ORNL/TM-2013/151

National Center for Computatational Sciences

Ceph Parallel File System Evaluation Report

Feiyi Wang Mark Nelson
ORNL/NCCS Inktank Inc.

Other contributors:

Sarp Oral
Doug Fuller
Scott Atchley
Blake Caldwell
James Simmons
Brad Settlemyer
Jason Hill
Sage Weil (Inktank Inc.)

Prepared by
OAK RIDGE NATIONAL LABORATORY
Oak Ridge, Tennessee 37831-6283
managed by
UT-BATTELLE, LLC
for the
U.S. DEPARTMENT OF ENERGY

This research was supported by, and used the resources of, the Oak Ridge Leadership Computing Facility, located in the National Center for Computational Sciences at ORNL, which is managed by UT Battelle, LLC for the U.S. DOE (under the contract No. DE-AC05-00OR22725).

Contents

List of Figures

1 Introduction

National Center for Computational Sciences (NCCS), in collaboration with Inktank Inc, prepared this performance and scalability study of Ceph file system. Ceph originated from Sage Weil's PhD research at UC Santa Cruz around 2007 and it was designed to be a reliable, scalable fault-tolerant parallel file system. Inktank is now the major developer behind the open-source parallel file system to shepherd its development and provide commercial support.

In comparison to other parallel file systems, Ceph has a number of distinctive features:

- Ceph has an intelligent and powerful data placement mechanism, known as CRUSH. The CRUSH algorithm allows a client to pre-calculate object placement and layout while taking into consideration of failure domains and hierarchical storage tiers.
- From the start, Ceph's design anticipated managing metadata and the name space with a cluster of metadata servers. It utilized a dynamic subtree partitioning strategy to continuously adapt metadata distribution to current demands.
- Ceph's design assumes that the system is composed of unreliable components; fault-detection and fault-tolerance (e.g., replication) are the norm rather than the exception. This is in line with the expectations and future directions of Exascale computing.
- Ceph is built on top of a unified object management layer, RADOS. Both metadata and the file data can take advantage of this uniformity.
- Most of the Ceph processes reside in user-space. Generally speaking, this makes the system easier to debug and maintain. The client-side support has long been integrated into Linux mainline kernel, which eases the deployment and out-of-box experience.

As part of this effort, we set up a dedicated testbed within NCCS for the Ceph file system evaluation. The goal of our study is to investigate the feasibility of using the Ceph for our future HPC storage deployment. This report presents our experience, results, and observations. While evaluating our results, please keep in mind that Ceph is still a relatively *young* parallel file system and its code base is changing rapidly. In between releases, we often experienced different stability and performance outcomes. We will try to make clear in the report when such changes occurred.

This research was supported by, and used the resources of, the Oak Ridge Leadership Computing Facility, located in the National Center for Computational Sciences at ORNL, which is managed by UT Battelle, LLC for the U.S. DOE (under the contract No. DE-AC05-00OR22725).

2 Testbed Environment Description

We used Data Direct Networks' (DDN) SFA10K as the storage backend during this evaluation. It consists of 200 SAS drives and 280 SATA drives, organized into various RAID levels by two active-active RAID controllers. The exported RAID groups by these controllers are driven by four hosts. Each host has two InfiniBand (IB) QDR connections to the storage backend. We used a single dualport Mellanox connectX IB card per host. By our calculation, this setup can saturate SFA10K's maximum theoretical throughput (~12 GB/s). The connection diagram is illustrated in Figure 1.

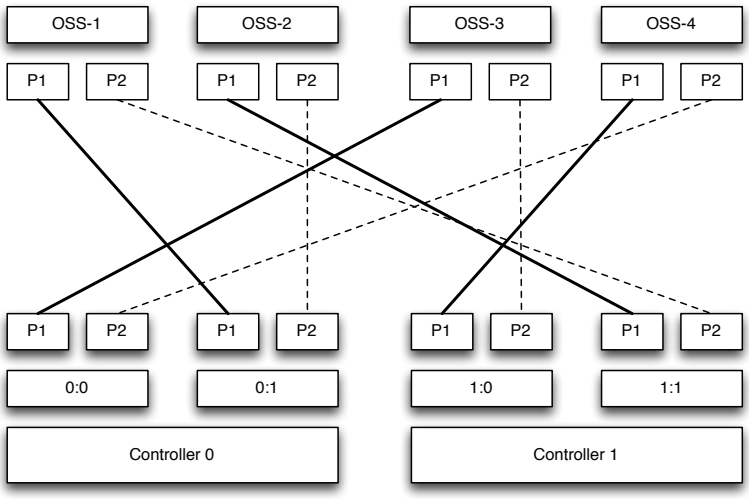

Figure 1: DDN SFA10K hardware and host connection diagram

Our Ceph testbed employs a collection of testing nodes. These nodes and their roles are summarized in Table 1. In the following discussion, we use "servers", "osd servers", "server hosts" interchangeably. We will emphasize with "client" prefix when we want to distinguish it from above.

Node	Role
tick-mds1	Ceph monitor node
spoon46	Ceph MDS node
tick-oss[1-4]	Ceph OSD servers
spoon28-31, spoon37-41	Ceph client nodes

Table 1: Support nodes involved in Ceph testbed

All hosts (client and servers) were configured with Redhat 6.3 and kernel version 3.5.1 initially, and later upgraded to 3.9 (rhl-ceph image), Glibc 2.12 with syncfs support, locally patched. We used the Ceph 0.48 and 0.55 release in the initial tests, upgraded to 0.64 and then to 0.67RC for a final round of tests.

3 Baseline Performance

3.1 Block I/O over Native IB

We first established baseline performance by measuring block I/O performance. At the block-level, with each LUN configured as a RAID6 8+2 array, we had the following results as shown in Table 2.

