Mover Queues and Transfer Parameters

13th April 2010
University of Wuppertal
Xavier Mol <xavier.mol@kit.edu>

Outline

I. Mover queues
   1. Why several queues?
   2. How to define queues?

II. Transfer parameters
   1. (gsi-)dcap
   2. gsiftp a.k.a. gridftp
      1. passive gridftp a.k.a. gridftp2
   3. SRM
   4. xrootd

Why several queues?

- Different transfer protocols often serve different use cases
  - dcap: (possibly) non-continuous, non-sequential (i.e. random access) transfers for hours
  - gridftp: normally very fast but also stressful for the system
  - Combined in one queue the dcap transfers will block gridftp transfers (which will timeout)
  - Normally the amount of dcap transfers on a site is much higher than for gridftp

Why several queues?

- So the simplest solution is to distribute transfers with different usage pattern over different **queues** (per pool)
- Moreover, every queue may be configured seperately regarding their capacity and timeouts
- In combination with the pool selection unit (psu) transfers can even be distributed over different pools (will not be discussed here)

How to define queues?

- Originally through editing the batch files for pools and doors
  - Single pools may also be configured by means of their respective *poollist* files
- However configuration with the dCacheSetup file is advised (less error prone)
  - Every supported transfer protocol offers also an dedicated variable for definition of the queue name

How to define queues?

- There are neither restrictions nor conventions for queue names
- Errors (e.g. inconsistent names or typos) are silently ignored
  - Whether a queue is present or not does not influence mover distribution
  - Every pool always has a default queue where movers without suitable queue will end up in
    - The first queue mentioned for the variable `poolIoQueue` is the default queue

How to define queues?

- In pool setup file
  - capacity of the queues

    ```
    mover set max active <int> -queue <qname>
    ```

  - timeouts of the queues

    ```
    jtm <qname> -sleep=<int> -total=<int>
    ```

- In dCacheSetup file

  `dcapIoQueue`, `gsidcapIoQueue`, `gsiftpIoQueue`, `xrootdIoQueue`

- If value of `remoteGsiftpIoQueue` is not set then movers will share with gsiftp-io-queue

Transfer parameters

- As a general remark:

  Most parameters for tuning of transfer performance or behaviour are not meant to be changed by (unexperienced) administrators! In order to ensure this these parameters are not documented.

Transfer parameters

- Once again configuration can be done by editing batch files for the respective doors of the protocol
- But this method is old-fashioned and discouraged, too
- Use the dCacheSetup file to alter the values for the transfer parameters
- Except as noted otherwise all time values are given in seconds

(gsi-)dcap parameters

- Define the name of the queue used for dcap movers
  `dcapIoQueue=`*`<string>`*
- Define the capacity of the dcap-queue (per dCache pool)
  `dcapMaxLogin=`*`<int>`*
- The client may specify a queue to use
  `dcapIoQueueOverwrite=`*`(denied|allowed)`*
- For gsidcap just prepend `gsi` to the variable names

gsiftp a.k.a. gridftp param.

- Set the interval for sending performance markers

  `performanceMarkerPeriod=`*`<int>`*

  – Performance markers are sort-of a heart beat of an active transfer

gsiftp a.k.a. gridftp param.

- Define timeouts for reactions from either PoolManager, PoolDomain or PnfsManager

```
gsiftpPoolManagerTimeout=<int>
gsiftpPoolTimeout=<int>
gsiftpPnfsTimeout=<int>
```

gsiftp a.k.a. gridftp param.

- Setting maximum and default stream count for a file transfer.

  `gsiftpMaxStreamsPerClient=<int>`

  `gsiftpDefaultStreamsPerClient=<int>`

- Remove (most likely) corrupted/incomplete files after irregular transfer exit

  `gsiftpDeleteOnConnectionClosed=<bool>`

  – This variable must stay at true! Otherwise retries of a failed job will always result in an (file-already-exists-)error

gsiftp a.k.a. gridftp param.

