Request Management System

Ph.Charpentier
CERN, LHCb

5th Dirac User Workshop
May 27-29 2015

Introduction

- Aim of the RMS
  - Asynchronous execution of any operation
  - Mostly DMS-related operations
  - … but could be any operation
    - E.g. ForwardDISET operations for calling any DIRAC service
- Current usage of RMS
  - All centralised DMS operations (including failover)
  - … also Monitoring and Accounting failover
- Completely re-engineered since DIRAC v6r10 (K.Ciba)
- Since then maintained and developed by Chris H and PhC
- Not much new since last workshop, just a year ago…

Requests structure

- Requests
  - They are a set of Operations
  - Status fully driven by a FSM (from status of operations)
    - Status set artificially `Assigned` while being owned by an Agent
- Operations
  - They have a Type, a Status, possibly additional parameters
    - Status driven by an FSM (from Files status) or set otherwise
  - They may act on Files
- Files
  - In case an Operation acts on a (list of) file(s)
  - Each file has a Status

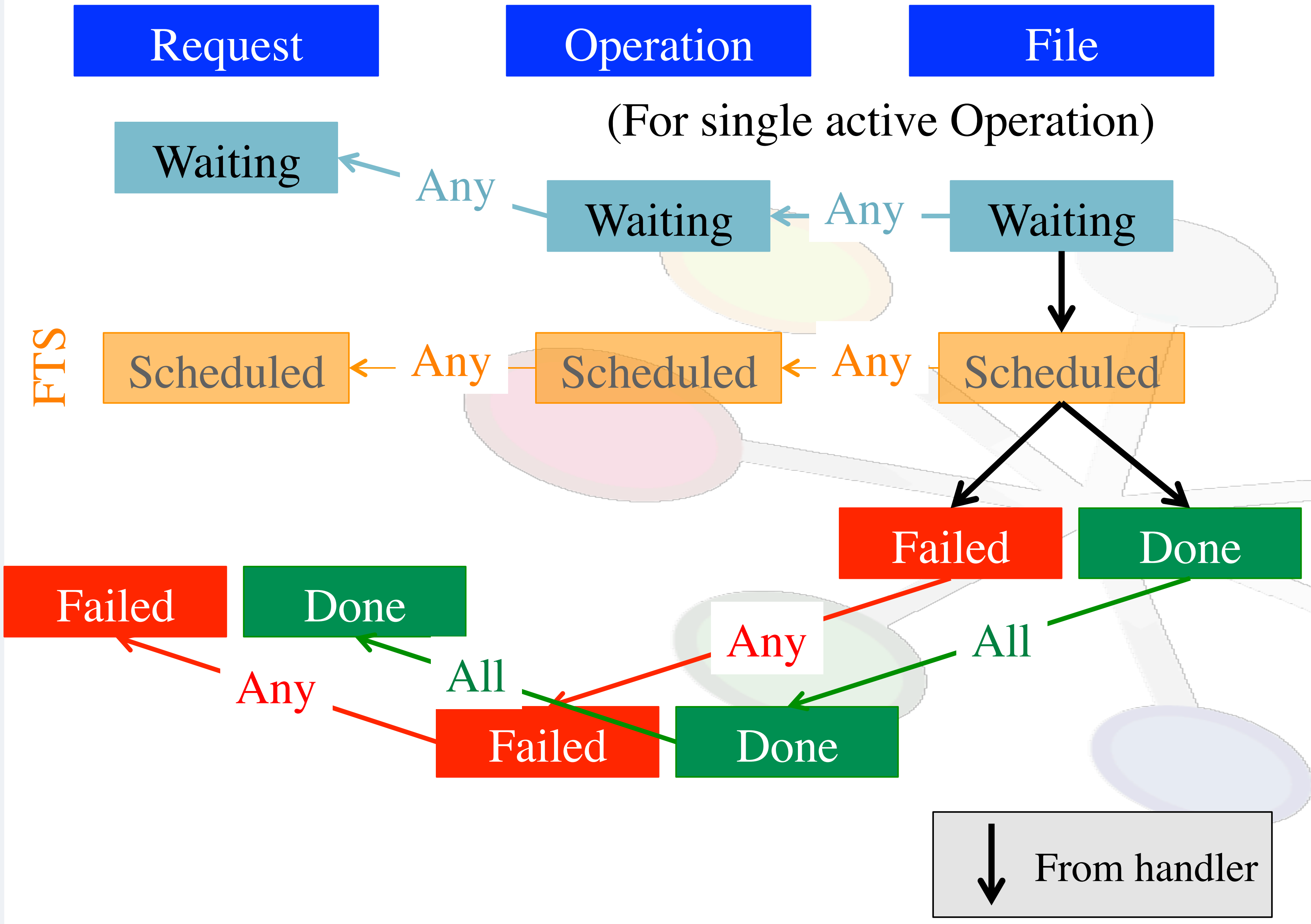
```
Request name='00036569_00014650_job_77936911' ID=728560 Status='Failed' Job=77936911
Created 2014-05-20 09:58:13, Updated 2014-05-20 12:06:54
Owner: '/DC=es/DC=irisgrid/O=ecm-ub/CN=Ricardo-Graciani-Diaz', Group: lhcb_data
  [0] Operation Type='ReplicateAndRegister' ID=1409859 Order=1 Status='Failed'
      SourceSE: CNAF-FAILOVER - TargetSE: GRIDKA-DST - Created 2014-05-20 09:57:56, Updated 2014-05-20 12:06:54
    [01] ID=1543098 LFN='/lhcb/LHCb/Collision12/SWIMSTRIPPINGD02KSKK.MDST/
00036569/0001/00036569_00014650_4.swimstrippingd02kskk.mdst' Status='Failed' Error='No such file or directory'
  [1] Operation Type='RemoveReplica' ID=1409860 Order=2 Status='Queued'
      TargetSE: CNAF-FAILOVER - Created 2014-05-20 09:57:56, Updated 2014-05-20 12:06:54
    [01] ID=1543099 LFN='/lhcb/LHCb/Collision12/SWIMSTRIPPINGD02KSKK.MDST/
00036569/0001/00036569_00014650_4.swimstrippingd02kskk.mdst' Status='Waiting'
```

