Queueing in dCache

Paul Millar
Berlin, 2013.05.28
Mythical self-organising users?

Credit: Florenz Kley@flickr
Queues: dealing with uneven loads

Credit: toronto_pcu@flickr
Always fast-enough too expensive
Threads and ThreadPools

- A thread is the smallest unit of independent CPU activity supported by Java (and most OSes)
- Creating a thread is (relatively) expensive
  - A thread-pool allows a thread to be used for multiple tasks
Queue + ThreadPool

Credit: mimi anderson@flickr
Queues and dCache

Credit: Sarah Macmillan@flickr
Queue overflowing are a symptom
Queues and dCache

- Impossible to get a comprehensive talk on this subject
  - It's far, far too big a topic!
- Instead, take a **worked example**:
  - Uploading a file using SRM and FTP
  - Client chooses to write file into a specific space-reservation
  - File is custodial/nearline, so flushed to tape
- With the following limitations:
  - Everything runs smoothly (no errors)
  - only presenting some of the interactions between components
    Skip over some details, when they are unenlightening.
  - No advise on tuning
- Many details are specific for this worked example.
Helicopter view

- Client connects to...
  - SRM and issues a prepareToPut.
  - GridFTP door, preparing dCache for upload
  - Pool, delivering data
  - SRM and issues a putDone command.
- Independent from the putDone, the pool flushes file to tape
The communication: srmPrepareToPut

Client

- prepareToPut()

SRM

- login
- login OK

- Does file already exist?
- No
- request parent directory metadata
  - <metadata>
- Check user can write
- Select door
- Increase READY turl count
- <turl>

gPlazma

- mark space in use

PnfsManager

Space Manager
Communication: FTP (part 1)

- USER

200 User logged in

PUT

create file

<SelectWritePool>

SelectWritePool

select LinkGroup

Store <PnfsId>

PoolAcceptFile

lookup AL & RP

PoolAcceptFile

Update load-model

PoolAcceptFile

update status

PoolManager

Pool

client

FTP door

gPlazma

PnfsManager

Space Manager
Communication: FTP (part 2)

- 127 PORT
- TransferStarted
- Last byte & close connection
- <size>,<AL>,<RP>,<checksum>
- TransferFinished
- Update load-model
- Update status
- Log transfer
- 226 Transfer complete.
- QUIT
- 221 Goodbye

Diagram showing the FTP communication process with various components such as client, FTP door, gPlazma, PnfsManager, Space Manager, PoolManager, Pool, and Billing.
Communication: srmPutDone

prepareToPut() → SRM

login

login OK

file exists?

yes

decrease READ TURL count

SUCCESS
Communication: pool-flush

- **Pool**: Time to flush file to tape
- **PnfsManager**: File exists?
  - Yes: Run tape integration script
  - FileFlushed: Update storage info
- **Space Manager**: FileFlushed
  - Free space
Queues: messages
Messages: generating quick reply
Messages: slower replies
Messages: replying (blocking)
Messages: registered call-back
Messages: pure asynchronous
Messages: tunnel
SRM: srmPrepareToPut
srmPrepareToPut: Jetty server

1. Wait for request: Disconnect if client takes too long
2. Parse request
3. Run SRM code to build reply
4. Send reply
5. If client requested it or HTTP/1.0
   Disconnect
6. Loop
srmPrepareToPut: generating reply
SRM and databases

- Thread
  - DB Query
  - Jdbc Queue
    - srmMaxNumber OfJdbcTasksInQueue (1,000)
  - DB Connection Pool
    - DB Connection Pool
      - max (50)
      - min(0)
      - Never Expires
  - Thread
    - srmJdbcExecution ThreadNum (5)
PnfsManager

MessageQueue

Thread

pnfsNumberOfThreads(4)

PnfsQueue MaxSize (0)

Single Thread

pnfsNumberOfThreads(4)

pnfsNumber OfThread Groups (1)
PnfsManager

PnfsThread

DB Connection Pool

Thread

Send Result

Folding

max (90)

idle (4 hours)

min (30)
SpaceManager

MessageQueue

ThreadManager

ThreadManager

DB Connection Pool

Thread

Thread

Send Result

max (30)

idle (4 hours)

min(0)
FTP upload and pool flush
FTP door

TCP Backlog

messageQueue

Thread

Create

Thread

Create

Thread

Command Queue

FTP Cmd

Unbounded

Single Thread

Process Command

Create

React on notification
PoolManager

MessageQueue

SelectWrite Pool

AcceptFile

Thread

Create

Forward message

Reply with result
Pool: AcceptFile

- MessageQueue
  - max=50
  - idle=1 min
  - min=5

- Mover Queue
  - maxActive Movers

- Thread

- TCP Backlog

- TransferStarted
- PnfsManager: update
- Billing: Info
- Door: TransferFinished

- Open socket

- Notify: new precious file
Pool: flush

- Notify: new precious file
- PnfsManager: file exists?
  - Yes
    - run script
    - PnfsManager: FileFlushed
    - SpaceManager: FileFlushed
Billing

MessageQueue

Thread

Write to file

Write to db
SRM srmPutDone
srmPutDone: Jetty server

1. Wait for request: Disconnect if client takes too long
2. Parse request
3. Run SRM code to build reply
4. Send reply
5. If client requested it or HTTP/1.0 Disconnect
6. Loop
srmPutDone: generating reply

Thread

gPlazma: login

OK

PnfsManager: Does file exist?

Yes

Decrease READY count
Summary

- Queues are not evil
- Queues should be tuned for good performance, even under heavy load
  - Simply increasing queues isn’t always optimal
- Quite a number of queues in dCache:
  - not all are obvious, not all are tunable