Red Hat Performance Briefs

Red Hat OpenStack Platform on Red Hat Ceph Storage

Cinder Volume Performance at Scale

Performance and Scale Engineering

Version 1.1 February 2017

RHEL 7.3
RHOSP 10
RHCS 2.0

Table of Contents

100 East Davie Street
Raleigh NC 27601 USA
Phone: +1 919 754 3700
Phone: 888 733 4281
Fax: +1 919 754 3701
PO Box 13588
Research Triangle Park NC 27709 USA

The GPG fingerprint of the security@redhat.com key is: CA 20 86 86 2B D6 9D FC 65 F6 EC C4 21 91 80 CD DB 42 A6 0E

Chapter 1. Executive Summary

This document describes large-scale I/O characterization testing performed by the Red Hat Performance Engineering group on:

- Red Hat Enterprise Linux (RHEL) 7.3
- Red Hat OpenStack Platform (RHOSP) 10
- Red Hat Ceph Storage (RHCS) 2.0

using Glance images, Nova instances, and Cinder volumes on RHCS. The authors are not aware of any other RHOSP-RHCS test performed in Red Hat at this scale. Up to 1.8 PB of storage, in an external RHCS cluster of 29 servers, and 20 RHOSP compute nodes were used to exercise as many as 512 RHOSP instances with FIO (Flexible I/O) benchmarks measuring application latency percentiles as a function of time while the following operational events were simulated:

- OSD failure and recovery
- OSD node failure and recovery
- RHCS Monitor failure and recovery

The purpose of this testing was not to achieve world-record performance by extensive tuning, but to document and understand the user experience for a normal configuration of this type with respect to performance, and specifically to understand issues customers might experience with such a configuration in production.

Some integration issues with RHOSP and RHCS were found and documented as bugzilla reports. These problems have either been fixed or are being fixed in the upcoming release. See Appendix: Issues for details.

Chapter 2. Test Environment

This section describes the hardware, software, RHCS, and RHOSP configurations used for the systems under test.

2.1. Hardware

Table 1 identifies the specifications of the hardware used in testing.

Table 1. Hardware Configuration

RHCS OSD Node	Supermicro SSG-6048R-E1CR36H with 256GB RAM, 2-socket (28) Intel Xeon E5-2660 v4 @2.00GHz (56 threads/server) Dual-port 40-GbE (SFP+) - Intel XL710 LSI 3108 disk controller (2) P3700 800GB NVMe (journals) (36) 2TB 7.2k SATA (OSDs) (2) 500GB 7.2k SATA (system)
RHCS Monitor	Supermicro SSG-6018R-MON2 with 64GB RAM, 2-socket (20) Intel Xeon E5-2630 v4 @2.20GHz (40 threads/server) (2) 80GB SSD SATA (system) Dual-port 10-GbE (SFP+)
RHOSP Controller **RHOSP Compute**	Dell PowerEdge R620 with 64GB RAM, 2-socket (12) Intel Xeon E5-2630 v4 @2.20GHz (24 threads/server) (4) 1TB SATA (1) Intel X520 Dual Port 10-GbE NIC
RHOSP Instance (VM)	1 CPU, 1GB Memory, 20GB system disk
Network	Juniper QFX5200 Switches, 100-GbE uplinks

2.2. Software

Table 2 lists the versions of relevant software packages used in testing.

Table 2. Software Configuration

RHCS OSD Nodes (29)	RHEL 7.3 RHCS 2.0 tuned active profile: *throughput-performance*
RHOSP Controllers (3)	RHEL 7.3 RHOSP 10 RHCS 2.0 tuned active profile: *throughput-performance*

RHOSP Computes (20)	RHEL 7.3 RHOSP 10 RHCS 2.0 tuned active profile: *virtual-host*
RHOSP Instances (512)	RHEL 7.3 fio-2.14 tuned active profile: *virtual-guest*

See Appendix: Tuned Profiles for details of the profiles referenced in the Software Configuration table.

2.3. Red Hat Ceph Storage

The RHCS cluster was deployed separately using ceph-ansible (v1.0.5-32) prior to the RHOSP configuration. See Appendix: References for the documentation used to enable the repository for the Red Hat Storage Console and Ansible installer.

All OSDs were added in parallel deploying 1.8PB of RHCS in approximately 60 minutes. The testing was performed using as many as 29 RHCS OSD nodes, 5 RHCS monitors and 20 RHOSP compute nodes. Each RHCS OSD node allocated 35-36 2TB local disks as OSDs (788 total) and two NVMe SSDs for journaling.

The RHCS configuration file used during testing is in Appendix: RHCS Configuration File. Each RHOSP instance had one 100GB cinder volume created, attached, pre-allocated, XFS formatted, and mounted as /dev/vdb for all I/O tests.

2.3.1. PG Settings

The number of placement groups (PGs) for the pool (pg_num) as well as the number of PGs to use when calculating data placement (pgp_num) were adjusted using the recommended https://access.redhat.com/labs/cephpgc/Ceph [PGs per Pool Calculator] specifying OpenStack as the Ceph use case with no erasure coding or cache tiering. Because this testing focused on Cinder volume performance, no cinder-backup pool was used and the percentage of data for that pool was added to that of Cinder volumes. Testing started with 29 OSD nodes and a total 1042 OSDs so the initial PG counts were calculated as seen in Table 3.

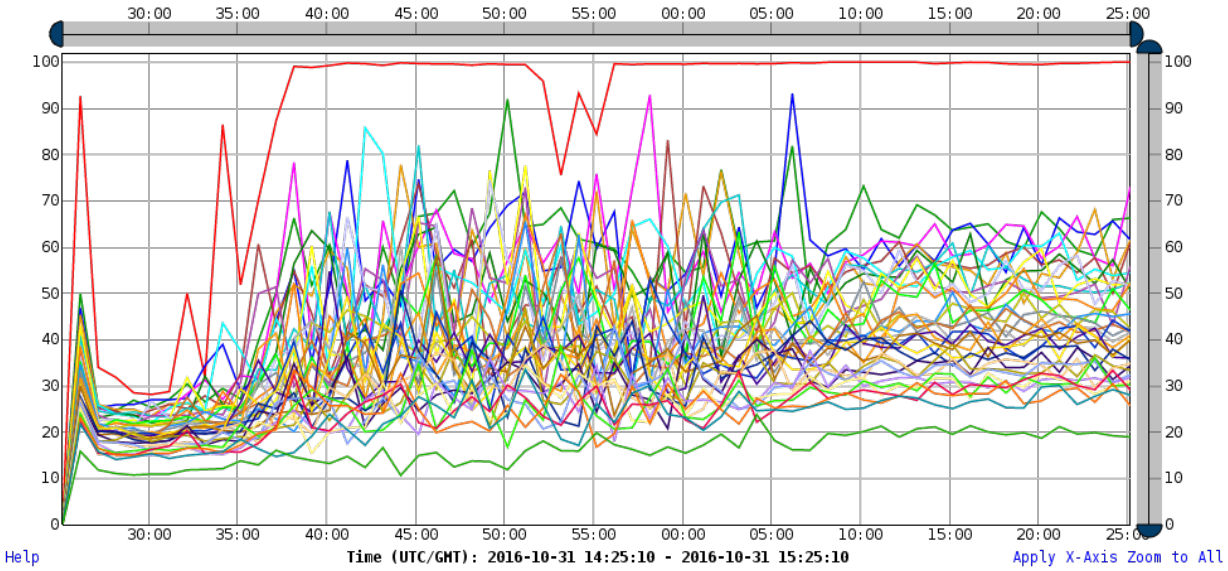
Table 3. RHCS PG Calculations

Pool Name	Pool Type	Replica Size	# OSDs	% Data	Target PGs/OSD	PG Count
volumes	Replicated	3	1042	78	100	**32768**
vms	Replicated	3	1042	15	100	**4096**
images	Replicated	3	1042	7	100	**2048**

A ratio of 100 PGs/OSD was originally used in PGcalc, resulting in a PG count of 32768. This resulted in quite a spread in the OSD disk utilization, as shown in the next figure, with the lowest utilization

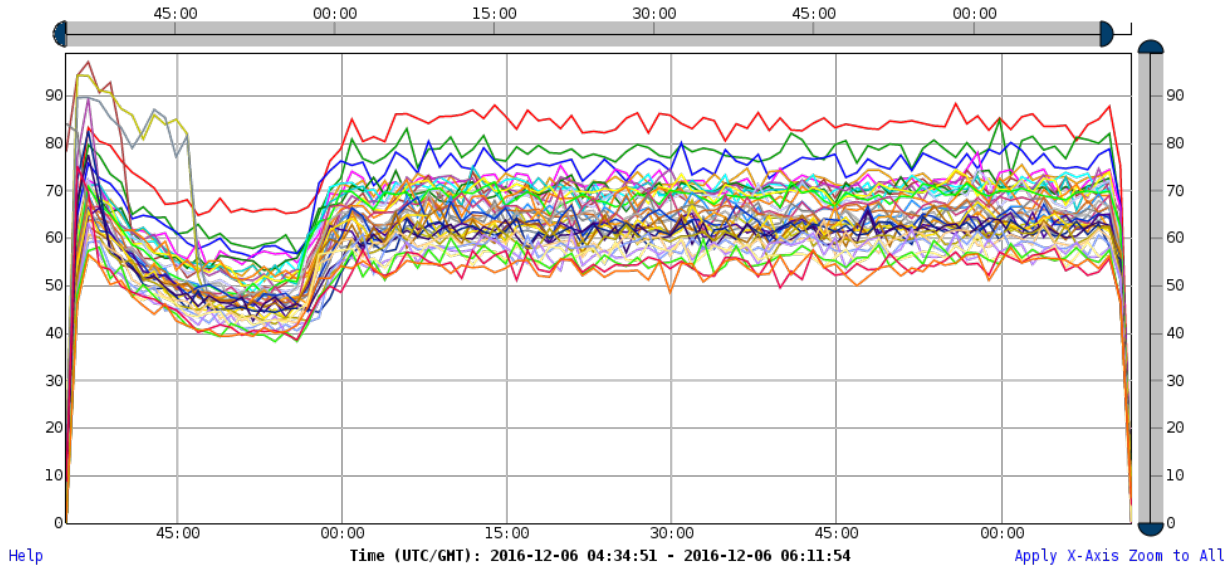
approximately 20% and the highest at 100%. Assuming random I/O throughput is limited by the maximum HDD utilization, then a much higher throughput could be achieved if all the HDDs had even utilization.

