Tuning SRM

Paul Millar

on behalf of the rest of the dCache team.

dCache workshop 2015
What is improved with SRM in 2.10

- Never abort a requests that has been worked on.
- Control over the induced load.
A request's journey: the simple* requests

* Requests include rm, mv, rmdir, mkdir, ...

TCP queue
Jetty queued
In progress
Done

SUCCESS
A request's journey: sched. non-transfers

Jetty

TCP queue  Jetty queued  Queued  In progress  Done

Request

SUCCESS

Bring-online, Copy, Ls, Reserve-space
A request's journey: asynchronous
A request's journey: retries
A request's journey: transfers
Throttling number of transfers

Max concurrent transfers
srm.request.*.max-transfers
Protecting the rest of dCache
Don't run out of memory!

Jetty

TCP queue
Jetty queued
Queued
In progress
RQueued
Ready
Done

Total requests
srm.request.*.max-requests

Load on dCache
srm.request.*.max-inprogress

Max concurrent transfers
srm.request.*.max-transfers

Request

QUEUED
INPROGRESS
SUCCESS

Retry
Info of non-transfer requests

--- scheduler-1s (Scheduler for LS operations) ---

Queued ........................... 0  [TQueued]
Waiting for CPU ............ 0  [PriorityTQueued]
Running (max 50) ............ 0  [Running]
Running without thread ... 0  [RunningWithoutThread]
Waiting for callback ...... 0  [AsyncWait]
In progress (max 50) ..... SUM >> 0
Queued for retry ............... 0  [RetryWait]

Total requests (max 50000) ....... 0

In progress per user soft limit : 100 requests
Maximum number of retries : 10
Retry timeout : 60000 ms
Retry limit : 10 retries
## Info of non-transfer requests

--- scheduler-put (Scheduler for PUT operations) ---

<table>
<thead>
<tr>
<th>State</th>
<th>Count</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Queued</td>
<td>0</td>
<td>[TQueued]</td>
</tr>
<tr>
<td>Waiting for CPU</td>
<td>0</td>
<td>[PriorityTQueued]</td>
</tr>
<tr>
<td>Running (max 12)</td>
<td>0</td>
<td>[Running]</td>
</tr>
<tr>
<td>Running without thread</td>
<td>0</td>
<td>[RunningWithoutThread]</td>
</tr>
<tr>
<td>Waiting for callback</td>
<td>0</td>
<td>[AsyncWait]</td>
</tr>
<tr>
<td><strong>In progress (max 50)</strong></td>
<td>SUM &gt; 0</td>
<td></td>
</tr>
<tr>
<td>Queued for retry</td>
<td>0</td>
<td>[RetryWait]</td>
</tr>
<tr>
<td>Queued for transfer</td>
<td>0</td>
<td>[RQueued]</td>
</tr>
<tr>
<td>Waiting for transfer (max 50000)</td>
<td>0</td>
<td>[Ready]</td>
</tr>
</tbody>
</table>

---

Total requests (max 500000) ........ 0

---

In progress per user soft limit : 100 requests
Maximum number of retries : 10
Retry timeout : 60000 ms
Retry limit : 10 retries

---

### Diagram

```
<table>
<thead>
<tr>
<th></th>
<th>Total requests</th>
<th>Load on dCache</th>
<th>Max concurrent transfers</th>
</tr>
</thead>
<tbody>
<tr>
<td>srm.request.*.max-requests</td>
<td>srm.request.*.max-inprogress</td>
<td>srm.request.*.max-transfers</td>
<td></td>
</tr>
</tbody>
</table>
```

---

Jetty

TCP queue

SRM

Queued

In progress

RQueued

Ready

Done

Request

Total requests

Load on dCache

Max concurrent transfers
Tuning points not covered...

