Is Open Source good enough? A deep study of Swift and Ceph performance

Jiangang.duan@intel.com

11/2013

The OpenStack Summit
Join Us In Hong Kong November 5-8, 2013
Agenda

• Self introduction
• Ceph Block service performance
• Swift Object Storage Service performance
• Summary
Self introduction

• Jiangang Duan (段建钢)
• Open source Cloud Software team in Intel @ Shanghai
• OpenStack related performance

This is the team work:
• Chendi Xue, Chen Zhang, Jian Zhang, Thomas Barnes, Xinxin Shu, Xiaoxi Chen, Yuan Zhou, Ziyuan Wang, etc.
Agenda

• Self introduction
• Ceph Block service performance
• Swift Object Storage Service performance
• Summary
Ceph Test Environment

- Full 10Gb network
- 4 Xeon UP server for Ceph cluster:
  - 16GB memory each
  - 10x 1TB SATA HDD for data through LSI9205 HBA (JBOD), each is parted into 1 partition for OSD daemon
  - 3x SSD for journal directly connected with SATA controller, 20GB for each OSD
# Ceph Test-Bed Configuration: Software

## Ceph cluster

<table>
<thead>
<tr>
<th>OS</th>
<th>Ubuntu 12.10</th>
</tr>
</thead>
<tbody>
<tr>
<td>Kernel</td>
<td>3.6.6</td>
</tr>
<tr>
<td>Ceph</td>
<td>Cuttle Fish 0.61.2</td>
</tr>
<tr>
<td>FileSystem</td>
<td>XFS</td>
</tr>
</tbody>
</table>

## Client host

<table>
<thead>
<tr>
<th>OS</th>
<th>Ubuntu 12.10</th>
</tr>
</thead>
<tbody>
<tr>
<td>Kernel</td>
<td>3.5.0-22</td>
</tr>
<tr>
<td>OpenStack</td>
<td>Folsom (2012.2)</td>
</tr>
</tbody>
</table>

## Client VM

<table>
<thead>
<tr>
<th>OS</th>
<th>Ubuntu 12.10</th>
</tr>
</thead>
<tbody>
<tr>
<td>Kernel</td>
<td>3.5.0-22</td>
</tr>
</tbody>
</table>

## Network tuning

- Enable Jumbo Frame (1496 bytes -> 8000 bytes)

## XFS tuning

- osd mount options xfs = rw,noatime,inode64,logbsize=256k,delaylog
- osd mkfs options xfs = -f -i size=2048

## Ceph Tuning

- Default replication setting (2 replicas), 4096 pgs.
- Enlarge read ahead size to 2048 Kb for OSD disks.
- Max Sync Interval 5s - > 10s
- Large Queue and Large Bytes
- OSD OP Thread 2 -> 20
- Turn off FileStore Flusher
- Sync Flush False -> True (to avoid a OSD crash bug)
Methodology

• Storage Interface:
  • RBD (aka volume) is attached to one virtual machine via QEMU RBD driver

• Workload
  • FIO to simulate 4 IO pattern:
    • 64k Sequential Read/Write (SR and SW) with 60MB/sec max per VM
    • 4k Random Read/Write (RR and RW) with 100IOPS max per VM

• Run Rules
  • Volume is pre-allocated before testing with dd
  • Drop OSD page cache before each measurement
  • Duration 400 secs warm-up, 600 secs data collection
  • 60GB per Volume, increase Stress VM number to get full load line

• QoS requirement:
  • Less than 20ms latency for random read/write
  • Per VM throughput larger than 90% of the predefine threshold
    • 54MB/sec for SR/SW and 90IOPS for RR/RW
Random read performance

Max measured value at ~4600 IOPS
When scale to 30VM, per VM IOPS drop to 95

For more complete information about performance and benchmark results, visit Performance Test Disclosure
Random write performance

- Peak at ~3291 IOPS
- When scale to 40VM, per VM VW drop to 82 IOPS per VM

For more complete information about performance and benchmark results, visit Performance Test Disclosure
Sequential read performance

- Max measured value at ~2759MB/s
- When scale to 40VM, per VM BW drop to 57MB/sec

For more complete information about performance and benchmark results, visit Performance Test Disclosure.
Sequential write performance

Max at 1487MB/sec

When scale to 30VM, per VM BW drop to 45MB/sec per VM

For more complete information about performance and benchmark results, visit Performance Test Disclosure
Latency analysis

- Latency is strongly related to size of QD
- Random is good enough. Sequential still has improve room.