- The range of port numbers to be used for transfers

  `clientDataPortRange=`*`<int1>`*`:`*`<int2>`*

  (where `int1` < `int2`)

- For internal communication the doors may use a different interface

  ```
  gsiftpAdapterInternalInterface=\
    <internal IP>
  ```

- This variable is neglected

  `FtpTLogDir`

passive gridftp a.k.a. gridftp2

- With activating passive gridftp transfers the gridftp doors may deligate the data transfer to the involved dCache pools
  - this depends on the used client commands, i.e. GET or PUT for FTP via FTS (“gridftp2 commands”)
  - in FTS 2.2.3 these are default
- Otherwise data will be tunneled through a proxy at the gridftp door

passive gridftp a.k.a. gridftp2

- `gsiftpAllowPassivePool=`*`<bool>`*
- The default is:
  - '`false`' for FTP doors
  - '`true`' for pools
- If set to `true` at the door, then the setting at the individual pool will be used

srm parameters

- srm falls back to gridftp for the datatransfer
- Hence settings for srm may override parameters for gridftp
  - e.g. `parallelStreams`
- Mostly srm parameters are an extension to gridftp transfers
- Often parameters are defined as default values; i.e. clients may specify own values

srm parameters

- Specify default version for srm usage
  `srmVersion=(version1|version2)`
- Value will be prepended to all SURL paths
  `pnfsSrmPath=(absolute path|/)`
- Redefine maximum number of parallel data streams per transfer
  `parallelStreams=<int>`
- Caching of proxy information needed for communication with gPlazma
  `srmAuthzCacheLifetime=<int>`

srm parameters

- Every srm transfer creates a TURL stored in the srm
- These have a limited life time (milliseconds) specifyed by

```
srmGetLifeTime=<int>
srmBringOnlineLifeTime=<int>
srmPutLifeTime=<int>
srmCopyLifeTime=<int>
```

- When life time is exceeded TURLs will be garbage collected; running transfers are not influenced

srm parameters

- Activate regular “vacuuming” of the srm database and set the interval

  `srmVacuum=<int>`

  `srmVacuumPeriod=<int>`

- Specifies the number of bytes for buffering of third party transfers (non-srm clients)

  `srmBufferSize=<int>`

  `srmTcpBufferSize=<int>`

srm parameters

- Another neglected variable

  `srmProxiesDirectory=/tmp`
- Maybe activates special developers debug output from srm; independently from log4j!

  `srmDebug=<bool>`
- Unfortunately we could not find out in time for what this timeout is needed

  srmTimeout=*<int>*

srm parameters

- Probably equivalent to a variable like `remoteGsiftpMoverTimeout`

  `srmMoverTimeout=<int>`

- Similar timeouts are also defined for gridftp

  `srmPoolManagerTimeout=<int>`

  `srmPoolTimeout=<int>`

  `srmPnfsTimeout=<int>`

srm parameters

- ‘`remote-`’ signals transfers between two (dCache) endpoints (doors)

```
remoteCopyMaxTransfers=<int>
remoteHttpMaxTransfers=<int>
remoteGsiftpMaxTransfers=\
  (${srmCopyReqThreadPoolSize}|<int>)
```

- If not set than will be shared with other queues of the same protocol
- For http this is probably not even supported by current client tools