SAS single LUN sequential read	1.4 GB/s
SATA single LUN sequential read	955 MB/s
Single host with four SAS LUNs	2.8 GB/s
Single host with seven SATA LUNs	2.6 GB/s

Table 2: Block I/O performances on single LUN and single host

Single host in this table refers to one of four tick-oss nodes. Four tick-oss nodes drive the SFA10K backend storage. Overall, we observe that the entire system can perform at 11 GB/s, compared to DDN SFA10K's theoretical maximum of 12 GB/s.

3.2 IP over IB

Ceph uses the BSD Sockets interface in `SOCK_STREAM` mode (i.e., TCP). Because our entire testbed is IB-switched, we used IP over IB (IPoIB) for networking[1]. Through simple `netperf` tests, we observed that a pair of hosts connected by IB QDR using IPoIB can transfer data at 2.7 GB/s (the hosts are tuned per recommendations from Mellanox). With all four hosts (OSD servers), we expect the aggregate throughput to be in the neighborhood of 10 GB/s.

Unfortunately there was not enough time to do more detailed analysis of the network performance. However, as we observed later, RADOS is performing no more than 8 GB/s driven by four server hosts. This confirms that we have provisioned enough network bandwidth. In other words, IP over IB is not a bottleneck in this case.

4 System Tuning

Base upon past experience and further experimentation, we started out with the following set of tuning parameters on OSD servers:

[1] As of writing of this report, Inktank is investigating using rsockets to improve performance with IB fabric

nr_requests	2048
read_ahead_kb	4096
scheduler	deadline

Table 3: System tuning parameters

5 XFS Performance As Backend File System

Ceph supports multiple backend file systems including Btrfs, Ext4, and XFS. Due to its maturity and Inktank's recommendations, we chose XFS.[2] We experimented with XFS to acquire a set of configuration parameters which provided optimal performance for the SFA10K. We sampled a selected set of parameters (block size, queue depth, request size, sequential read and write). We settled on the following configuration: mount with `nobarrier`, `noatim`, `inode64` options. The `inode64` option had a notable improvement on sequential write (around 20%).

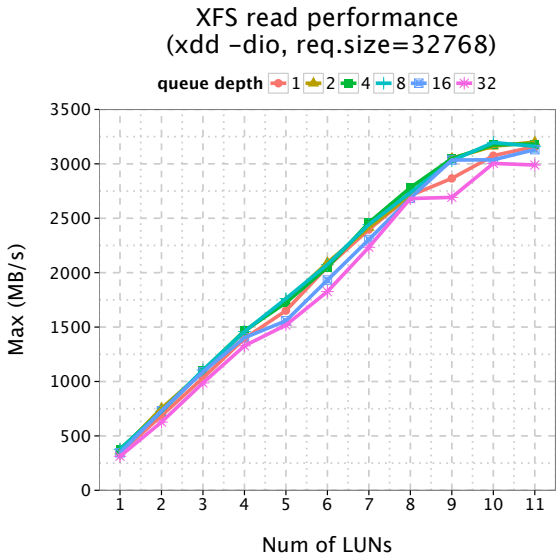

Figure 2: XFS read performance scaling on number of devices

The read and write performance results are summarized in Figures 2 and 3, respectively. These graphs show scaling behavior over the number of LUNs, which indicates that XFS can reach the peak write performance with just five SATA LUNs. Increasing number of LUNs beyond 5, degradation happened. Also, we did not observe any obvious differences in performance when varying the queue depth parameter. For these tests, we used the `xdd` benchmark with direct I/O and a request size of 32 KB.

[2] It should be noted that in testing cases while we can compare XFS vs. Brtfs, Brtfs generally shows better performance results.

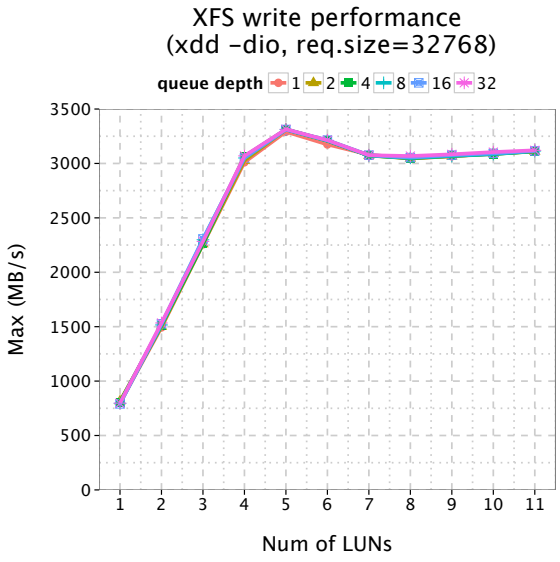

Figure 3: XFS write performance scaling on number of devices

6 Ceph RADOS Scaling: Initial Test

RADOS is a reliable distributed object store, the foundational component for CephFS file system. There are two types of scaling tests we are interested at the RADOS layer:

- scaling on the number of OSD servers
- scaling on the number of OSDs per OSD server

Our system setup poses some limitations on the scalability tests we wanted to perform. In particular, we currently had four OSD servers, eight clients, and eleven OSD servers per client. The scaling tests, therefore, will be within these constraints.

We used the open-source RADOS Bench tool, developed by Inktank, to perform our initial performance analysis of the underlying RADOS layer. RADOS Bench simply writes out objects to the underlying object storage as fast as possible, and then later reads those objects in the same order as they were written.

We observed that using two or more client processes and many concurrent operations are important when performing these tests. We tested eight client processes with 32 concurrent 4 MB objects in flight each. We created a pool for each RADOS Bench process to ensure that object reads come from independent pools (RADOS Bench is not smart enough to ensure that objects are not read by multiple processes and thus possibly cached). A sync and flush is performed on every node before every test to

ensure no data or metadata is in cache. All tests were run with replication set to one. The backend file systems were XFS, BTRFS and EXT4 file systems were not tested at this time.