Note the uneven disk utilization levels across the 36 OSDs.



OSD Disk Utilization

The PG count for the volumes pool was doubled to 65536 to better spread the data distribution more evenly across the OSDs. The images pool PG count was decreased to 1024 as the pool was using less percentage of data than initially estimated. Note the disk utilization levels are higher and more evenly distributed.



OSD Disk Utilization

See the section on Performance Statistics for information regarding the statistic collection and graph generation.

Some time after initial testing began, a rack consisting of seven OSD nodes was removed from the

configuration, reducing the total OSD count to 787. Because the count had been reduced by roughly 25%, the PG levels were then reduced by the same amount which resulted in using 49152, 3072 and 768 for volumes, vms and images, respectively.

2.3.2. Operating System Settings

Some RHEL 7.3 settings on the OSD nodes as well as RHOSP compute nodes were altered for use with a RHCS cluster.

OSD Process Identifiers Limit - kernel.pid_max

The total number of available system threads (i.e., the number of simultaneous processes that can run in separate memory spaces) is determined by `kernel.pid_max` which defaults to 32768 but a large number of PGs can require a greater value. Although most of these threads are idle until an OSD repair/backfill is initiated, they are still part of the ultimate limit. RHCS typically recommends increasing kernel.pid_max to a very high value.

libvirt max_files and max_processes

In this RHOSP configuration, the the file descriptor limits for libvirtd had to be increased in order to avoid problems with the computes communicating with the ceph monitors resulting in qemu-kvm I/O hangs. The RHEL 7 default setting for `libvirt max_files` and `max_processes` is 1024, which is greatly insufficient for deployment with a RHCS cluster. Both max_files and max_processes were increased by a factor of 32.

2.4. Red Hat OpenStack Platform

RHOSP 10 was installed and configured via a RHOSP director (RHOSPd) node using Ansible playbooks to deploy the undercloud and RHOSP configuration files. See Appendix: References for the location of the playbook.

The overcloud consists of three highly available controller nodes and 20 compute nodes. This configuration is somewhat sub-optimal in a couple areas:

1. It did not have a representative ratio of RHCS clients (i.e., RHOSP compute nodes) to RHCS servers. Typically there are many more compute nodes than RHCS servers.
2. For observing sequential I/O performance, an optimal configuration would typically have network bandwidth on clients greater than the bandwidth available to the servers.

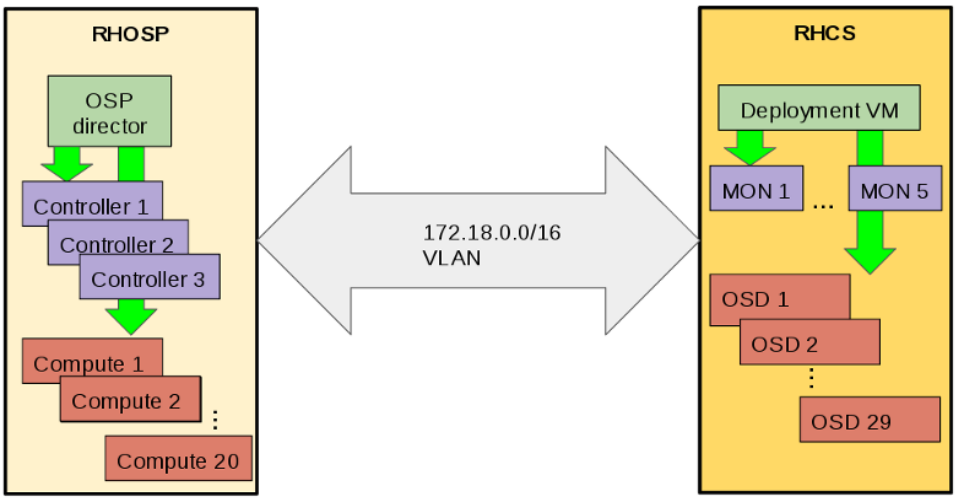
RHOSPd was instructed to deploy no RHCS servers. Nova, Glance, and Cinder were configured to use an existing, external RHCS cluster described in Section Red Hat Ceph Storage using the procedure documented in Red Hat Ceph Storage for the Overcloud, specifically chapter 3: Integrating an Existing Ceph Storage Cluster With an Overcloud. The .yaml file used to link to the external RHCS cluster can be found in Appendix: puppet-ceph-external.yaml with the necessary changes highlighted.

Neutron was configured with:

- the control plane on a 10-GbE public interface with Jumbo Frames (MTU=9000).
- RHCS on a 40-GbE interface with Jumbo Frames (MTU=9000).
- tenant traffic using a 10-GbE interface with Jumbo Frames (MTU=9000).
- isolation using VXLAN.

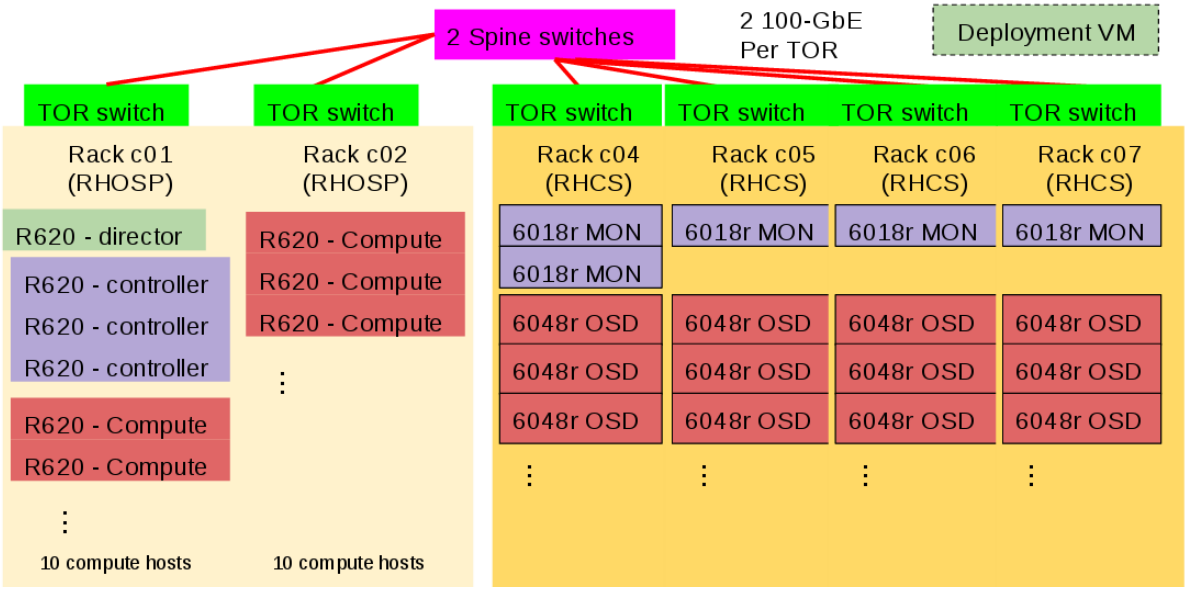
To improve repeatability of the throughput test results to the thinly provisioned RBD volumes, blocks are pre-allocated using dd prior to formatting them with an XFS filesystem. This is done because RHCS does not allocate or initialize the RBD volume when it is created so the first write to a Cinder block will incur greater overhead than subsequent writes where allocation is not required.

The following figure illustrates the RHOSP and RHCS connectivity.



Test Environment Connectivity

This figure identifies the RHOSP RHCS hardware used for each role.



Test Environment Hardware Roles

2.4.1. Issues

Some integration issues with RHSOP and RHCS have been found and documented as bugzilla reports. These problems have either been fixed or are being fixed in an upcoming release. See Appendix: Issues for further details. With the preceding exceptions, the system installed and performed well with respect to hardware utilization for a variety of workloads in this particular configuration. That being said, this configuration did not have a representative ratio of compute nodes to RHCS servers. See the Test Workload section for additional discussion.

