Optimizing Ceph for High Throughput

Sébastien Ponce
sebastien.ponce@cern.ch
The setup

Clients

CASTOR

Ceph

• client and CASTOR machines are batch nodes
• all machines have 10 Gb/s connection
• ceph machines have 540 disks in total
• ceph cluster has 2 PB of effective space
The setup

![Diagram showing the setup between Clients, CASTOR, and Ceph]

Some details

- client and CASTOR machines are batch nodes
- all machines have 10 Gb/s connection
- ceph machines have 540 disks in total
- ceph cluster has 2 PB of effective space
CASTOR - Ceph Network layout
Default test conditions

Client setup
• pure writing, 300MB files
• 10 concurrent threads per node, 8 nodes

CASTOR setup
• using xrootd protocol, with 64MB buffers
• 16 slots per disk server

Ceph setup
• erasure coded pool (8+3)
• striping into 8MB rados objects
Lesson 1: the buffer size in transfers

Situation up to xrootd 4.2

- maximum size of xrootd buffers is 2MB
- relation between buffer size and transfer speed
  - single stream, single box
  - with recompiled version of xrootd

<table>
<thead>
<tr>
<th>Buffer size (MB)</th>
<th>2</th>
<th>32</th>
<th>64</th>
</tr>
</thead>
<tbody>
<tr>
<td>Speed (MB/s)X</td>
<td>65</td>
<td>300</td>
<td>&gt;500</td>
</tr>
</tbody>
</table>
Lesson 1: the buffer size in transfers

Situation from xrootd 4.3 on

• big buffers have been added
• activate with
  
  xrd.buffers maxbsz <bsz>
• now max buffer size is 1 GB
Lesson 1 : the buffer size in transfers

Situation from xrootd 4.3 on

• big buffers have been added
• activate with
  \texttt{xrd.buffers maxbsz <bsz>}
• now max buffer size is 1 GB
• but async reading from ceph is broken in 4.3...
Lesson 2: the level of parallelization

Ceph has latency

- so async transfers are fundamental
- but they need to be sufficiently numerous
Lesson 2: the level of parallelization

Ceph has latency

- so async transfers are fundamental
- but they need to be sufficiently numerous
Lesson 3: be asynchronous

We got weird patterns
Lesson 3: be asynchronous

Ceph seemed slow to ack async writes

- Ack coming up to 20s after end of write
- ack tend to be delayed until end of activity
- file writing synchronize and files all end together
- at that moment, we wait
Lesson 3: be asynchronous

libradosstriper code was not 100% async

- striper locks the striped object during write
- on write completion, it calls unlock
- no async unlock in rados library
- these sync calls were serializing the end of writes
- and limiting writes/s throughput
Lesson 3 : be asynchronous

libradosstriper code was not 100% async

- striper locks the striped object during write
- on write completion, it calls unlock
- no async unlock in rados library
- these sync calls were serializing the end of writes
- and limiting writes/s throughput

Fix and effect

- added async unlock in rados and striper
- Ack now come within reasonable time
- Overall throughput almost unchanged...
Lesson 4: be parallel when async

Steps of an async call to ceph

- **Client threads**
  - build completion object containing callback
  - call of an async method of the rados(striper) API
  - calling thread can go on with something else

- **Objecter thread**
  - receives a callback from Messenger
  - uses the completion object to issue client callback
Lesson 4: be parallel when async

Limitation

- you can have many client threads
- you get a single callback thread per Objecter
  - that is by IOCtx or Striper object
- this single thread serializes all client callbacks
Lesson 4: be parallel when async

Limitation

• you can have many client threads
• you get a single callback thread per Objecter
  • that is by IOCtx or Striper object
• this single thread serializes all client callbacks

Solution

• multithreaded clients need to use a pool of IOCtx or Striper objects
Lesson 4 : be parallel when async

Impact of using the IOCtx pool
Lesson 4: be parallel when async

Impact of using the IOCtx pool

Is it something to integrate into rados library?
Lesson 4 aftermath...

Found out to be due to full disks
And garbage collection being too slow
Lesson 4 aftermath...

Found out to be due to full disks
And garbage collection being too slow
Lesson 5: back to lesson 3 (be async)

What is the GC doing?

- Deleting files at 3GB/s, 10 files per second
- And 100 32MB rados objects per second
Lesson 5 : back to lesson 3 (be async)

What is the GC doing ?

- Deleting files at 3GB/s, 10 files per second
- And 100 32MB rados objects per second

Why was it slow ?

1. no async remove in the striper library
Lesson 5: back to lesson 3 (be async)

What is the GC doing?

- Deleting files at 3GB/s, 10 files per second
- And 100 32MB rados objects per second

Why was it slow?

1. no async remove in the striper library
2. no async stat in the radosstriper library
   - called from garbage collector
Lesson 5: back to lesson 3 (be async)

What is the GC doing?

- Deleting files at 3GB/s, 10 files per second
- And 100 32MB rados objects per second

Why was it slow?

1. no async remove in the striper library
2. no async stat in the radosstriper library
   - called from garbage collector
3. no async stat / getxattr in the rados library
   - called within striper stat
Lesson 6: back to lesson 4

- GC also needs a IoCtx pool...
Conclusions

Ceph can deliver high throughput

- after a bit of learning
- and a couple of fixes
- one major lesson: BE ASYNCHRONOUS

Next steps

- bigger setup (15 proxies, 10GB/s)
- concurrent tape streams
- production for the Alice experiment