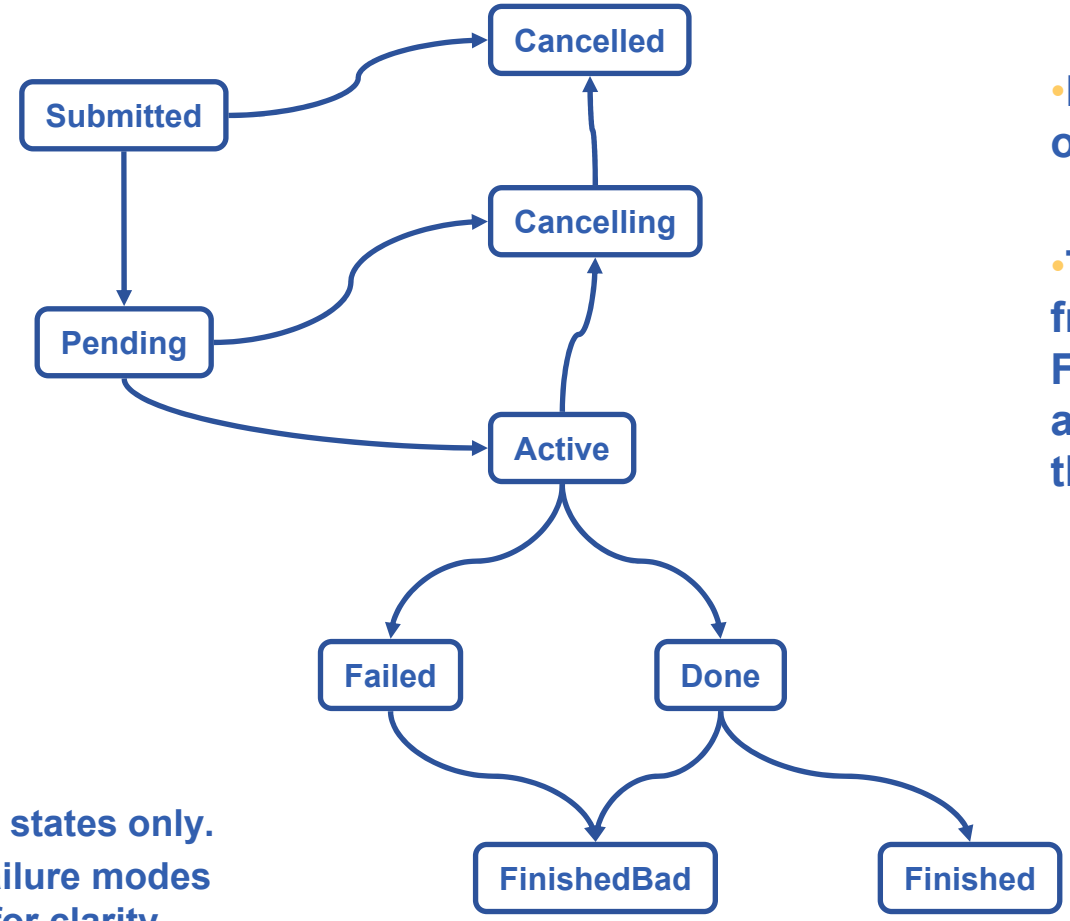
- Step 1: Resolution and allocation

- "Allocation" can be more complex
- *Rewrite* the FTS allocator to do what you need it to do

- **EXAMPLE:** gLite FPS, you submit jobs containing LFNs rather than SURLs and the new allocator uses the gLite catalog to resolve those files to SURLs