The remaining issues, maximum latency and uneven OSD utilization, were observed but are not yet resolved. Aditional work will be required to better understand them. Note that no problems with 99% latency percentile were observed, which were typically well under one second and only reached two seconds briefly during simulated hardware failure/recovery events.

Chapter 3. Test Workload

This project used random reads and writes to measure the effect of scaling instances and a mixed random read and write workload for all fault insertion tests. Due to the aforementioned sub-optimal configuration, testing focused on random I/O performance with smaller transfer sizes rather than sequential I/O performance to avoid being bottlenecked at the network layer.

3.1. Test Design

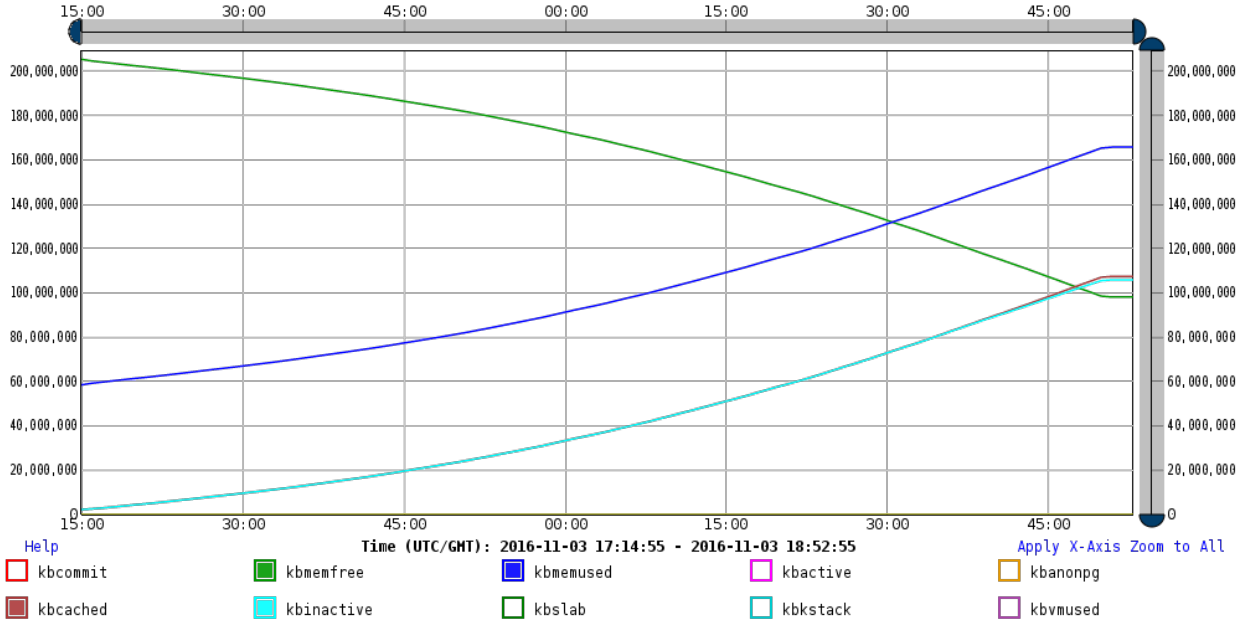
All I/O tests are designed to:

- represent steady state behavior by using random I/O test durations of 10 min + 10 sec/guest to ensure run times are sufficient to fully utilize guest memory and force cache flushes to storage. While the instance scaling tests utilized the calculated runtime durations, fault insertion testing did not and instead ran for the duration of RHCS recovery.
- ensure data travels the full I/O path between persistent storage and application by:
  - using vm.dirty_expire_centisecs in conjunction with vm.dirty_ratio.
  - dropping all (server, compute, guest) caches prior to start of every test.
  - ensuring workload generators do not read or write the same file or offset in a file more than once, reducing opportunities for caching in memory.
  - using O_DIRECT with random I/O to bypass guest caching.
- use a data set an order of magnitude larger than aggregate writeback cache size to ensure data is written to disk by end of test.
- push I/O to its limits (excluding low instance count random I/O tests due to rate limiting) with no regard for system operational safety margins that would normally be taken into consideration in production environments.

To produce more repeatable test results and avoid RHCS caching data in memory, the following modifications were applied to the test environment:

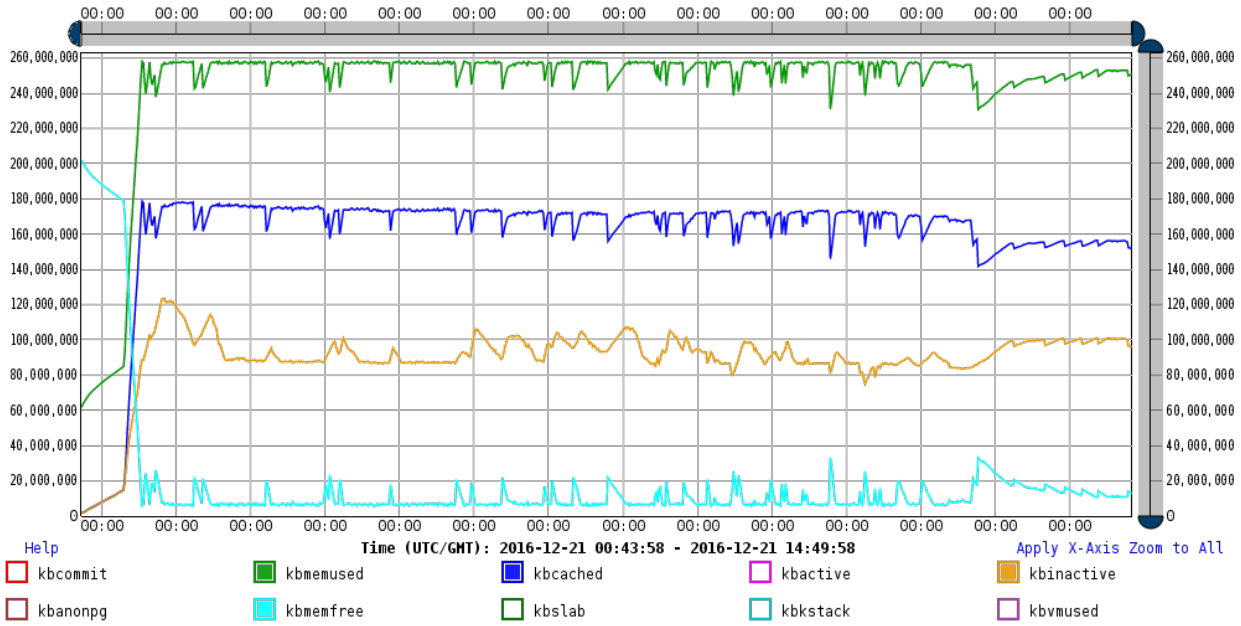
1. the cinder volume sizes were increased from 16GB to 100GB to avoid fitting within memory cache.
2. the FIO iodepth (how many I/Os it issues to the OS at any given time) was increased from 2 to 4.
3. the FIO ioengine was changed from sync to libaio.

The following figures highlight the RHCS memory (in KB) and OSD disk utilization before and after the changes.



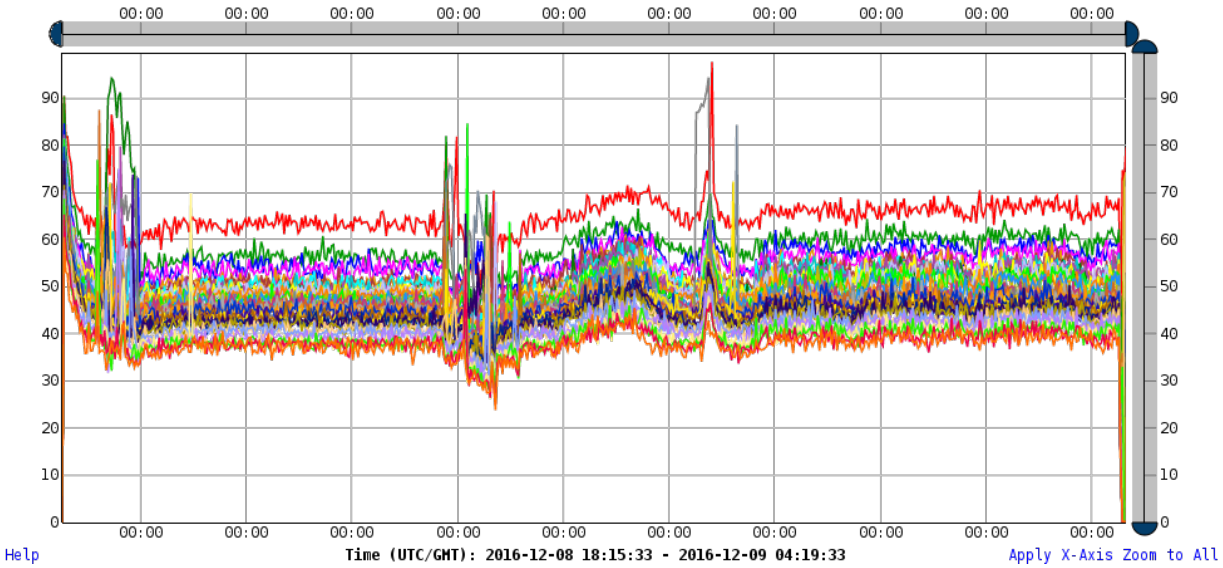
OSD Node Memory (KB) Usage (prior to workload changes) - 512 Instance Random I/O

Using all memory stabilized the workload results and eliminated data caching.