RMS

Request execution

- A single Agent type is in charge of executing requests:
  - **RequestExecutingAgent**
    - Uses process pools (therefore independent environments)
  - Note: requests don't have a type, therefore multiple agents can be run but not for specific operations
- Operations are executed serially
  - When an Operation is Done, the next one (if any) is executed (if Failed, execution stops)
    - If no next operation, the Request is Done
- Execution is delegated to Operation Handlers
  - Mapping between Operation Type and Handler can be defined in the CS
  - Easily extendable (new type, new handler)
- Operations are executed using the request's owner credentials
  - Exception: if the owner is part of the production team (so-called shifters)
    - Then a Data Manager credential is used

State machine



Current operation handlers

RMS

- ReplicateAndRegister
  - Uses FTS unless otherwise setup (using group owner)
  - In LHCb: lhcb_user doesn't use FTS but directly Replica/ DataManager
- RegisterFile / RegisterReplica
  - Only register in file catalog(s)
- RemoveFile / RemoveReplica
  - Self explanaory
- PhysicalRemoval / PutAndRegister / ReTransfer
  - Implemented, not used by LHCb (but for Online upload)
- ForwardDISET
  - Make any DISET call (arguments passed as a blob)
  - No "Files"

- Extensions for LHCb
  - LogUpload

New features (I)

- The RMS is now using SQLAlchemy as DB inerface
  - Much better control of DB content and access
  - Was a good exercise for moving other DBs
  - Not a simple one due to the many getters and setters used in the RMS
  - Available as of v6r13

- Delayed retries
  - When an operation fails, it may be desirable to not retry immediately
  - E.g. in case of access problems to an SE that may be caused by it being down (not Banned) or a network problem
  - Possible to delay next execution
    - Add a waiting time to the Waiting operations
      - Unless set by the user code
    - Set automatically for certain types of “errors” (e.g. when an SE is banned)

RMS

New features (II)

- Request optimization
  - When requests are created by jobs, operations are created sequentially
  - In case a series of operations are identical, they can be grouped into a single operation
    - Advantage: a failing operation will not prevent others to execute
    - Example: file removal after a merging job
      - Without optimization, any failure prevents other removals to take place
      - With optimization, all files will be removed but those that fail
  - Similarly for the frequent failover pair of operations:
    - ReplicateAndRegister (from Failover to destination)
    - RemoveReplica (from Failover)
    - Whenever possible (same destination, i.e. most cases), group replication into a single operation
    - If same failover SE, group removal (not always possible)

RMS

Future plans

- Optimization of optimization…
  - Currently done server-side when calling putRequest()
    - Advantage: the request is directly inserted with optimisation
    - Caveat: optimisation is hard-coded, i.e. not flexible
  - Plan to use plugins for optimizing requests
  - Not yet a clear implementation
    - One possibility would be that requests are created as wrapped within a single "OptimizeRequest" operation
    - The OptimizeRequest handler examines the whole request, expands it with the included operations and sets the OptimizeRequest operation Done
    - This may have several interesting applications
- Requests linked to jobs
  - Currently request done -> job Done
  - What to do if the request fails?
    - Currently the job remains "Completed": very bad!
    - Requires manual intervention…
    - Better use plugins as the decision may be VO-dependent

RMS

FTS transfers

- Part of DMS, but closely related to RMS
- As of v6r13, FTS3 native REST interface is supported
- FTS2 command line mode still supported
  - As FTS2 servers are discontinued, we plan to decommission it in 2-3 releases
- FTS system uses RSS to know the status of servers
  - Just committed, not yet included in v6r13
  - Will allow to use a pool of servers without any human intervention
- User transfers
  - Currently not supported (jobs submitted with DM proxy)
  - With REST interface this is supported (credentials can be associated to FTS jobs)
- The changes above imply changes in the CS
  - See the DIRAC twiki for instructions (FTS3 separate configuration)

Conclusion

- The RMS was a very successful test bench for using SQLAlchemy for DB interaction
  - Avoid some security flaws (as spotted by EGI)
- Easily extendable for any kind of asynchronous operation
- Plan to use more plugin functionality
  - Less hard-coded actions (although there will be a baseline implementation)
  - Request optimization and verification
  - Job-related requests final status action
- FTS replication system moved to FTS3 REST interface
  - FTS2 frozen but supported until not used any longer
  - Any user credential can be used for transfers
    - Allows user file transfers
    - Registration using the same credentials
- Scalability can be achieved adding more processes in the process pool