- The rest of the system works as before

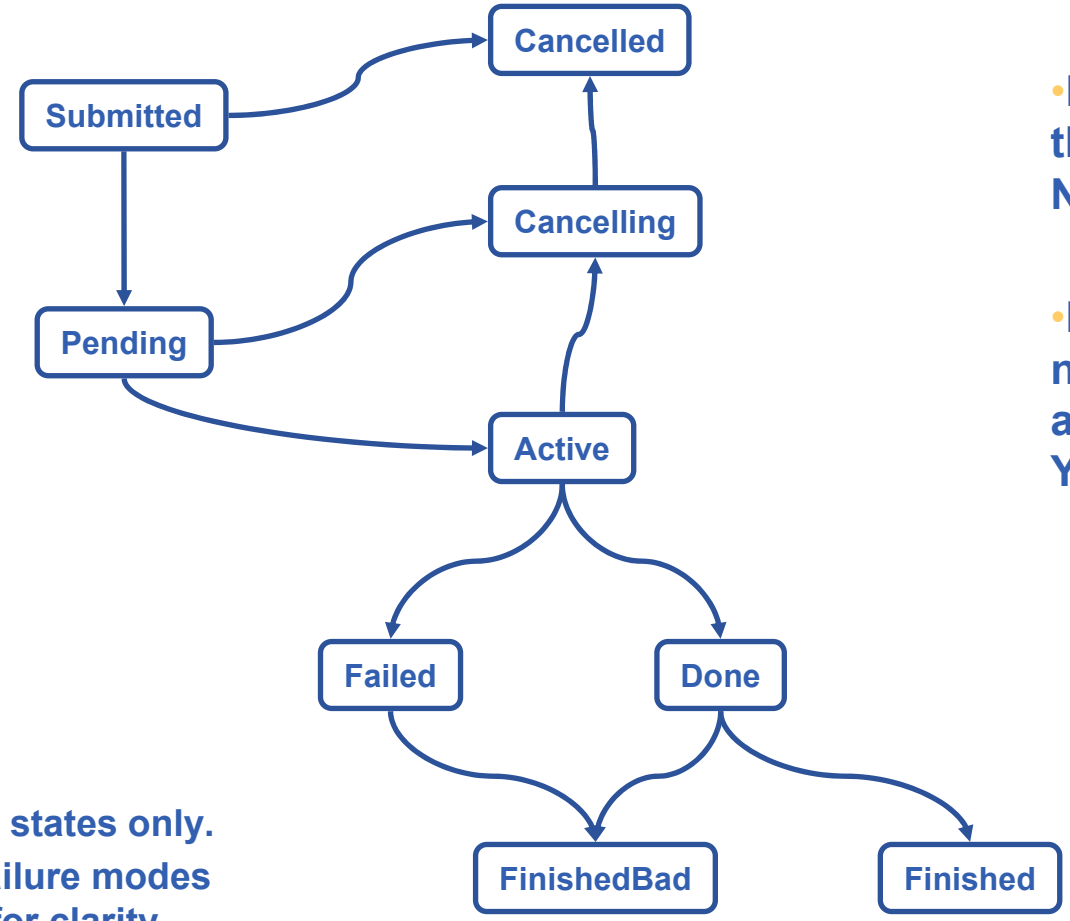
How to plug onto the FTS



- Step 2: Post processing
- Need two new states and one additional agent
- The agent moves state from Failed / Done to Finished / FinishedBad and does *something* along the way