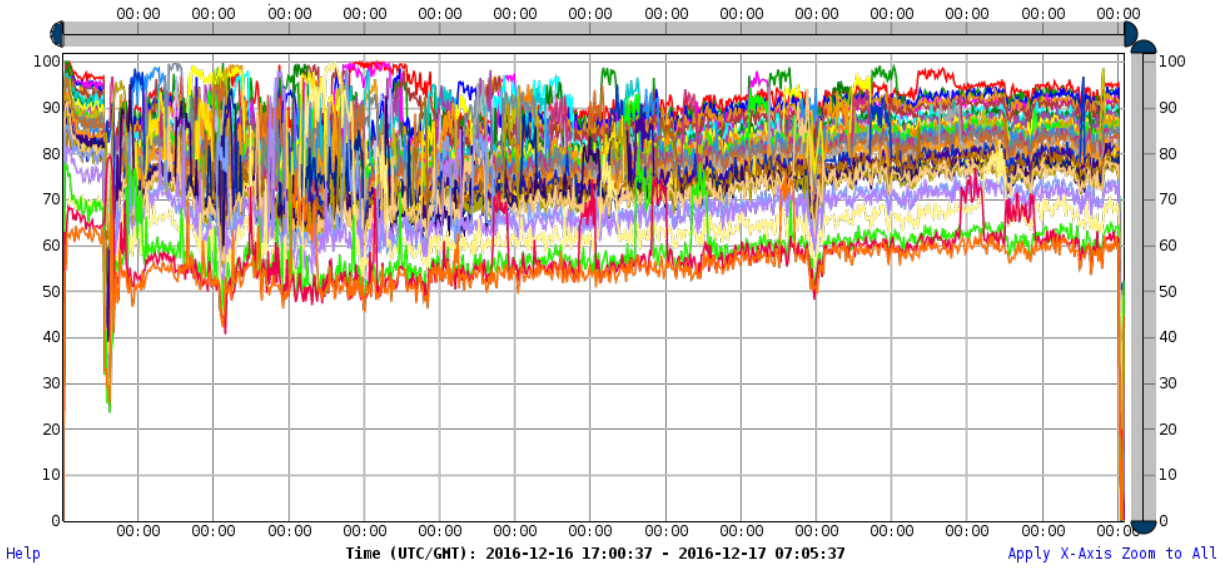
OSD Node Memory (KB) Usage - 512 Instance Random I/O

Before using libaio, only approximately 50% utilization was achieved from our HDDs on average.



OSD Disk Utilization (prior to workload changes) - 512 Instance Random I/O

The OSD utilization after altering the workload.



OSD Disk Utilization - 512 Instance Random I/O

With the partial exception of IOPS limited random I/O tests, all instances are running at the maximum throughput that the hardware will allow. In a more real world example, instances often do not saturate the RHCS cluster. The RHOSP RHCS hardware configuration should be designed to avoid the potential for over-subscribing the storage system.

3.2. Performance Statistics

Performance statistics were collected during testing using the Pbench Benchmarking and Performance Analysis tool. `pbench-uperf` was used to test network speeds before testing began and `pbench-fio` was used to auto start and stop performance statistic collections (iostat, sar) before and after FIO tests. Additionally, pbench auto-generates graphs for the duration of the test for each statistical tool configured to execute.

3.3. Network

`pbench-uperf` (see the section on Performance Statistics) was used for verifying both stream and request-response network speeds. The network flow was designed to correspond to the manner in which RHCS works, where OSD hosts are both sending and receiving requests to which they must respond. In the command syntax:

```
pbench-uperf -t rr -p tcp -i 8 -r 60 -m 4096,131072,4194304 -p tcp -C
host1,host2,host3,...,hostN -S hostN,host1,host2,...,hostN-1
```

host[k] sends traffic to host[k-1 mod N] using 4K, 128K and 4M request/response sizes. See Appendix: Network Tests for a description of the network testing methodology and the issue discovered and resolved resolved by doing so. Any multi-rack configuration should use a similar network testing methodology to establish network performance prior to production.

3.4. Large File Random I/O

Large-file random multi-stream reads and writes using an FIO based workload for testing pure random I/O on Cinder volumes. This workload executes a random I/O process inside each instance in parallel using options to start and stop all threads at approximately the same time. This allows aggregation of the per-thread results to achieve system-wide IOPS throughput for RHOSP on RHCS. Additionally, FIO can rate limit throughput, run time and IOPS of individual processes to measure instance scaling throughput as well as the OSD node's aggregate RHOSP instance capacity.

The following rate limits were applied for scaling VM workloads:

- maximum 35 IOPS per instance
- maximum run time

```
/usr/local/bin/fio --client=/root/vms.list.512 --max-jobs=512 --output-format=json
fio.job
```

This is the FIO job file used with the previous FIO command where runtime is the calculated maximum run time (10 minutes plus 10 seconds per instance).

```
[global]
ioengine=libaio
bs=4k
iodepth=4
direct=1
fsync_on_close=1
time_based=1
runtime=5720
clocksource=clock_gettime
ramp_time=10
startdelay=20

[fio]
rw=randrw
size=95g
write_bw_log=fio
write_iops_log=fio
write_lat_log=fio
write_hist_log=fio
numjobs=1
per_job_logs=1
log_avg_msec=60000
log_hist_msec=60000
directory=/mnt/ceph/fio
```

3.4.1. FIO Latency Histogram Logging

In order to capture for the first time a measurement of application latency percentiles as a function of time, a brand-new measurement method was added to FIO by the Performance & Scale team, FIO latency histogram logging.

The FIO benchmark has always had histograms for measuring latency percentiles over the duration of the entire test and has a very creative way of generating histogram buckets that encompass the huge range of possible latency values, from one microsecond up to seconds. This is documented in `fio/stat.h` comments and the code to convert a latency to a histogram bucket array index is in a small routine `plat_val_to_idx()` in `stat.c`. However, latency percentiles were only calculated at the end of the test and consequently nothing could be said about how those latency percentiles varied over the duration of the test. For example, to search for latency spikes when a simulated hardware failure occurred, it would require many short FIO tests. This approach does not work for long-duration tests as the RHCS backfill process can take up to 12 hours in some cases.

Instead, upstream FIO was changed to periodically output the in-memory histogram that each FIO

process was maintaining. Additionally, it output the change in the histogram buckets instead of the raw counters. The resulting records have this CSV (comma-separated-value) format:

- field 1: timestamp in milliseconds
- field 2: I/O direction (0 = read, 1 = write)
- field 3: blocksize (I/O transfer size in bytes)
- fields 4-1218: histogram buckets

A typical histogram log record looks like this:

```
60020, 0, 4096, 0, 0, ...  1, 2, 2, 1, ....
```

where the timestamp is 60.02 seconds, the I/O direction is 'read', and the blocksize is 4KB. Unfortunately this output format does not provide the location of the histogram bucket latency boundaries. A dump-fio-histo-log.py program is available to generate the histogram latency bucket boundaries aligned with the fields. When executed on a histogram log, the first row will contain the lowest latency in each bucket and the bucket values will align with that row so each line contains the latency for that bucket. When using this program, note that the maximum latency that can be precisely measured by this histogram bucket approach is approximately 17 seconds. Latencies higher than that are counted in the last histogram bucket. This is why some of the graphs below appear cut off at this level. Some of the latencies observed in the fio_clat*log files were much higher, in the hundreds of seconds. This can be resolved by recompiling FIO with more histogram bucket *groups*.

An fiologparser_hist.py tool was written to merge these histograms from different FIO processes for calculation. See Appendix: References for the location of these tools. fiologparser_hist.py is also part of the pbench-fio RPM available as part of the `pbench` toolset which postprocesses the output of fiologparser_hist.py to generate graphs of latency percentiles as a function of time (see the section on Performance Statistics).