6.1 Scaling on number of OSDs per server

In the following test, a single Ceph host drives n OSDs, where n increases from one to eleven. The result is illustrated in Figure 4. We ran the test against a single client with four concurrent processes. In this test case, we observe that the OSD server exhibits near linear scalability up to nine OSDs, and is still trending upwards at eleven OSDs. This suggests that we have not reached the saturation point yet. Additional testing would require provisioning more OSDs on the SFA10K backend.

Figure 4: RADOS scaling on number of OSDs

Figure 5: RADOS scaling on number of servers

6.2 Scaling on number of OSD servers

In this test, we exercise OSD servers from one to four, driven by four hosts each with four RADOS Bench process. Each additional OSD server adds eleven more OSDs into play. We observe that Ceph exhibits linear scaling with regard to number of servers as well, at least in the given set of servers. However, the peak performance we are seeing is about the half of what are expecting from the SFA10K (compare to the baseline block I/O performance number presented in Section 3).

For writes, the lost performance is attributed to the way Ceph performs journaling: Ceph does not support meta-data only journaling, therefore every write is the equivalent of a double-write: once to the data device, once to the journaling device. This effectively cuts the observed system bandwidth in half. That said, it does not explain the read performance – it is a little better than write, but still far from the theoretical maximum.

IOR parameter	Note
`-F`	file per process
`-a POSIX`	use POSIX API
`-w -r -C`	do both write and read test, `-C` is to change task ordering for read back so it will not read from the write cache.
`-i 3 -d 5`	3 iterations and delay 5 seconds betewen iterations
`-e`	perform `fsync()` upon POSIX write close
`-b 8g or 16g`	the block size
`-t 4k to 4m`	the transfer size
`-o file`	mandatory test file

Table 4: IOR parameter setup

7 Ceph File System Performance: Initial Test

We used the synthetic IOR benchmark suite for file system level performance and scalability test. The particular parameter setup is show in Table 4. Each client node has 6 GB of physical memory, the block size is set so as to mitigate cache effects. In addition, the test harness program issues the following commands at the beginning of each test:

```
# sync
# echo 3 | tee /proc/sys/vm/drop_caches
```

Here, 0 is the default value of `drop_caches`; 1 is to free pagecaches, 2 is to free dentries and inodes, 3 is to free pagecache, dentries, and inodes.

Our first round of tests was less than ideal as we encountered various issues. For the sake of completeness, we first summarize the results, then discuss further tuning efforts and improvements.

The full permutation of IOR parameters were not explored due to I/O errors we encountered. We were, however, able to record results in two extreme cases as far as transfer size is concerned: 4 KB and 4 MB, using a fixed number of OSD servers (4) and fixed block size (8 GB), the results are illustrated in Figure 6 and 7, we make the following observations:

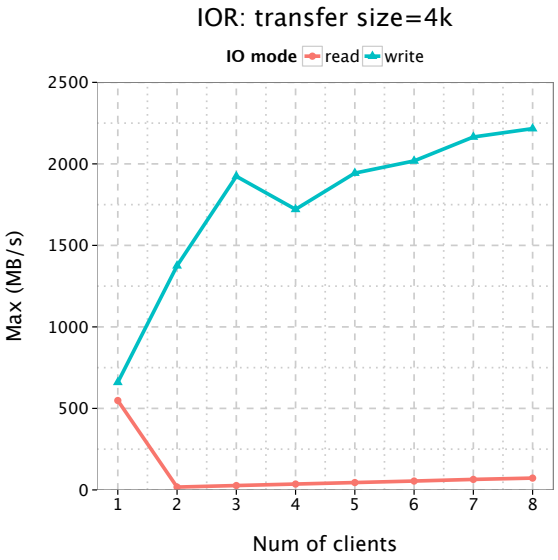

Figure 6: IOR tests: 4 KB transfer size

Figure 7: IOR tests: 4 MB transfer size

- The small read (4 KB transfer size) performance is almost an anomaly – we will investigate why it is so low compare to write performance and present improved results in Section 9.
- The large read (4 MB transfer size) performance is almost half of the RADOS read performance.
- The write performance is also about half of what we can obtain from RADOS Bench. When number of clients reaches 8, there is a significant performance drop as well.

We will describe the efforts and results on performance improvement in the following sections.

8 Improving RADOS Performance

After the initial test results, we tried various combinations of tweaks including changing the number of filestore op threads, putting all of the journals on the same disks as the data, doubling the OSD count, and upgrading Ceph to a development version which reduces the seek overhead caused by `pginfo` and `pglog` updates on XFS (these enhancements are now included as of the Ceph Cuttlefish release, v0.61). The two biggest improvements resulted from disabling CRC32c checksums and increasing the OSD count on the server. With these changes, we are seeing better results.

We ran a script written by Inktank for internal Ceph testing to perform sweeps over Ceph configuration parameters to examine how different tuning options affect performance on the DDN platform. The

