Many core clients and KNL scalability

Lustre Developer Summit 2016

Grégoire Pichon
22-09-2016
Test System

▶ Hardware

- Bull Sequana X1210 blade (prototype)
  - 1 socket Intel Xeon Phi 000A @1.20GHz
  - 66 cores / 264 cpus
- 192GB DDR4 memory
- 16GB MCDRAM memory
- 1 EDR Infiniband interface
- socket mode: SNC-4 Sub-NUMA Clustering

▶ Software

- Kernel 3.10.0-327.28.3.el7.x86_64
- MOFED 3.3-OFED.3.3.1.0.0.1
- Lustre 2.8.57
Lustre Compute Partitions and Worker Threads

- Default configuration
  - 12 compute partitions, 22 cpus each (4 numa nodes: 72/64/64/64 cpus)
  - ~380 worker threads (ptlrpcd, pttlrpc_hr, kibInd_sd, ldlm_cb, ...)
  ⇒ does not fit the hardware architecture (thread quartets, pairs of core: tile)
  ⇒ too much working threads

- Used configuration
  options libcfs cpu_pattern="0[0-15] 1[18-33] 2[34-49] 3[50-65]"
  - 4 compute partitions, 16 cpus each, 1 cpu per core
  - ~107 worker threads
LNet Performance

Lnet_selftest

LNet bandwidth
KNL - lustre 2.8.57 - size=1M

LNet iops
KNL - lustre 2.8.57

► Lustre Network performance on KNL platform
  - large data transfer bandwidth is OK
  - but small operations rate is 3 times slower
Large Sequential IOs – File Per Process

**IOR**

IOR FPP

KNL - lustre 2.8.57

![Graph showing throughput (MiB/s) vs. tasks for IOR FPP on KNL with lustre 2.8.57. The graph includes a legend indicating write and read data points.](image)
Large Sequential IOs – File Per Process
Profiling - 240 tasks

<table>
<thead>
<tr>
<th>perf.report.ior.write.240tasks.lu2.8.57.txt</th>
</tr>
</thead>
<tbody>
<tr>
<td>24.53% IOR</td>
</tr>
<tr>
<td>1.56% IOR</td>
</tr>
<tr>
<td>0.99% IOR</td>
</tr>
<tr>
<td>0.92% IOR</td>
</tr>
</tbody>
</table>

- major contention on
  - page_zone(page)-&gt;lru_lock

- minor contention on
  - cl_object-&gt;coh_attr_guard
  - client_obd-&gt;cl_loi_list_lock
  - osc_object-&gt;oo_lock
  - obd_import-&gt;imp_lock

<table>
<thead>
<tr>
<th>perf.report.ior.read.240tasks.lu2.8.57.txt</th>
</tr>
</thead>
<tbody>
<tr>
<td>39.31% IOR</td>
</tr>
<tr>
<td>6.60% IOR</td>
</tr>
</tbody>
</table>

- osc_lru_alloc() & osc_lru_reclaim()
  logic appears to be expensive