These lines were added to the FIO job files:

```
write_hist_log=1
log_hist_msec=60000
```

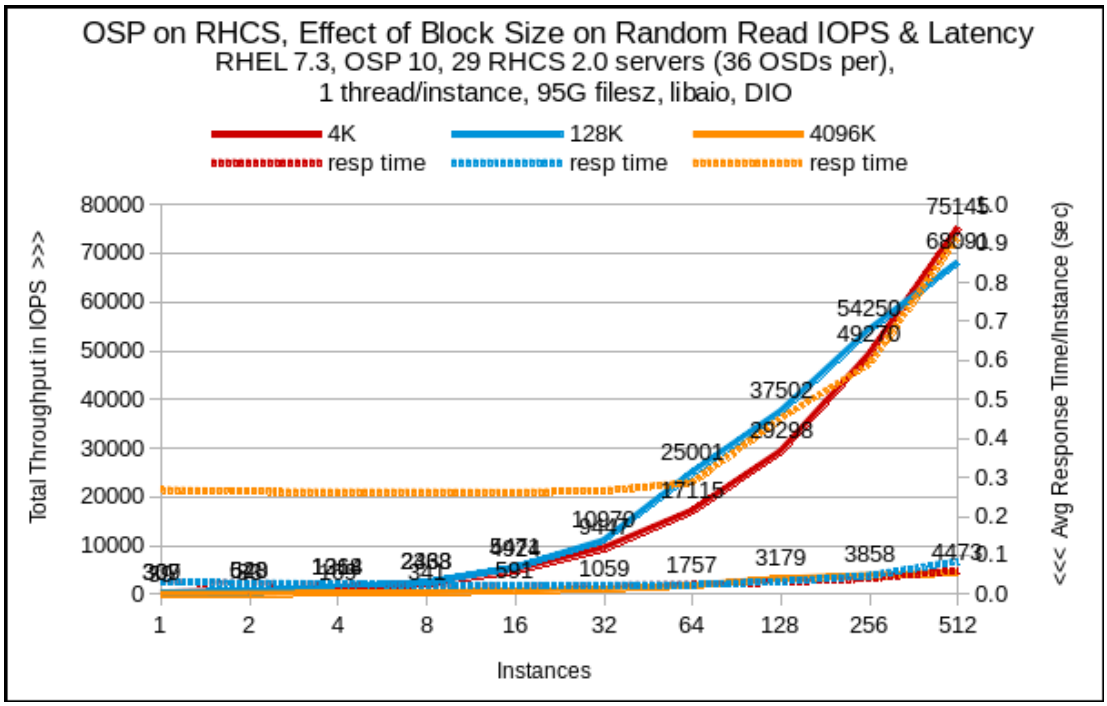
which instruct FIO to output a histogram record into the log every 60 seconds resulting in a set of files with names such as fio_clat_hist.1.log.vmNNN. There is a single job per virtual machine, thus write_hist_log=1 is fixed. The Nova instance hostname is appended to the log file when it is copied to the test driver. The interval of 60 seconds may seem like a long time, but when running a multi-hour test and 512 processes are emitting these records, a longer interval is preferred so the merge tool (fiologparser_hist.py), which is currently single-threaded, can process the results in a reasonable amount of time. Even under these circumstances, the fiologparser_hist.py processing had to be deferred until later so that the testbed could be made available for the next test. Future work will include speedier post-processing as fiologparser_hist.py should be able to run with multiple cores.

Chapter 4. Test Results

Our traditional I/O scaling tests were performed using increasing instance counts. All multi-instance tests ensured even distribution across compute nodes. The results of all testing are in the following sections.

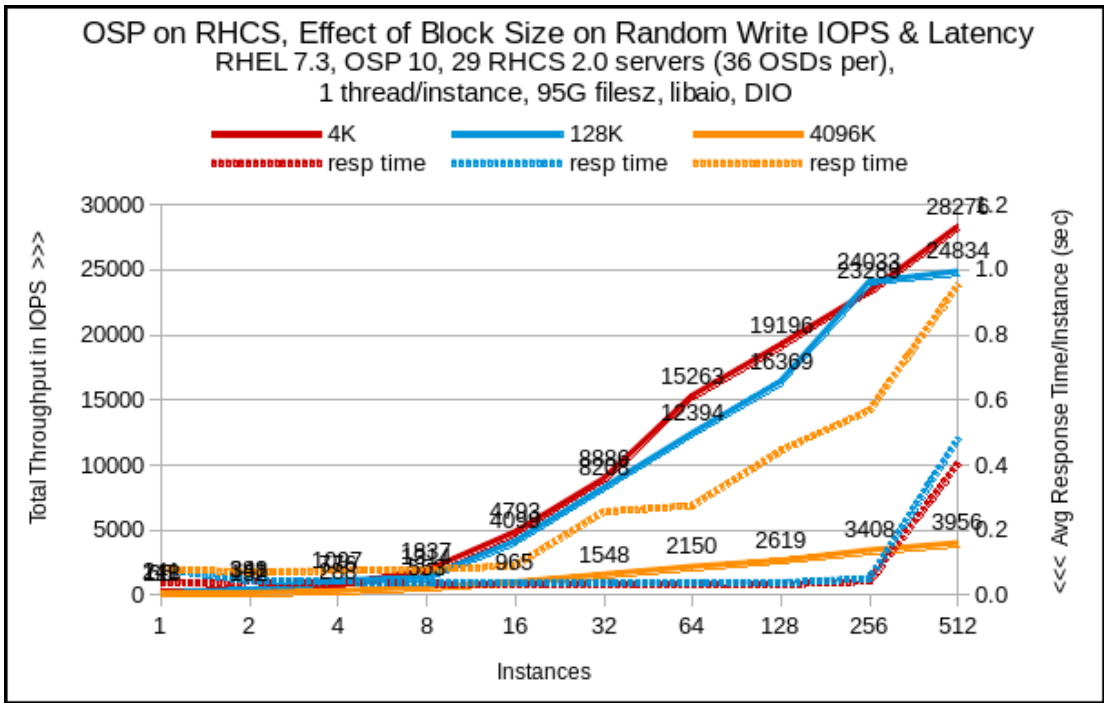
4.1. Effect of I/O Block Size

The following figures highlight the effect of varying block sizes on random read and write IOPS and 95th percentile latencies.



Effect of Block Size - 512 Instance Random Reads

This workload does nothing but I/O as fast as it can. In a real-world application, perhaps not all guests would be doing I/O simultaneously so the system administrator may be able to scale to 1024 guests without incurring additional latency.



Effect of Block Size - 512 Instance Random Writes

Note that the 128KB random reads are close to 4KB random reads in performance. This is somewhat logical given the seek time is so much larger than the transfer time and the networks are extremely fast. Latencies are sub-millisecond for both but for 4MB random reads, latency starts to climb. At 512 instances, throughput is approximately 5000 IOPS which means transfers are occurring at 20 GB/sec, close to line speed for the 20 compute hosts, each of which has a 10-GbE NIC.

At 512 instances, speeds approximately 4000 IOPS are achieved from the cluster, which corresponds to 16 GB/sec of application I/O. However, RHCS is doing 3-way replication so 48 GB/sec were actually transmitting across the network, or roughly 1.65 GB/sec/server. The network inter-rack request-response test was able to push 17 Gbit/sec/server (2.1 GB/s/server) using a 4 MB request size, so this is approximately 80% of line speed and not a bad result when considering the network is operating in full-duplex mode.

Given the results, a block size of 4KB was chosen for all scaling and fault insertion tests to avoid any bottleneck in the network or compute node layer.

4.2. RHCS Recovery

RHCS recovery is made possible by its data replication across multiple OSDs. Replication occurs at the PG layer, the degree of which (i.e., the number of copies) is asserted at the storage pool layer, and all PGs in a pool will replicate stored objects into multiple OSDs. In our 3-way replicated test environment, every object stored in a PG is written to three different OSDs.

When OSDs are added or removed in a cluster, RHCS will *rebalance* (redistribute objects across the cluster) by moving PGs to or from RHCS OSDs to restore balance. The process of migrating PGs and the

objects they contain can reduce the operational performance of the cluster considerably. To maintain operational performance, RHCS performs this migration with *backfilling* (scanning and synchronizing the entire contents of a PG) which runs at a lower priority than read or write requests. RHCS uses backfilling for rebalancing when a data loss is incurred and for *repatriation* (returning data to its original location) when lost hardware is replaced.

4.3. Fault Insertion

This testing was performed to evaluate the impact of RHCS recovery on I/O performance. The test plan included the loss of:

- a single OSD
- an entire OSD node
- a RHCS monitor node

during a mixed random read/write workload.

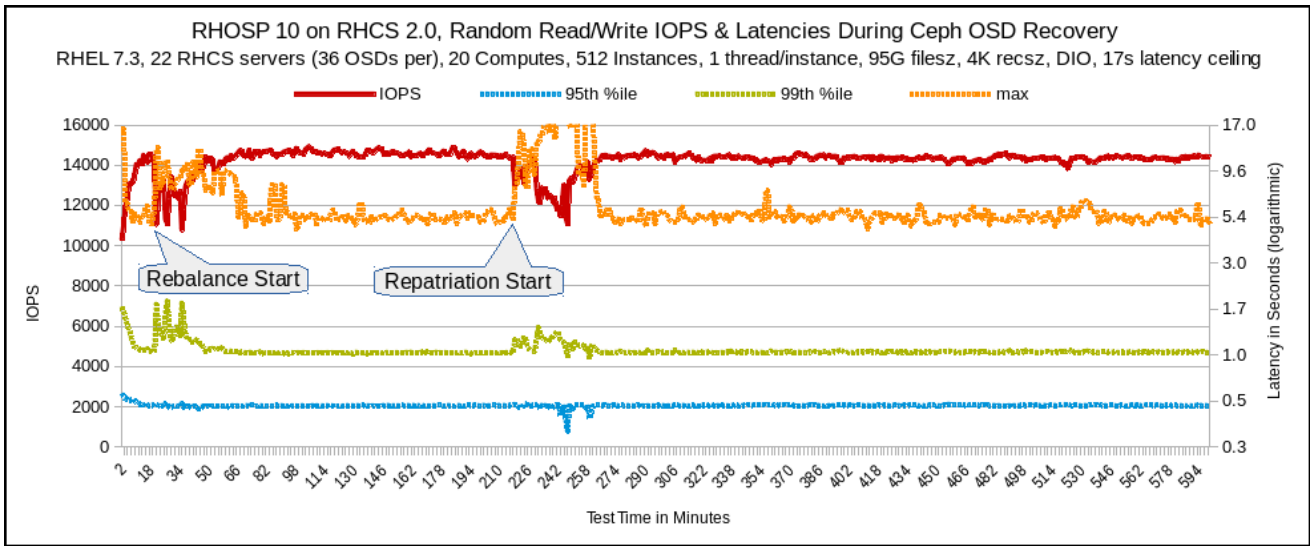
4.3.1. Tuning for Recovery

Initial fault insertion tests observed high maximum latencies. To better control the OSD disk utilization levels, IOPS rate limiting (maximum 35 per instance) was used for fault insertion testing.