- NB: Job states only.
- Some failure modes omitted for clarity

How to plug onto the FTS



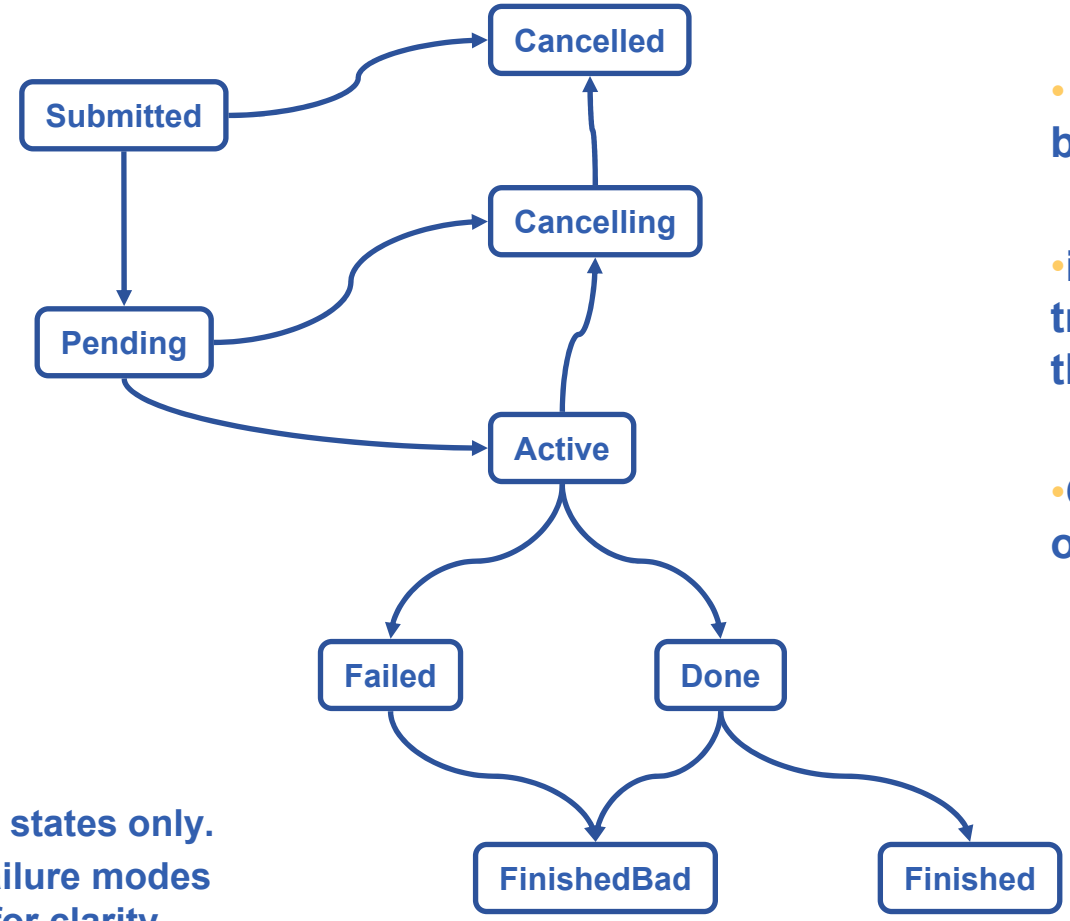
- Step 2: Post processing

- Finshed: all files transfers in the job finished OK.
No need to check.

- FinishedBad: there was a non-resolvable problem with at least one of the file transfers.
You should check the job.

- NB: Job states only.
- Some failure modes omitted for clarity

How to plug onto the FTS



- NB: Job states only.
- Some failure modes omitted for clarity

- Step 2: Post processing
- **EXAMPLE:** gLite FPS does the basic catalog update.
- i.e. for all file successfully transferred in the job, it updates the gLite replica catalog.
- Clearly, more complex operations are possible.

How to plug onto the FTS

- **How to make your framework work with the FTS**
- **Three options:**
  - Only use the FTS (i.e. just the file transfer) and do everything else in your existing frameworks
  - Use the agent framework we provide to do some of the operations
  - Write your own agent

Agent framework

- **For both of the steps, we can provide a basic fill-in-the-blanks agent**
  - It will load a library that you write, that does whatever action you need
- **Suitable for simple catalog resolution and update**
  - This approach is being used for the gLite “FPS”.
- **Pros:**
  - It’s simple to write
  - No chance of interference with core transfer system
- **Cons:**
  - Limits the information you can have to do your action. i.e. no arbitrary access to the underlying DB tables