srm parameters

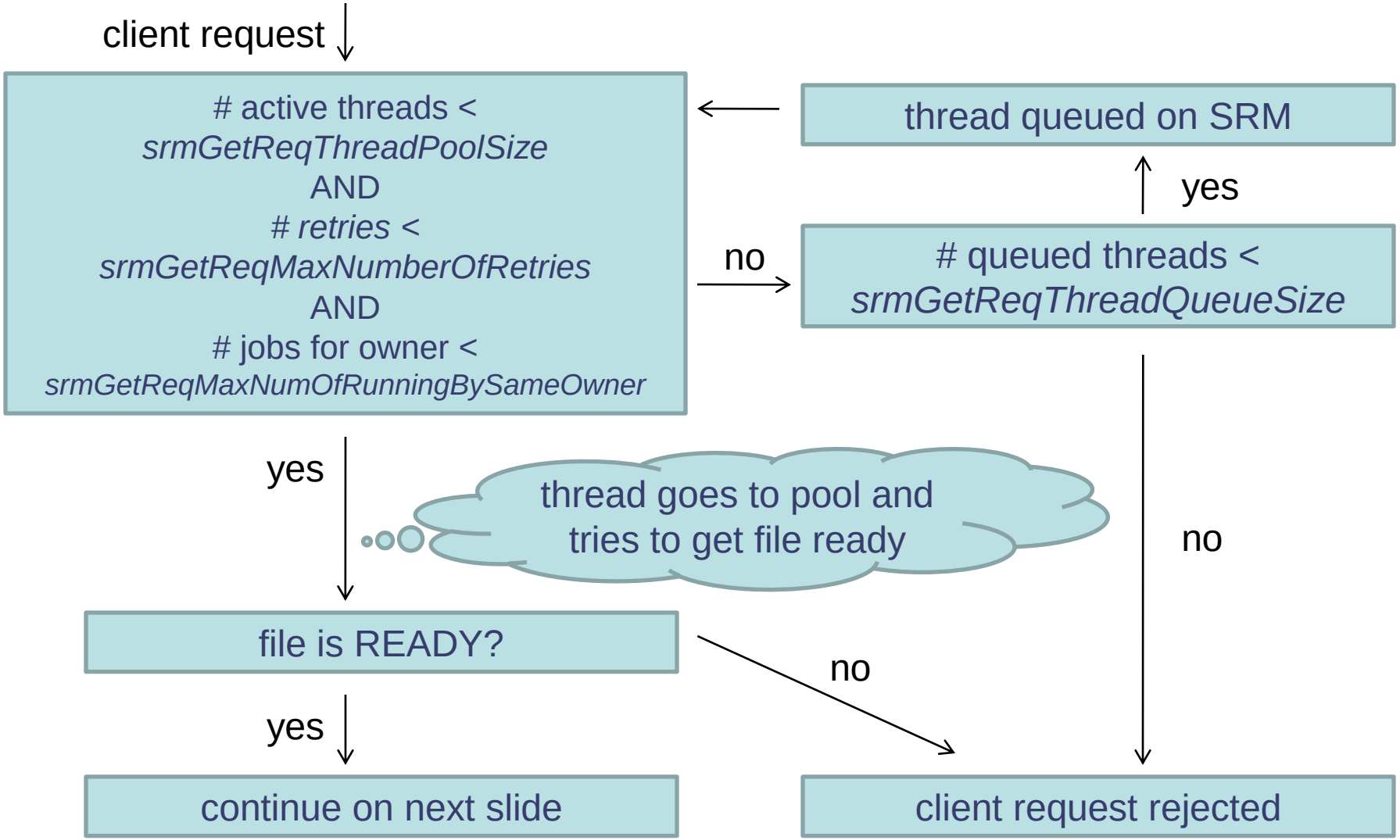
- Four identical sets of variables specific for each kind of srm transfer

```
srm<k1>ReqThreadQueueSize=<int>
srm<k1>ReqThreadPoolSize=<int>
srm<k2>ReqMaxWaitingRequests=<int>
srm<k2>ReqReadyQueueSize=<int>
srm<k1>ReqMaxReadyRequests=<int>
srm<k1>ReqMaxNumberOfRetries=<int>
srm<k1>ReqRetryTimeout=<int>
srm<k1>ReqMaxNumOfRunningBySameOwner=<int>
```