The following tuning was applied to the RHCS OSD nodes and the RHOSP compute nodes to prevent queuing in front of application I/O requests in an effort to reduce the high maximum latencies.

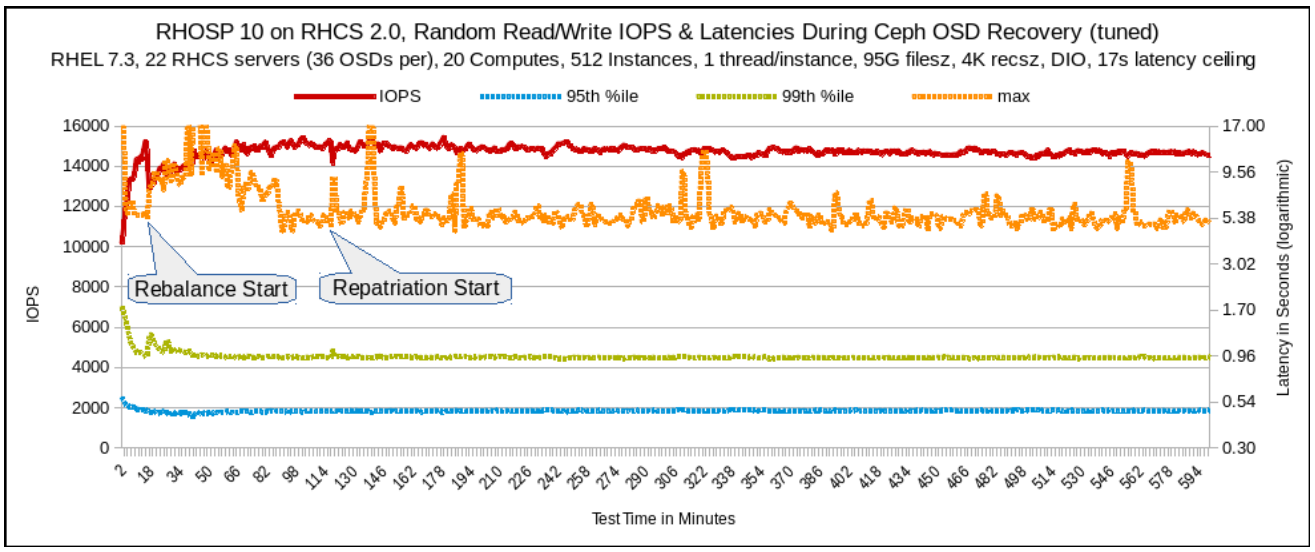
- RHCS OSD nodes:
  - disable RHCS deep scrubbing
  - reduce vm.dirty_ratio from 40 to 10
  - reduce vm.dirty_background_ratio from 10 to 5
  - reduce /sys/block/sd*[a-z]/bdi/max_ratio from 100 to 10
  - disable transparent hugepages (THP)
  - set /sys/block/sd*[a-z]/queue/nr_requests to 64 (default 128)
- RHOSP compute nodes:
  - disable THP and kernel shared memory (KSM)

The following figures graph the IOPS and call latencies during a mixed random read/write workload including the loss and replacement of an OSD. As described above in FIO Latency Histogram Logging, FIO reports a maximum 17 second latency regardless if the latency exceeded that time.



Effect of OSD Loss (untuned) - 512 Instance Random Reads/Writes

Note the spike in both maximum and 95th percentile latencies as well as its impact on IOPS in the previous figure when the downed OSD is replaced (repatriation) as opposed to the following figure after tuning for recovery.



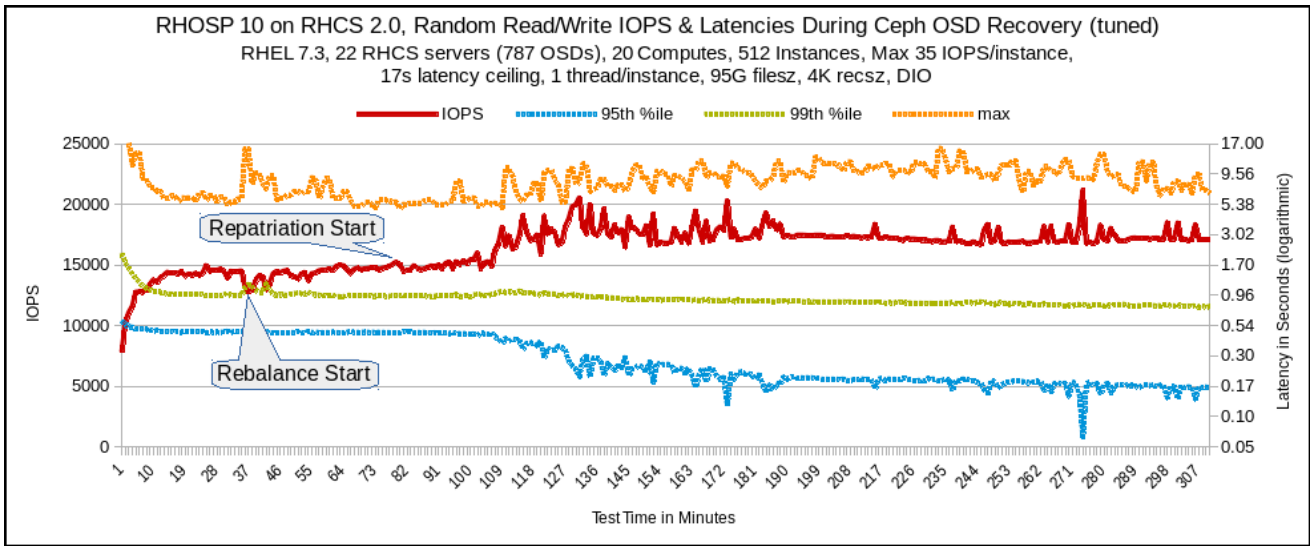
Effect of OSD Loss (tuned) - 512 Instance Random Reads/Writes

In the tuned recovery test, when rebalance starts there is not as much of a latency spike in the 99th percentile as there is in the untuned recovery. However, maximum latency seems worse.

Repatriation happens much sooner in the tuned recovery test because backfilling for the rebalance completed more quickly. There seems to be much lower duration in the maximum latency spike for repatriation in the tuned test. Additionally, there is no increase in 99% latency in the tuned test, unlike the untuned test.

4.3.2. OSD Loss

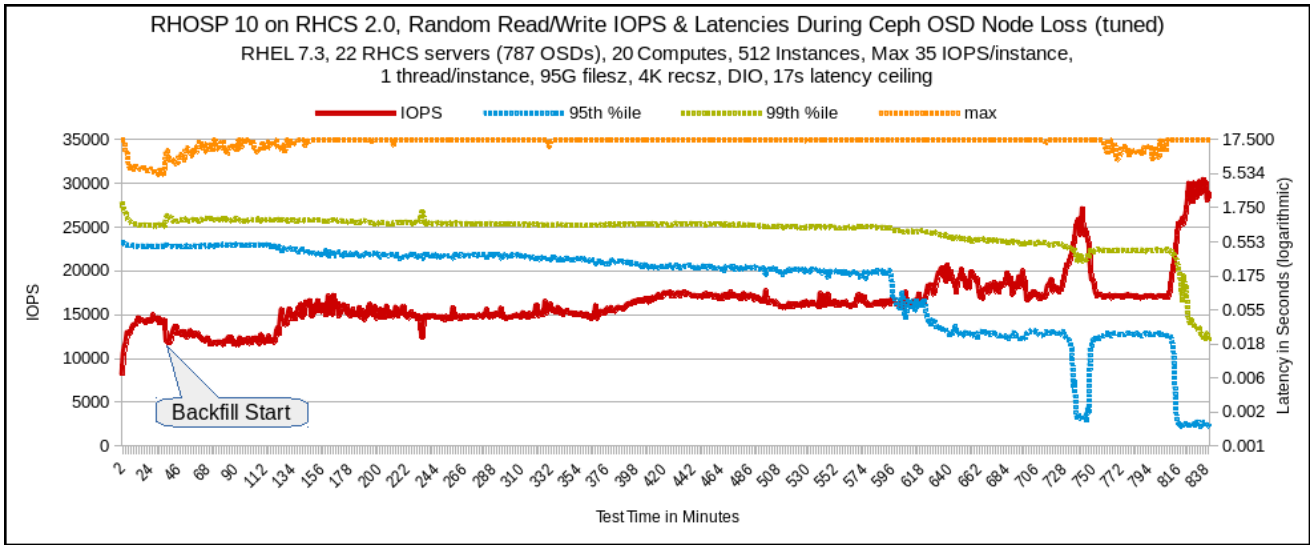
This testing implemented a maximum 35 IOPS per instance rate limit (by adding `rate_iops=35` to the FIO job file) and used the aforementioned Tuning for Recovery for reducing maximum latency. Recall that FIO reports a maximum 17 second latency regardless if the latency exceeded that time but only two such cases are seen (this omits the case seen on the left X-axis boundary because of the cache drop that occurs before each test).