For more complete information about performance and benchmark results, visit Performance Test Disclosure
IO trace analysis for Sequential read

- Ceph converts the logical sequential to random disk access
- Read ahead can help, but not enough
• 4x 200GB SSD with 8 OSD (60GB data + 10GB journal each)
• Good enough full SSD performance.
• ~55K with 1ms latency; ~80K with 2ms, peak 85K with 3ms

For more complete information about performance and benchmark results, visit Performance Test Disclosure
Ceph Data Summary

- Random performance is pretty good.
- Sequential performance has room to do better.
- Challenges and opportunities:
  - Throttle both BW and IOPS to better support multiple tenant case
  - Mix SSD and HDD together to achieve a balanced capacity/performance.

<table>
<thead>
<tr>
<th>Workload Pattern</th>
<th>Max Measurable Throughput</th>
<th>Throughput Consider QoS</th>
<th>In Theory Disk Throughput (IOPS and MB/s)</th>
<th>In Theory Network Throughput (IOPS and MB/s)</th>
<th>Efficiency</th>
</tr>
</thead>
<tbody>
<tr>
<td>4KB random read CAP=100</td>
<td>4582</td>
<td>2858</td>
<td>3600</td>
<td>31250</td>
<td>79%</td>
</tr>
<tr>
<td>4KB random write CAP=100</td>
<td>3357</td>
<td>3000</td>
<td>4000</td>
<td>31250</td>
<td>75%</td>
</tr>
<tr>
<td>64KB seq read QD=64 CAP=60</td>
<td>2759</td>
<td>2263</td>
<td>6400</td>
<td>4000</td>
<td>57%</td>
</tr>
<tr>
<td>64KB seq write QD=64 CAP=60</td>
<td>1487</td>
<td>1197</td>
<td>3200</td>
<td>4000</td>
<td>37%</td>
</tr>
</tbody>
</table>

Assume 160MB/s for SR/SW and 90IOPS for RR and 200IOPS for RW and 40Gb Network BW limit

<table>
<thead>
<tr>
<th>Workload Pattern</th>
<th>Total Capacity (GB)</th>
<th>Total Throughput (IOPS)</th>
<th>60GB Volume Number Consider Capacity</th>
<th>60GB Volume Number Consider Performance</th>
</tr>
</thead>
<tbody>
<tr>
<td>40x 1TB with replica=2</td>
<td>20000</td>
<td>2800</td>
<td>333.3</td>
<td>30</td>
</tr>
<tr>
<td>3 x 200GB SSD with replica=2</td>
<td>150</td>
<td>85000</td>
<td>2.5</td>
<td>850</td>
</tr>
</tbody>
</table>

Consider Random IO only

For more complete information about performance and benchmark results, visit Performance Test Disclosure
Agenda

• Self introduction

• Ceph Block service performance

  • Swift Object Storage Service performance

• Summary
H/W Configuration

• Proxy Node x2
  – CPU = 2 * Intel Xeon E5 2.6GHz (8C/16T)
  – Memory = 64GB, 8x 8G DDR3 1333MHz
  – NIC = Intel 82599 Dual port 10GbE
  – OS = CentOS 6.4 (2.6.32-358.14.1)
  – BIOS = Turbo on, EIST on, CPU C-state ON

• Storage Node x10
  – CPU = 1* Intel Xeon E3 3.5GHz (4C/8T)
  – Memory =16 GB, 4x 4G DDR3 1333MHz
  – NIC = 1 * Intel 82576 Quad port 1GbE (4*1Gb)
  – Disk = 12* 1TB 7200rpm SATA, 1x 64GB SSD (3Gb/s) for account/container zone
  – OS = CentOS 6.4 (2.6.32-358.14.1)

• Network
  – 1x 10Gb to in/out of Proxy
  – 4x 1Gb for each storage node
Software Configuration

• Swift 1.9.0
• Swift Proxy Server
  – log-level = warning
  – workers = 64
  – account/container-recheck = 80640s
• Swift Account/Container Server
  – log-level = warning
  – workers = 16
• Swift Object Server
  – log-level = warning
  – workers = 32
• Swift replicator/auditor/updater
  – default

• Memcached Server
  – cache-size = 2048MB
  – max-concurrent-connections = 20480
## Test Methodology

COSBench is an Intel developed Benchmark to measure Cloud Object Storage Service performance. [https://github.com/intel-cloud/cosbench](https://github.com/intel-cloud/cosbench)