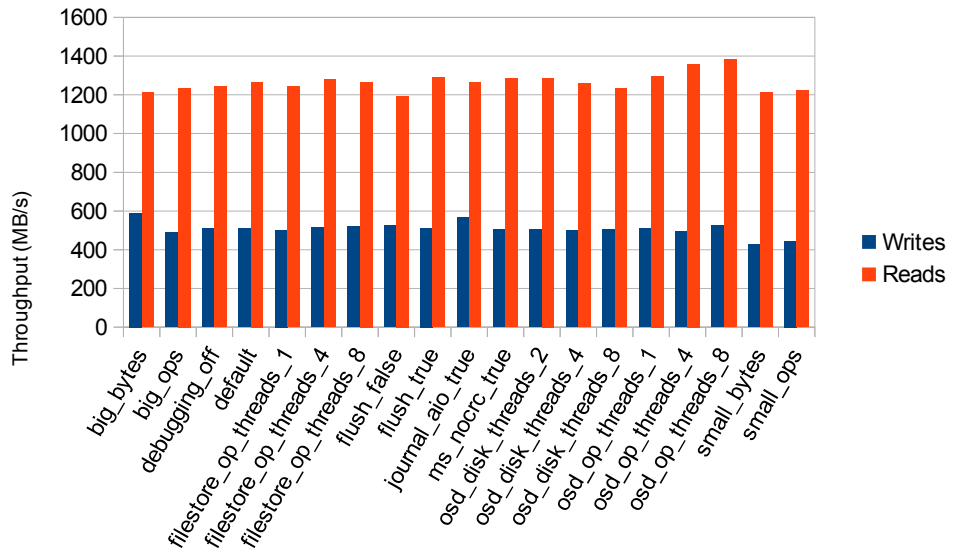

Figure 8: Evaluating parameter impact through sweeping test

result of this parameter probing is illustrated in Figure 8. Please refer to Appendix E for explanations of these probed parameters.

As a result of this testing, we improved performance slightly by increasing the size of various Ceph buffers and queues, enabling AIO journals, and increasing the number of OSD op threads.

8.1 Disable Cache Mirroring on Controllers

During a second round of test performed by Inktank, we noticed a dramatic drop on RADOS performance: even though write throughput on individual server met the expectation, it did not scale across servers.

We spent a significant amount of time investigating this phenomenon. Ultimately, we were able to replicate this finding when running concurrent disk throughput tests directly on the servers without Ceph involved. The second RAID processor on each DDN controller would max out when three or more LUNs were written concurrently. It turns out the root of the problem was a regression on DDN firmware update – in particular, the cache mirroring was not behaving as it should.[3]

With cache mirroring disabled, write performance when using all four servers improved dramatically, as illustrated in Figure 9. With BTRFS, for example, we hit over 5.5 GB/s from the clients. When accounting for journal writes, that is over 11 GB/s to the disks and very close to what the DDN chassis is capable

[3]DDN recently released a new firmware version and we were told the issue has been fixed. Unfortunately, we didn't get a chance to verify it during our test cycle.

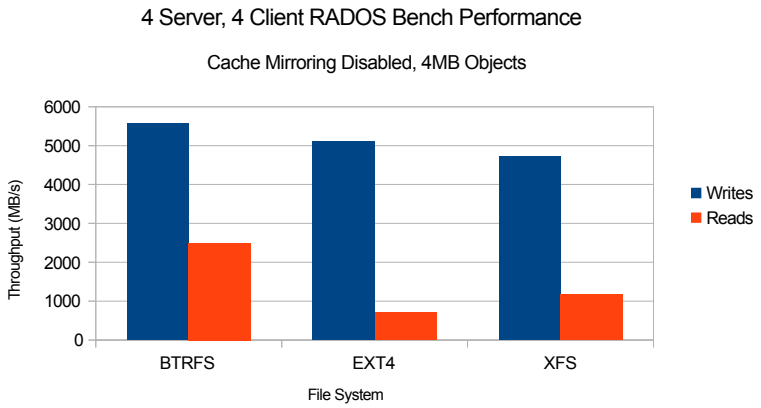

Figure 9: Evaluating RADOS bench after disabling cache mirroring

of doing. Unfortunately, read performance did not scale as well.

8.2 Disable TCP autotuning

During these tests, a trend that previously had been seen became more apparent. During reads, there were periods of high performance followed by periods of low performance or outright stalls that could last for up to 20 seconds at a time. After several hours of diagnostics, Inktank observed that outstanding operations on the clients were not being shown as outstanding on the OSDs. This appeared to be very similar to a problem Jim Schutt at Sandia National labs encountered with TCP autotuning in the Linux kernel.[4] TCP auto tuning enables TCP window scaling by default and automatically adjusts the TCP receive window for each connection based on link conditions such as bandwidth delay product. We have observed this will make a notable improvement on Ceph read performance, as the results shown in Figure 10.

Luckily, the fix was fairly straight forward by issuing the following command on all nodes:

```
echo 0 | sudo tee /proc/sys/net/ipv4/tcp_moderate_rcvbuf
```

Recent versions of Ceph work around this issue by manually controlling the TCP buffer size. The testing at ORNL directly influenced and motivated the creation of this feature!

8.3 Repeating RADOS Scaling Test

We now repeated the previous RADOS scaling tests with these improvements in place. The first test was done on a single node with RADOS Bench to see how close the underlying object store could get to the

[4] http://marc.info/?l=ceph-devel&m=133009796706284&w=2

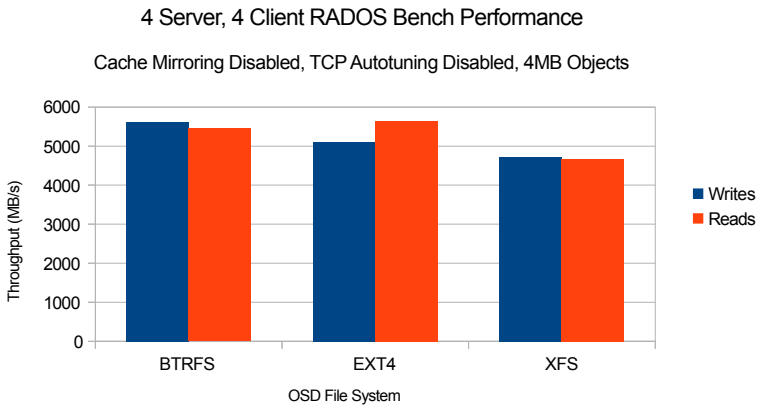

Figure 10: Evaluating RADOS bench after TCP auto tuning disabled

node hardware limitations as the number of OSDs/LUNs used on the node increased. Note all the tests performed were against XFS-formatted storage.