DB access

- **For both of the steps, you write your own agent**
  - It talks to the DB and does what you need
  - But.. It should restrict itself to the allowed state transitions, so that it doesn't interfere with the core FTS software
- **Pros**
  - Much more flexible, you have all the information available
  - You can add your own DB tables if you need them
- **Cons**
  - Chance of breaking the core transfer system if you mess up the states
  - Any future FTS schema changes may affect you

Summary

- **Core FTS software is in good shape**
  - On its way to becoming software that can provide a manageable, reliable service
- **Stress-testing underway**
  - Gaining operational experience with it early
  - Increasing as we head to SC3
- **We are now understanding how the experiment frameworks can plug onto it**
  - Interested to discuss further and work with experiments

Backup slides

Functional Requirements

#	Requirement Description	Current status
1.1	Asynchronously and reliably copy files	YES
1.2	Recovery from partial/failed transfers	YES
1.3	Report to user the current progress	YES
1.4	Allow user to prioritise their requests	NO. Not for SC3.
1.5	Multi-file requests as single atomic unit	YES
1.6	Access files over a cluster of data servers	YES
1.7	Handle files on disk, accessing via gridFTP	YES
1.8	Handle files via SRM (gridFTP as TURL)	YES
1.9	Other protocols should be usable	NO. Not for SC3.
1.10	Interact with mass storage when scheduling requests	PARTIAL Improvements for SC3.

Security Requirements

#	Requirement Description	Current status
2.1	Require user to be authenticated to use service	YES
2.2	Require user to be authorized to use service	YES
2.3	Use user's proxy to do the transfer	YES

Scalability Requirements

#	Requirement Description	Current status
3.1	Scale to 500MB/s continuous point-to-point	Still to demonstrate
3.2	Scale to 1GB/s peak point-to-point	Still to demonstrate
3.3	Scale to simultaneous transfer of 100 files point-to-point	Still to demonstrate
3.4	Be able to store 1000 multi-file requests in the “pending transfer” queue per point-to-point connection	Still to demonstrate

- Test setup is currently running which is incrementally stressing the transfer software
- Setup is as close to SC2 production setup as possible, and will be stepped up in preparations for SC3

Manageability Requirements

#	Requirement Description	Current status
4.1	Less than 1 day to setup and configure software at a site	YES
4.2	Require less than 1 FTE per 100 servers to manage	To be demonstrated
4.3	Be able to run for weeks unattended	To be demonstrated
4.4	Be able to be restarted and carry on with previous or ongoing jobs	YES
4.5	Require X interventions per month where X ~ 0 (related to #1.2)	PARTIAL. Some automated recovery for SC3.
4.6	Admins should be able to manage channels easily (pause, throttle, change params, …)	PARTIAL. Improvements for SC3.
4.7	System should be deployable on production setup (managed hardware, managed DB, …)	YES

Monitoring Requirements

#	Requirement Description	Current status
5.1	Bandwidth monitoring	NO. To do for SC3.
5.2	Channel performance monitoring	NO. To do for SC3.
5.3	Information for individual jobs	YES
5.4	Dynamic information on a monitoring web page.	PARTIAL. Improvements for SC3.
5.5	Information available as command-line tools	YES Improvements for SC3.

Scheduling Requirements

#	Requirement Description	Current status
6.1	Support simple FIFO scheduling	YES
6.2	More complex scheduling algorithms	NO. Not for SC3. (although it will be able to use srm-cp).
6.3	Allow for bandwidth quotas (MB/s , GB/day)	NO. Not for SC3.
6.4	Allow for per-VO bandwidth quotas	NO. Not for SC3.

User Interface Requirements

#	Requirement Description	Current status
7.1	Allow for user-level submission, monitoring, management and monitoring of transfer requests via command line.	YES Improvements for SC3.
7.2	Allow for user-level submission, monitoring, management and monitoring of transfer requests via a web-based GUI.	YES Improvements for SC3.
7.3	Allow for monitoring and management of the service via a GUI.	PARTIAL Improvements for SC3.