<table>
<thead>
<tr>
<th></th>
<th>Small-Scale Test</th>
<th>Large-Scale Test</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Test operations</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Operations</td>
<td>Read/Write</td>
<td>Read/Write</td>
</tr>
<tr>
<td><strong>Swift</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Object Size</td>
<td>128K/10M</td>
<td>128K/10M</td>
</tr>
<tr>
<td># of containers</td>
<td>100</td>
<td>10000</td>
</tr>
<tr>
<td># of objects/container</td>
<td>100</td>
<td>10000</td>
</tr>
<tr>
<td><strong>CObbench</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td># of workers</td>
<td>Start from 5, doubled until 2560</td>
<td>Start from 10, doubled until 2560</td>
</tr>
<tr>
<td>Ramp up (s)</td>
<td>300</td>
<td>300</td>
</tr>
<tr>
<td>Run time (s)</td>
<td>300</td>
<td>300</td>
</tr>
<tr>
<td>Ramp down (s)</td>
<td>100</td>
<td>100</td>
</tr>
<tr>
<td>Metrics</td>
<td>Operations per second</td>
<td>Operations per second</td>
</tr>
<tr>
<td>Compliant performance</td>
<td>avg response time &lt; 200ms + object size/2M s</td>
<td>avg response time &lt; 200ms + object size/2M s</td>
</tr>
<tr>
<td>B/W</td>
<td>MB/s</td>
<td>MB/s</td>
</tr>
</tbody>
</table>
## Swift Performance Overview

<table>
<thead>
<tr>
<th>#con x # obj</th>
<th>Object-Size</th>
<th>RW-Mode</th>
<th>Worker-Count</th>
<th>Avg-ResTime</th>
<th>Throughput</th>
<th>Bottleneck</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>--</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>ms</td>
<td>op/s</td>
<td>--</td>
</tr>
<tr>
<td>100x100</td>
<td>128KB</td>
<td>Read</td>
<td>640</td>
<td>51.77</td>
<td>12359.84</td>
<td>Proxy CPU = 100%, SN CPU = 72%</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Write</td>
<td>640</td>
<td>213.12</td>
<td>3001.65</td>
<td>Proxy CPU = 47%, SN CPU = 96%</td>
</tr>
<tr>
<td>10MB</td>
<td>Read</td>
<td>160</td>
<td>699.3</td>
<td>228.78</td>
<td></td>
<td>Proxy CPU = 61%, SN CPU = 55%, Proxy NIC 9.6Gb/s</td>
</tr>
<tr>
<td></td>
<td>Write</td>
<td>80</td>
<td>1020.8</td>
<td>78.37</td>
<td></td>
<td>Proxy CPU = 15%, SN CPU = 59%, Proxy NIC 9.6Gb/s</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>#con x # obj</th>
<th>Object-Size</th>
<th>RW-Mode</th>
<th>Worker-Count</th>
<th>Avg-ResTime</th>
<th>Throughput</th>
<th>Bottleneck</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>--</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>ms</td>
<td>op/s</td>
<td>--</td>
</tr>
<tr>
<td>10000*10000</td>
<td>128KB</td>
<td>Read</td>
<td>640</td>
<td>248.56</td>
<td>2574.78</td>
<td>Proxy CPU = 25% SN CPU = 52%, disk latency 22ms</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Write</td>
<td>80</td>
<td>62.75</td>
<td>1272.97</td>
<td>Proxy CPU = 21%, SN CPU = 57%, disk latency 18ms</td>
</tr>
<tr>
<td>10000*100</td>
<td>Read</td>
<td>160</td>
<td>699.17</td>
<td>228.83</td>
<td></td>
<td>Proxy CPU = 56%, SN CPU = 62%, Proxy NIC 9.6Gb/s</td>
</tr>
<tr>
<td></td>
<td>Write</td>
<td>320</td>
<td>4096.3</td>
<td>78.12</td>
<td></td>
<td>Proxy CPU = 16%, SN CPU = 81%, Proxy NIC 9.6Gb/s</td>
</tr>
</tbody>
</table>

- For large object, bottleneck is front end network
- 80% and 58% performance degradation in many-small-objects case

For more complete information about performance and benchmark results, visit [Performance Test Disclosure](#)
Problem: large-scale test 128K read 80+% drops

• Different IO pattern
  - large-scale: 34 KB per op
  - small-scale: 122KB per op