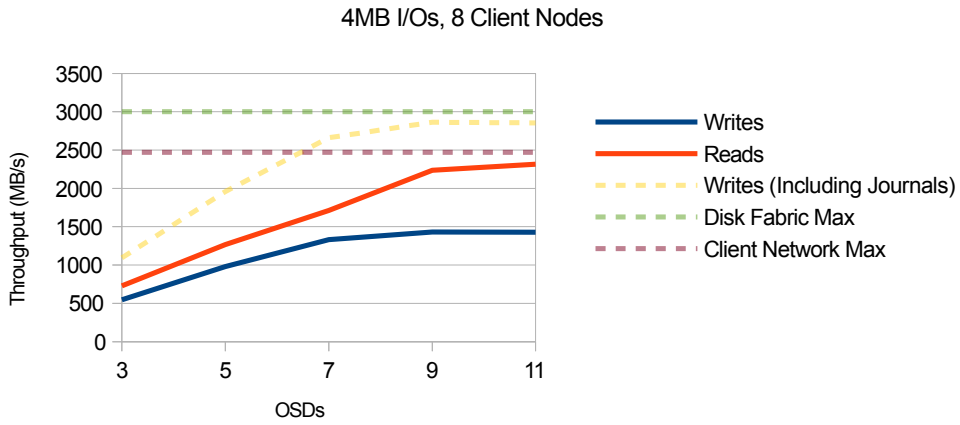

Figure 11: RADOS Bench Scaling on # of OSD, Ceph 0.64, 4 MB I/O, 8 Client Nodes

In the single server case as shown in Figure 11, "Writes (including Journals)" refers to how much data is actually being written out the DDN chasis, and blue line is how much data the clients are writing. We observe that performance gets very close to the hardware limits at roughly 9 OSDs per server and then mostly levels out.

We also repeated tests looking at RADOS Bench performance as the number of OSD server nodes increases from one to four. The results are summarized in Figure 12. As the number of nodes increases, performance scales nearly linearly for both reads and writes.

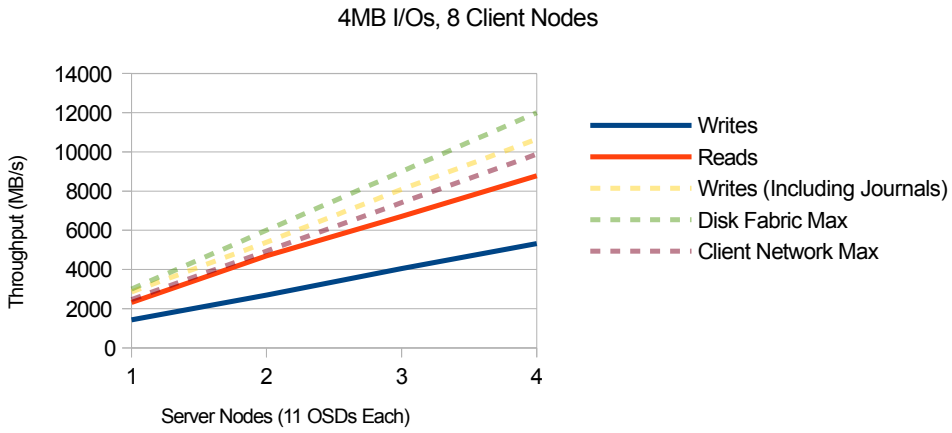

Figure 12: RADOS Bench Scaling on number of servers, Ceph 0.64, 4 MB I/O, 8 client nodes

9 Improving Ceph File System Performance

The initial stability issues mentioned in Section 7 are fixed by migrating from Ceph version 0.48/0.55 to 0.64, the latest stable version at the time of writing this report. Upgrading to the latest stable Ceph release allowed us to run a full IOR parameter sweep for the first time since we started evaluating the performance and scalability of the Ceph file system. This is another sign of how much Ceph development is currently in flux.

Another fix introduced by Ceph version 0.64 was in pool creation. The default data pool used by previous Ceph version were set to 2x replication by mistake. This potentially halved the write performance. With version 0.64 we explicitly set the replication level to 1, which is the preferred value for a HPC environment like ours running on high-end and reliable storage backend hardware (e.g. DDN SFA10K).

Even with these two changes in place, less-than-ideal write performance and very poor read performances were observed during our tests. We also observed that by increasing the number of IOR processes per client node, the read performance degraded even further indicating some kind of contention either on the clients or on the OSD servers.

9.1 Disabling Client CRC32

At this point, we were able to both make more client nodes available for Ceph file system-level testing and also install a profiling tool called `perf` that is extremely useful for profiling both kernel and user space codes. Profiling with `perf` showed high CPU utilization on test clients due to crc32c processing in the Ceph kernel client. crc32 checksums can be disabled by changing the CephFS mount options:

```
mount -t ceph 10.37.248.43:6789:/ /mnt/ceph -o name=admin,nocrc
```

With client CRC32 disabled, we repeated the IOR tests. New results are shown in in Figure 13.