- Tuning number of threads:
  - acceptor-thread(s), jetty-threads, scheduler-threads
- Closing idle connections,
- Tuning synchronous–asynchronous delay,
- Tuning DB behaviour,
  - connectors, persistence of requests
- Retry behaviour:
  - delay before retrying, number of retries to attempt
- Same-user request balancing,
- Number of in-flight rm requests

See Dmitry's previous talk on these points
one more thing...
Multiple SRM instances

Yes, you can do it...

even running multiple SRM instances on the same host*

... but for isolating multiple “customers”, not load-balancing.

* some limitations apply
Requirements

Each **SRM instance**...

- must use a different SRM databases (can be in same PostgreSQL instance)
- must be bound to different IP addresses (if on the same host)
- must be in different domains

**Clients** must...

- connect to same instance for duration of an asynchronous request.
- call srmPutDone / srmReleaseFiles on same SRM instance as srmPrepareToPut / srmPrepareToGet.

Probably easiest to have VO-specific SRM endpoint and clients that know to connect to that endpoint.
Thanks for listening!
What is SRM?

• Standard protocol for managing storage

• Features not available in other protocols:
  AL/RP, spaces, protocol negotiation, 2-stage commit for uploads, staging with pins, 3rd party copy, ...

• **Bulk** operations:

  BringOnline, PrepareToGet, Ls, PrepareToPut, Copy, CheckPermission, GetPermission, Rm, ReleaseFiles, PutDone, AbortFiles, ExtendFileLifeTime, ChangeSpaceForFiles, ExtendFileLifeTimeInSpace, PurgeFromSpace.

• **Asynchronous** operations:

  BringOnline, PrepareToGet, Ls, PrepareToPut, Copy, ReserveSpace, ChangeSpaceForFiles, UpdateSpace.
SRM: asynchronous operations

• What: tell client to come back later
• Why: some requests require SRM to communicate with other dCache components. While clients wait, memory and a thread are “wasted”
• How does it work: SRM starts a timer; if this goes off before reply is complete, tell client to come back.
• How to tune?
  srm.request.switch-to-async-mode-delay & .unit
• How to know what is the correct value?
SRM: tuning threads

- What: adjust the thread behaviour to match server
- Why: idle threads make server more responsive, maximum threads prevent running out of memory.
- How to tune?
  - srm.limits.jetty.threads.max
  - srm.limits.jetty.threads.min
  - srm.limits.jetty.threads.idle-time.max & .unit
  - srm.limits.jetty.threads.queued.max
- How to know what is the correct value?
  - 512 kB per thread (1,000 threads → 500 MiB)
  - takes time to create a thread (~ 0.25 ms), much better to avoid this
  - Tune threads.max and thread.queued.max so you don’t run out of memory
  - Tune threads.idle-time.max based on observed client behaviour
  - Tune threads.min on
SRM: client-view of requests

• General progression:

SRM_REQUEST_ {QUEUED → INPROGRESS → SUCCESS}

• What they mean:

QUEUED: no work done yet,
INPROGRESS: work started,
SUCCESS: finished.
SRM: internal states (simple case)

- Simple flow:
  
  PENDING $\rightarrow$ TQUEUED $\rightarrow$ RUNNING $\rightarrow$ DONE

- What they mean:

  PENDING – just received
  TQUEUED – not working on request
  RUNNING – dCache (SRM or elsewhere) working
  DONE – successful outcome.
SRM: internal states (transfer requests)

- Simple flow:
  PENDING → TQUEUED → RUNNING → RQUEUED → READY → DONE

- What they mean:
  PENDING, TQUEUED, RUNNING as before
  RQUEUED the TURL is ready but not handed to the client
  READY the TURL is in client's hands
  DONE the TURL is no longer valid, transfer was successful.
SRM: when too much activity

- When too much client activity, requests are queued
  - Need to remember client activity – writing to database for restart
What is SRM?

- Parameters for throttling client activity
- Parameters for recording client activity: Knowing what happened, surviving restart
- Parameters for protecting against OOM
- Parameters for generating TURLs
- Parameters controlling interaction with rest of dCache.