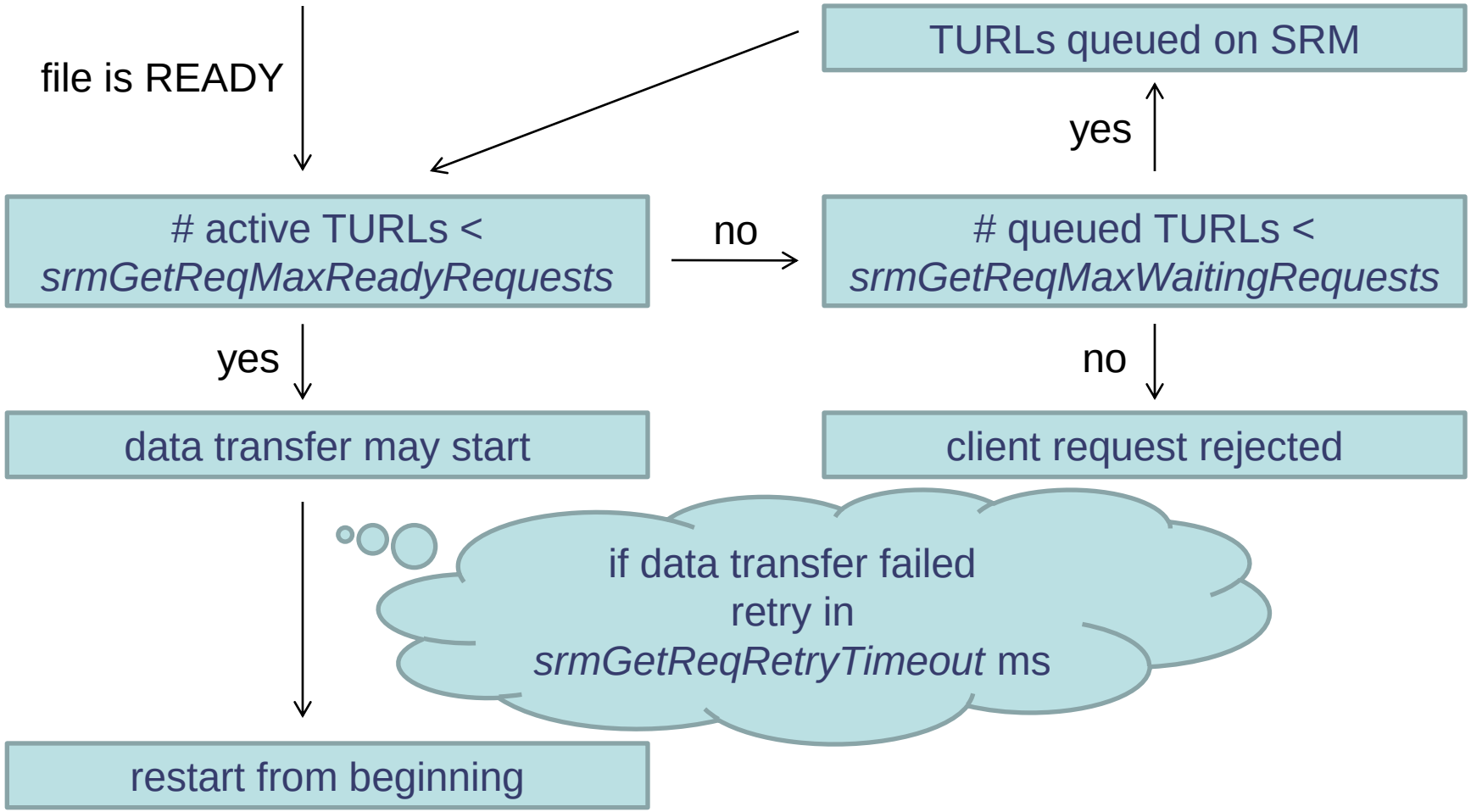
srm parameters

- k2 ∈ {BringOnline, Get, Put}
- k1 ∈ {k2, Copy}
- Explanation exemplified by the variables of get-transfers
- A statement from Timur Perelmutov (Fermilab) follows after the images

srm parameters



srm parameters



srm parameters

Timur Perelmutov (Fermilab):

"srmPrepareToGet and *srmBringOnline* requests are executed by the threads in the pool, *srmGetReqThreadPoolSize* specfies maximum number of such threads. When all the threads are busy, the rest of the requests are put on the queue. The maximum number of the elements in the queue is specied by *srmGetReqThreadQueueSize*. Once the files are prepared for reading, permissions are verified, files are staged, the files status is changed to Ready and a TURL is given to the user."

srm parameters

Timur Perelmutov (Fermilab):

"In order to limit the load of the system, that means to avoid clogging the system with to many parallel (SRM) transfers, maximum number of such requests is limited to *srmGetReqMaxReadyRequests*. The rest of the requests that are almost ready, except that all the transfer slots are occupied, are put on the ready queue, the maximal length of the queue is controlled by *srmGetReqMaxWaitingRequests*. If the execution of the request fails with non fatal error, the request is retried after the retry timout; the timeout time in milliseconds is controlled by *srmGetReqRetryTimeout*. "

srm parameters

Timur Perelmutov (Fermilab):

"If the request execution is retried *srmGetReqMaxNumberOfRetries* times but execution still fails the request is aborted and the error is propagated to the client. In order to implement fairness, we have the parameter *srmGetReqMaxNumOfRunningBySameOwner*. This way one user is limited to have a maximal number of running jobs at a time. If there are still jobs from another user in the queue, these will be started first."

xrootd parameters

- All parameters regarding usage of xrootd transfer protocol are well-documented in dCacheSetup

Last words

- Sources:
  - dCache, the Book
  - configuration files and source code
  - statements from experts/developers
- Credits go to…
  - dCache devs from DESY as well as FNAL
  - the German dCache support group