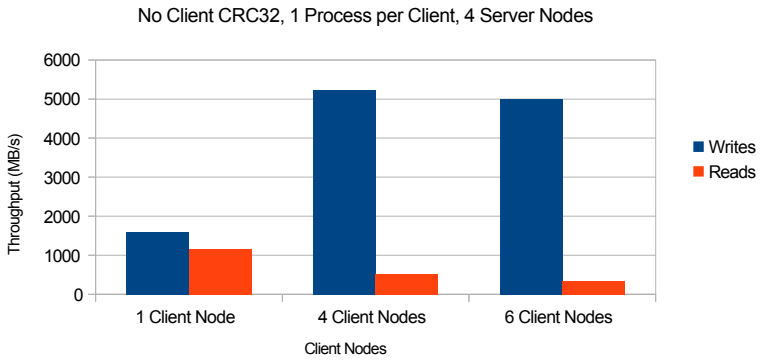

Figure 13: IOR test with disabling client-side CRC32

We observed that IOR write throughput increased dramatically and is now very close and comparable to the RADOS Bench performance. Read performance continued to be poor and continued to scale inversely with the increasing number of client processes. Please note that, since these tests were performed, Inktank has implemented SSE4-based CRC32 code for Intel CPUs. While any kernel based CRC32 processing should have already been using SSE4 instructions on Intel CPUs, this update will allow any user-land Ceph processes to process CRC32 checksums with significantly less overhead.

9.2 Improving IOR Read Performance

Deeper analysis with perf showed that there was heavy lock contention during parallel compaction in the Linux kernel memory manager. This behavior was first observed roughly in the kernel 3.5 time frame which was the kernel installed on our test systems.[5]

We upgraded our test systems with kernel version 3.9 and performed RADOS Bench test. The results were extremely positive and presented in Figure 14.

As can be seen, with the 3.9 kernel, while there was a slight improvement on write performance, read performance improved dramatically. In addition to the kernel change, Sage Weil from InkTank suggested increasing the amount of CephFS client kernel read-ahead cache size as:

```
mount -t ceph 10.37.248.43:6789:/ /mnt/ceph -o
   name=admin,nocrc,readdir_max_bytes=4104304,readdir_max_entries=8192
```

IOR results reflecting the read-ahead cache size change are presented in Figure 15.

By installing a newer kernel, increasing read-ahead cache size, and increasing the number of client IOR processes, we were able to achieve very satisfactory I/O performance.

[5]For more information, please refer to http://lwn.net/Articles/517082/ and https://patchwork.kernel.org/patch/1338691/.

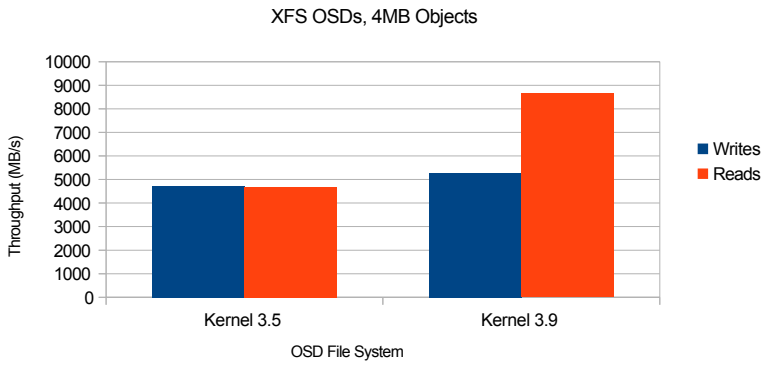

Figure 14: RADOS bench: Linux kernel version 3.5 vs. 3.9

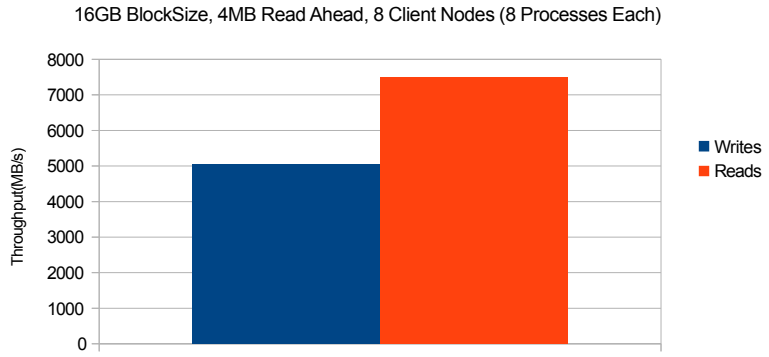

Figure 15: CephFS performance with kernel changes to 3.9, IOR with 4 MB transfer size

9.3 Repeating the IOR Scaling Test

As before, we ran IOR scaling tests with two cases: transfer size 4 KB and 4 MB. These results are illustrated in Figure 16. As expected, we saw saw improved read and write performance. These new read and write performance are in line with observed RADOS bench performance.

Throughout our IOR testing, we observed that the average write throughput is lower than the maximum. This behavior was observed during other tests as well, indicating that we may have periods of time where write throughput is temporarily degrading. Despite these issues, performance generally seems to be improving with respect to increasing number of clients. Writes seem to be topping out at around 5.2 GB/s (which is roughly what we would expect). Reads seem to be topping out anywhere from 5.6-7 GB/s, however it is unclear if read performance would continue scaling with more clients and get closer to 8 GB/s we observed with RADOS Bench.

