Overview of TReqS

Bernard Chambon, CC-IN2P3

FJPPL meeting, LYON
14-15 February 2017

Outline

- Reminder about TReqS : motivation, positioning, history
- Overview
  - Features
  - Architecture
  - Current status
  - Development process
- Next plans

Reminder about TReqS = Tape Request Scheduler

- Motivation
  It's a software companion to HPSS, that re-organize HPSS staging requests
  - Decrease number of tape movements, by grouping files per name of tape
  - Increase staging throughput, by re-ordering files to be staged from same tape, according to FPOT [1]
  - Control number of allocated drives for staging
- Positioning
  - Between storage middleware and HPSS (current clients of TReqS are dCache, XRootD)
  - For HPSS staging only (tape $\rightarrow$ disk)
- History
  - Running old implementation developed 7 years ago, but not fully reliable nor maintainable
  - New implementation, started from scratch at fall 2015 (sometimes called TReqS-2)
- Numbers
  - HPSS : 45 PB TAPE, 600 TB DISK. 2000 tape mounts per day, up to 5 times in peak
  - TReqS : 10 K staging requests per day, up to 5 times in peak

1. FPOT: logical File Position On Tape + file offset (if exists)

Main features

- Business point of view
  - Aggregate requests over time per tape, sorting files according to FPOT : → queue
  - Limit number of simultaneous running queues, per tape model
    (ex 10 drives allocated for T10K-D)
  - Provide role management (user's role = ADMIN, USER or NONE)
  - Provide control (on/off) on tape, on tape-model, on HPSS access, on queues processing, on submission of client requests
  - Provide cancelation of client requests (cancel = removing file from queue)
  - Provide persistence for requests (useful for server stop & start)
  - Provide archiving for ended requests (built-in CSV archiver)
- Implementation point of view
  - REST API, JSON format, HTTPS support
  - JSW : Java Service Wrapper, to run application as a UNIX service (stop | start | status)
  - Out-of-the-box monitoring web pages

Detailed feature about queue

- Queue = name of the structure grouping files per name of tape
- Mechanism
  - Created (if not exist) as soon as a request is issued, and the file has retrieved metadata from HPSS
  - As many queues as involved tape names (queue name = tape name).
    No limit in number of queues nor in the size of a queue
  - Start running of queues is triggered at fixed interval or by an admin command
    o Queues selection order is based on the queues creation dates
    o Numbers of simultaneous running queues controlled by the limit defined per tape model.
    o A running queue can accept new files for staging them in the same 'run'
  - A queue is made of sets (see screenshots) sorting files according to FPOT as much as possible
  - Queue are not persisted, but requests are $\Rightarrow$ queues are re-constructed at server restart
- Screenshots : Overall view on all queues, detailed view on one queue KT537700

Detailed feature about control on HPSS access

- Mechanism
  - Control on HPSS access = just a off/on switch, no action on HPSS
  - When HPSS is switched off
    - ★ Incoming requests are accepted, but WITHOUT query metadata to HPSS, requests are in SUSPENDED status
    - ★ After end of staging of the current file, NO new file staging occurs the queue is left intact
  - When HPSS is switched on
    - ★ Staging of files restart from previous position in queue (previous=when HPSS switched off)
    - ★ Processing of suspended requests, to query metadata to HPSS Requests will change to SUBMITTED status, files are dispatched to appropriate queues (It is a mono-threaded (but fast [1]) operation)

1. From bench : sequential query of metadata for 5k, 10k, 20k files took 35s, 70s, 140s

Architecture : model client-server

- Server
  - Written in Java (18,000 lines of code) and C++ (500 lines of code)
  - Using JMS for internal exchanges, H2 DB for persistence, HPSS API via JNI
  - Providing a REST API with JSON over HTTPS
  - See schema for more details
  - Project available here
- Client
  - Written in Python (2,000 lines of code), using REST API
  - Authentication is based on login/password
  - Project available here

Current status

- Production configuration
  - Clients : dCache (80% of all TReqS' requests) for ATLAS, CMS, LHCB, other from XRootD
  - Considering ATLAS + CMS (90% of dCache' requests)
    o For dCache ATLAS, 9 hosts with 90 pools /host => potentially 810 'treqs copy' requests [1]
    o For dCache CMS, 72 hosts with 10 pools /host => potentially 720 'treqs copy' requests
  - Estimation of a potential of ∼1500 simultaneous connections for queries
- Pre-production deployment schedule
  - First deployment near end of November 2016.
    Facing some classical 'pre-production' use cases (missing features, bugs, etc.)
  - New deployment on 20 January 2017
    Using clients from dCache pools for ATLAS and CMS
    (Other clients still addressing old implementation)
- Feedback from first pre-production usage
  - First major staging campaign from 02/08-12:00 to 02/10-16:00
    → **Spread over ∼52 hours : 75,000 files processed, 145 TB staged**
    → **Stability and fluidity were ok**
  - The product is nearly ready for a full production usage at CC-IN2P3
    Estimation of 5∼10 days of dev. for last changes

1. 'treqs copy' : client command issuing a request then querying file's status waiting for the end of file staging

Development process

- Project mgmt
  - Maven as project management,
  - Released versions over time : alpha (2016/Q1), beta (2015/Q2), 1.0 (2016/09), **1.1 (2017/01)**
- Test methods :
  - Unit tests
    ∼200 tests, running without real HPSS nor JMS.
    Using Jenkins for continuous integration
  - Integration tests
    ∼20 tests, requiring HPSS access and JMS instantiation (→ not running in Jenkins)
  - Load tests [1]
    Outside distribution, based on client classes in Python
- Project access
  - Code under LGPL-V3 licence, access to granted user [2] on gitlab.in2p3.fr.
  - Build procedure available in ADMIN-GUIDE (see ./treqs2-delivery/resources/ directory)
  - Docs : README, ADMIN-GUIDE, CHANGELOG
  - Recommended version is **1.1 (2017/01/24)**

1. Using simulatedHPSS : test set of files used for querying metadata and file staging (sleep(...))
2. Tomoaki Nakamura has a granted access

Next plans

- Make TReqS fully available for production
  Provide more metrics, improve logs
  For daily operation, a monitoring tool is currently being developed by an admin (outside TReqS distribution)
- From functional point of view : Enhance staging criteria
  - Better understanding of production profile of CC-IN2P3, improve queues management
  - Brainstorming about others requirements like
    o Managing priority of requests,
    o Controlling (and sharing) usage of resources by (between) user(s)
- From development point of view
  - Improve code quality (e.g. taking into account feedback from Sonar)
  - Increase gitlab.in2p3.fr usage (e.g. for CI, for deployment)

Thank you for your attention

Thanks to Lionel Schwarz (other TReqS software developer)