Effect of OSD Loss - 512 Instance Random Reads/Writes

4.3.3. OSD Node Loss

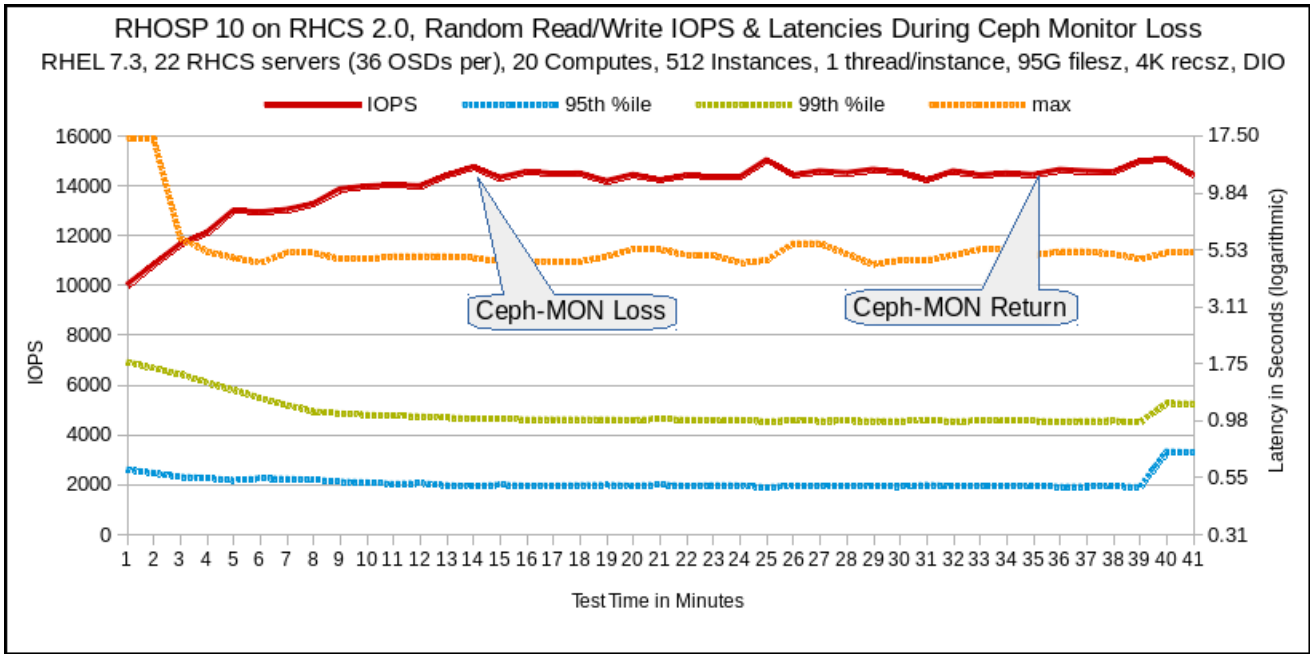
This is an obviously very poor maximum latency result. As soon as the backfill begins, unacceptable maximum latency is observed even though 99% latency is acceptable. For future study, this suggests the need to have more PGs backfilling but with a limited rate of backfilling to reduce the maximum latency.



Effect of OSD Node Loss - 512 Instance Random Reads/Writes

4.3.4. RHCS Monitor Loss

This figure indicates that the monitor loss did not seem to significantly affect latency at all. This test halted a ceph monitor node, simulated the loss of its database, and brought it back (including database restore). This occurred without application disruption.



Effect of Monitor Loss - 512 Instance Random Reads/Writes

Appendix A: References

1. *Enabling Ceph Repositories* - https://access.redhat.com/documentation/en/red-hat-ceph-storage/2/paged/installation-guide-for-red-hat-enterprise-linux/chapter-2-prerequisites#prereq-enable-ceph-ga-repo
2. *Ceph PGs per Pool Calculator* - https://access.redhat.com/labs/cephpgc/
3. *Red Hat Ceph Storage for the Overcloud* - https://access.redhat.com/documentation/en/red-hat-openstack-platform/10/paged/red-hat-ceph-storage-for-the-overcloud/
4. *Integrating an Existing Ceph Storage Cluster With an Overcloud* - https://access.redhat.com/documentation/en/red-hat-openstack-platform/10/paged/red-hat-ceph-storage-for-the-overcloud/chapter-3-integrating-an-existing-ceph-storage-cluster-with-an-overcloud
5. *Pbench Benchmarking and Performance Analysis Framework* - http://distributed-system-analysis.github.io/pbench/
6. *bz 1388119: Can't snapshot image from nova instance (on RHOSP10)* - https://bugzilla.redhat.com/show_bug.cgi?id=1388119
7. *tr 17573: Librbd hangs because of failure to create socket* - http://tracker.ceph.com/issues/17573
8. *bz 1384079: Openstack OOO client disregards ceph keys from YAML* - https://bugzilla.redhat.com/show_bug.cgi?id=1384079
9. *bz 1375378: Document ceph-ansible method for host with different block device names* - https://bugzilla.redhat.com/show_bug.cgi?id=1375378
10. *bz 1389502: Director should increase kernel.pid_max on ceph-backed compute nodes* - https://bugzilla.redhat.com/show_bug.cgi?id=1389502
11. *bz 1389503: Director should increase libvirtd FD limits on ceph-backed compute nodes* - https://bugzilla.redhat.com/show_bug.cgi?id=1389503
12. *dump-fio-histo-log.py* - https://github.com/bengland2/fio/blob/dump-fio-histo-log/tools/dump-fio-histo-log.py
13. *fiologparser_hist.py* - https://github.com/axboe/fio/blob/master/tools/hist/fiologparser_hist.py

Appendix B: Tuned Profiles

Below are the system settings applied by tuned for each profile. The differences from the default throughput-performance profile are highlighted.

B.1. throughput-performance

CPU governor = performance
energy_perf_bias=performance
block device readahead = 4096
transparent hugepages = enabled
kernel.sched_min_granularity_ns = 10000000
kernel.sched_wakeup_granularity_ns = 15000000
kernel.sched_migration_cost_ns = 500000
vm.dirty_ratio = 40
vm.dirty_background_ratio = 10
vm.swappiness = 10

B.2. virtual-host

CPU governor = performance
energy_perf_bias=performance
block device readahead = 4096
transparent hugepages = enabled
kernel.sched_min_granularity_ns = 10000000
kernel.sched_wakeup_granularity_ns = 15000000
`kernel.sched_migration_cost_ns = 5000000`
vm.dirty_ratio = 40
`vm.dirty_background_ratio = 5`
vm.swappiness = 10

B.3. virtual-guest

CPU governor = performance
energy_perf_bias=performance
block device readahead = 4096
transparent hugepages = enabled
kernel.sched_min_granularity_ns = 10000000
kernel.sched_wakeup_granularity_ns = 15000000
kernel.sched_migration_cost = 500000
`vm.dirty_ratio = 30`
vm.dirty_background_ratio = 10
`vm.swappiness = 30`

Appendix C: Issues

Ceph tracker 17573 illustrates a problem with librbd where it hangs instead of returning an error status, causing VMs to hang until an I/O timeout occurs, long after they are booted. This is due to RHCS librbd opening TCP sockets only as they are needed instead of opening sockets to all the OSDs immediately. The net effect is that the user thinks the guest is working, and then it stops. This problem can be avoided if RHOSP deploys the guests with higher file descriptor and pid-max limits. This is addressed in rows 5 and 6 of the next table.

Bug ID	Status	Description
bz 1388119	Fixed in RHOSP 10	All nova snapshot attempts (even as admin user) fail to create new image.
tr 17573	Open	librbd does not return a failure to the calling application so the user experiences a hang instead of receiving an error.
bz 1384079	Fixed in RHOSP 10	OOO client disregards the CephClusterFSID and CephClientKey from supplied YAML file
bz 1375378	Fixed in RHCS 2.1	Document ceph-ansible method for host with different block device names. If any disk in the devices: list for ceph-ansible is unavailable ceph-ansible will not start any OSD on that host.
bz 1389502	Fixed in RHOSP 10	kernel.pid_max too low on ceph backed compute nodes. See OSD Process Identifiers Limit - kernel.pid_max for more detail.
bz 1389503	Fixed in RHOSP 11	Insufficient libvirtd file descriptors on ceph backed compute nodes. See libvirt max_files and max_processes for more detail.