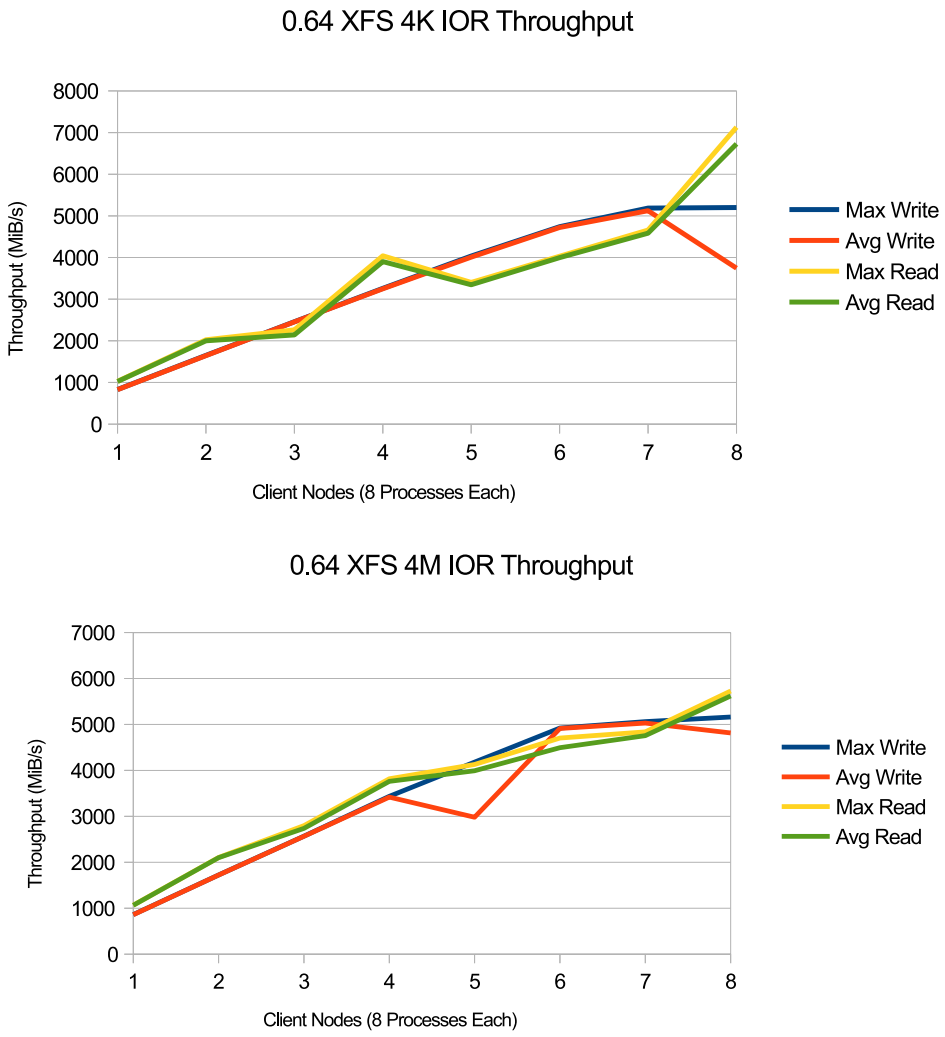

Figure 16: IOR Scaling Test: 4 KB and 4 MB transfer size

10 Metadata Performance

In our particular setup, we only had one metadata server (MDS) configured. Therefore, this is not a scalability test on the performance of Ceph clustered MDS, which would have been very interesting. Instead, we focus on a single MDS performance and exercise it with up to 8 clients to observe the single MDS performance scaling. We used mdtest as our benchmark tool. Mdtest parameters used for this test are:

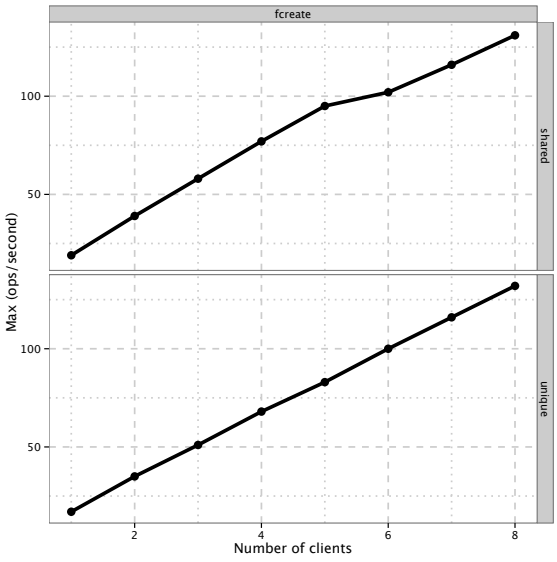

Figure 17: File creation vs. number of clients

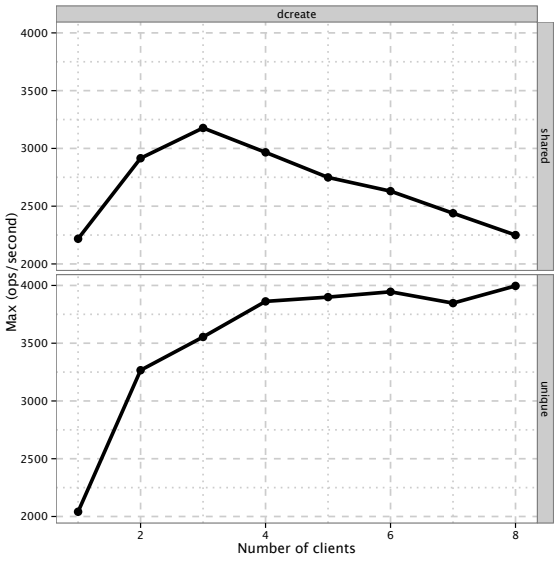

Figure 18: Directory creation vs. number of clients

- `-w 1048576 -y`: for each created file, we write 1 MB data and perform a sync operation. This is a more realistic use case scenario than just open, close and removal sequence of metadata operations.
- `-n 1000`: Per client file *and* directory workload. For eight clients, the total number of files and directories in the workload is 8,000. Since we did not specify either `-D` or `-F`, this is a mixture of both.
- `-d /mnt/cephfs/tmp`: we do specify a directory, but unlike under Lustre file system, where you can have single client multiple mounts (for increasing workload per client), here we just give the test an explicit home.
- `-u`: without this option, we are exercising shared directory; with this option, we are exercising unique directories.

Each test iterates five times and we are presenting the maximum out of all five iterations. We summarize the results as follows:

- With either shared or unique directory (`-u`), stat operations for directories and files exhibit strong linear scaling. Same strong linear scaling is also observed for file read operations.
- While the other operations seems unaffected or flat by the number of clients, it is not so if we zoom

in, see 17 and 18: as number of clients increases, we observed the contention for shared directory. The performance degradation amount to 50% or more.