• Blktrace shows lots of IO generated by XFS file system
  - Meta data cannot be cached with 10k * 10k objects
  - Frequent disk access for that info

For more complete information about performance and benchmark results, visit Performance Test Disclosure
Option 1: Using larger memory to cache Inodes

- Set vfs_cache_pressure =1 and run "ls -R /srv/node" to preload the cache
- Close the gap less than 10% although this is expensive option.
  - 100M object with 3 replica need 300GB memory size
Option 2 : Using SSD to cache Inodes

• Use SSD to Cache inode and metadata w/ flashcache
• 50%-100% performance boost
• Still big room (70%) comparing to base scale
Alternative Solutions and Next step

- Take a small inode size in latest XFS?
- Using a different Filesystem (e.g., reiserfs, btrfs)?
- Taking NoSQL?
- Using customized indexing and object format?
  - Haystack (Facebook) and TFS (Taobao FS)
- More work for SSD as cache
  - Write back vs. write throughput
  - Where does the latency go?
  - Data Lost issue, mirror the SSD or combine them with container/OS disk together?
Summary

• Many emerging open source storage projects to satisfy BIG DATA requirement

• Ceph and swift are widely used in OpenStack environment

• Good result however still improvement areas

• Call action to work together
Disclaimers

INFORMATION IN THIS DOCUMENT IS PROVIDED IN CONNECTION WITH INTEL PRODUCTS. NO LICENSE, EXPRESS OR IMPLIED, BY ESTOPPEL OR OTHERWISE, TO ANY INTELLECTUAL PROPERTY RIGHTS IS GRANTED BY THIS DOCUMENT. EXCEPT AS PROVIDED IN INTEL'S TERMS AND CONDITIONS OF SALE FOR SUCH PRODUCTS, INTEL ASSUMES NO LIABILITY WHATSOEVER AND INTEL DISCLAIMS ANY EXPRESS OR IMPLIED WARRANTY, RELATING TO SALE AND/OR USE OF INTEL PRODUCTS INCLUDING LIABILITY OR WARRANTIES RELATING TO FITNESS FOR A PARTICULAR PURPOSE, MERCHANTABILITY, OR INFRINGEMENT OF ANY PATENT, COPYRIGHT OR OTHER INTELLECTUAL PROPERTY RIGHT.

A “Mission Critical Application” is any application in which failure of the Intel Product could result, directly or indirectly, in personal injury or death. SHOULD YOU PURCHASE OR USE INTEL’S PRODUCTS FOR ANY SUCH MISSION CRITICAL APPLICATION, YOU SHALL INDEMNIFY AND HOLD INTEL AND ITS SUBSIDIARIES, SUBCONTRACTORS AND AFFILIATES, AND THE DIRECTORS, OFFICERS, AND EMPLOYEES OF EACH, HARMLESS AGAINST ALL CLAIMS COSTS, DAMAGES, AND EXPENSES AND REASONABLE ATTORNEYS’ FEES ARISING OUT OF, DIRECTLY OR INDIRECTLY, ANY CLAIM OF PRODUCT LIABILITY, PERSONAL INJURY, OR DEATH ARISING IN ANY WAY OUT OF SUCH MISSION CRITICAL APPLICATION, WHETHER OR NOT INTEL OR ITS SUBCONTRACTOR WAS NEGLIGENT IN THE DESIGN, MANUFACTURE, OR WARNING OF THE INTEL PRODUCT OR ANY OF ITS PARTS.

Intel may make changes to specifications and product descriptions at any time, without notice. Designers must not rely on the absence or characteristics of any features or instructions marked “reserved” or “undefined”. Intel reserves these for future definition and shall have no responsibility whatsoever for conflicts or incompatibilities arising from future changes to them. The information here is subject to change without notice. Do not finalize a design with this information.

The products described in this document may contain design defects or errors known as errata which may cause the product to deviate from published specifications. Current characterized errata are available on request. Contact your local Intel sales office or your distributor to obtain the latest specifications and before placing your product order.

Copies of documents which have an order number and are referenced in this document, or other Intel literature, may be obtained by calling 1-800-548-4725, or go to: http://www.intel.com/design/literature.htm%20

This document contains information on products in the design phase of development.

Intel and the Intel logo are trademarks of Intel Corporation in the U.S. and/or other countries.

*Other names and brands may be claimed as the property of others.

Copyright © 2012 Intel Corporation. All rights reserved.