Appendix D: ceph.conf

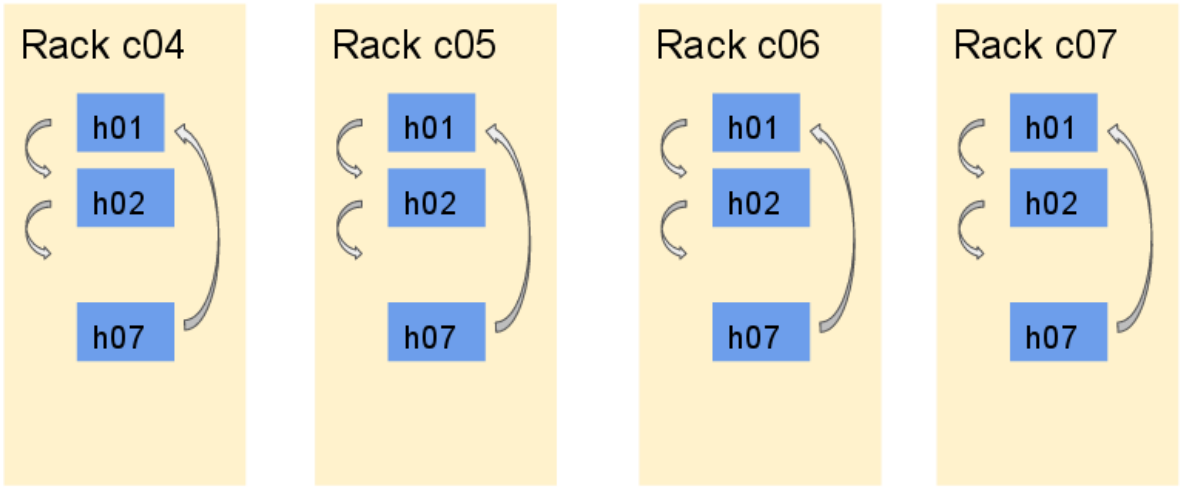
```
[global]
fsid = ae48c07d-36c5-4f3e-bfce-fdb4158bbb07
max open files = 131072

[client]
admin socket = /var/run/ceph/$cluster-$type.$id.$pid.$cctid.asok
log file = /var/log/ceph/qemu-guest-$pid.log

[mon]
[mon.c04-h33-6018r]
host = c04-h33-6018r
mon addr = 172.18.65.121
[mon.c05-h33-6018r]
host = c05-h33-6018r
mon addr = 172.18.65.118
[mon.c07-h29-6018r]
host = c07-h29-6018r
mon addr = 172.18.64.226
[mon.c07-h30-6018r]
host = c07-h30-6018r
mon addr = 172.18.64.238
[mon.c06-h29-6018r]
host = c06-h29-6018r
mon addr = 172.18.64.251

[osd]
osd mkfs type = xfs
osd mkfs options xfs = -f -i size=2048
osd mount options xfs = noatime,largeio,inode64,swalloc
osd journal size = 5000
osd_max_backfills = 1
osd_recovery_op_priority = 1
osd_recovery_max_active = 1
cluster_network = 172.18.0.0/16
public_network = 172.18.0.0/16
```

Appendix E: puppet-ceph-external.yaml

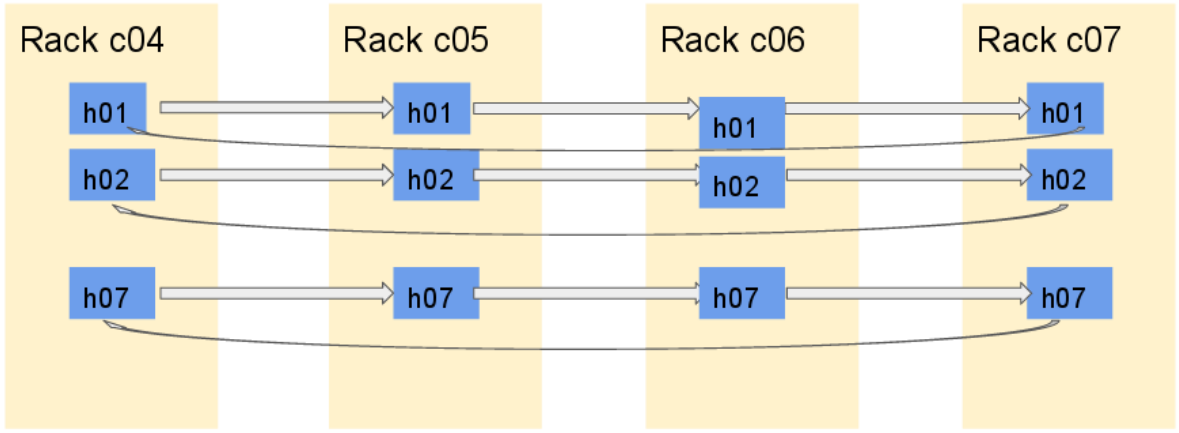
```
resource_registry:
  OS::TripleO::Services::CephExternal: /usr/share/openstack-tripleo-heat-
        templates/puppet/services/ceph-external.yaml
  OS::TripleO::Services::CephMon: OS::Heat::None
  OS::TripleO::Services::CephClient: OS::Heat::None
  OS::TripleO::Services::CephOSD: OS::Heat::None
parameter_defaults:
  CephClusterFSID: 'ae48c07d-36c5-4f3e-bfce-fdb4158bbb07'
  CephClientKey: 'AQC5tz1YQBBWNxAAmR8mIpHuMZstFpaHRyIQtA=='
  CephAdminKey: 'AQApmT1YXW9aORAAesZLbNr6PbeOR6mUMwMKkQ=='
  CephExternalMonHost: '172.18.65.121, 172.18.65.118, 172.18.64.251, 172.18.64.226, 172.18.64.238'
  NovaEnableRbdBackend: true
  CinderEnableRbdBackend: true
  CinderBackupBackend: ceph
  GlanceBackend: rbd
  GnocchiBackend: rbd
  NovaRbdPoolName: vms
  CinderRbdPoolName: volumes
  GlanceRbdPoolName: images
  GnocchiRbdPoolName: metrics
  CephClientUserName: openstack
  CinderEnableIscsiBackend: false
```

Appendix F: Network Testing Methodology

Network validations included verifying I/O speeds within each rack (intra) as well as across racks (inter) as illustrated in the next two figures.

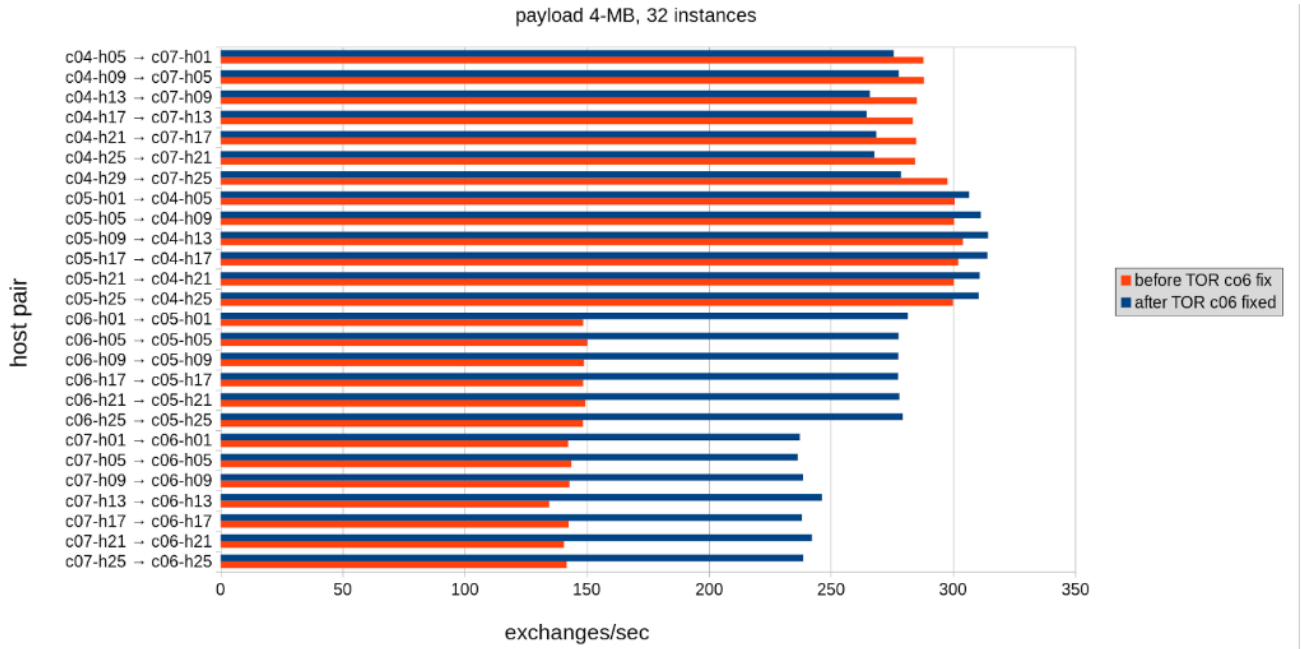


Intra-Rack Network Testing



Inter-Rack Network Testing

A subset of initial tests produced speeds only half that of the rest where one switch was common to all of the suspect results. This revealed an issue with one of the two 100-GbE ports in the top-of-rack switch as seen in the results below.



Inter-Rack Network Throughput


redhat.