- Though the same saturation (or degradation) trend was not observed for file creation operation, it is likely due to lack of workload stress on MDS.

The results also show that file creation rate is significantly lower than directory creation rate. We stipulated two contributing factors: one is the file creation is associated with 1 MB data write and followed by a fsync operation, which is a rather heavy weight operation compare to directory creation. Another factor is that we obtained above results from early version of Ceph without all the tunings and improvement applied. Therefore, we repeated the same test on the latest CephFS 0.64. The result is illustrated in Figure 19. As expected, we observed a significant improvement on the file creation rate. We also note that as number of clients increase, the file creation rate decreases rapidly, which suggested some form of lock contention. This presents some interesting issues to be investigated further.

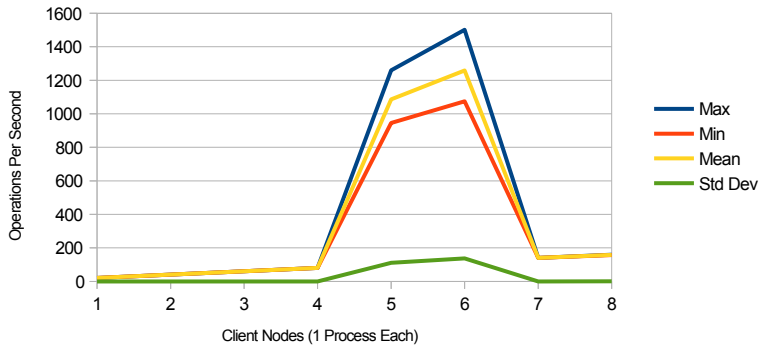

Figure 19: mdtest of file creation on Ceph 0.64

11 Observations and Conclusions

Below are our preliminary observations from mostly performance perspective:

- Ceph is built on the assumption that the underlying hardware components are unreliable, with little or no redundancy and failure detection capability. This assumption is not valid for high-end HPC centers like ours. We have disabled replication for pools, we haven't measured and quantified processing overhead and we do not know yet if this would be significant.
- Ceph performs **metadata + data** journaling, which is fine for host systems that has locally attached disk. However, this presents a problem in DDN SFA10K-like hardware, where the backend LUNs

are exposed as block devices through SCSI Request Protocol (SRP) over IB. The journaling write requires twice the bandwidth compare to Lustre-like metadata-only journaling mechanism. For Ceph to be viable in this facility, journaling operations will need to further design and engineering.

- We observed consistent results and linear scalability at the RADOS level. However, we did not observe the same at the file system level. We have experienced large performance swings during different runs, very low read performance when transfer size is small, and I/O errors tend to happen when system is under stress (more clients and large transfer sizes). These are not particularly reproducible results, but it suggests that there are opportunities for code improvement and maturation.

Appendix A - CephFS Final Mount Command

```
mount -t ceph 10.37.248.43:6789:/ /mnt/ceph -o
      name=admin,nocrc,readdir_max_bytes=4104304,readdir_max_entries=8192
```

Appendix B - OSD File System Options

```
btrfs mkfs options: -l 16k -n 16k
btrfs mount options: -o noatime

ext4 mkfs options: <none>
ext4 mount options: -o noatime, user_xattr

xfs mkfs options: -f -i size=2048
xfs mount options: -o inode64,noatime

Note: since this testing was preformed, two additional XFS options have been
shown to improve performance on some system:

Additional xfs mkfs option: -n size=64k
Additional XFS mount option: -o logbsize=256k
```

Appendix C - CRUSH Map

This crush map is just an example of full OSD mapping. As we create and re-create different testbed setups with different OSD configurations, the map will change accordingly.

```
# begin crush map

# devices
device 0 osd.0
device 1 osd.1
device 2 osd.2
device 3 osd.3
device 4 osd.4
device 5 osd.5
device 6 osd.6
device 7 osd.7
device 8 osd.8
device 9 osd.9
device 10 osd.10
device 11 osd.11
device 12 osd.12
device 13 osd.13
device 14 osd.14
device 15 osd.15
device 16 osd.16
device 17 osd.17
device 18 osd.18
device 19 osd.19
device 20 osd.20
device 21 osd.21
device 22 osd.22
device 23 osd.23
device 24 osd.24
device 25 osd.25
device 26 osd.26
device 27 osd.27
device 28 osd.28
device 29 osd.29
device 30 osd.30
device 31 osd.31
device 32 osd.32
```

```
device 33 osd.33
device 34 osd.34
device 35 osd.35
device 36 osd.36
device 37 osd.37
device 38 osd.38
device 39 osd.39
device 40 osd.40
device 41 osd.41
device 42 osd.42
device 43 osd.43

# types
type 0 osd
type 1 host
type 2 rack
type 3 row
type 4 room
type 5 datacenter
type 6 root

# buckets
host tick-oss1 {
        id -2                # do not change unnecessarily
        # weight 11.000
        alg straw
        hash 0        # rjenkins1
        item osd.0 weight 1.000
        item osd.1 weight 1.000
        item osd.10 weight 1.000
        item osd.2 weight 1.000
        item osd.3 weight 1.000
        item osd.4 weight 1.000
        item osd.5 weight 1.000
        item osd.6 weight 1.000
        item osd.7 weight 1.000
        item osd.8 weight 1.000
        item osd.9 weight 1.000
}
host tick-oss2 {
```

```
        id -4               # do not change unnecessarily
        # weight 11.000
        alg straw
        hash 0        # rjenkins1
        item osd.11 weight 1.000
        item osd.12 weight 1.000
        item osd.13 weight 1.000
        item osd.14 weight 1.000
        item osd.15 weight 1.000
        item osd.16 weight 1.000
        item osd.17 weight 1.000
        item osd.18 weight 1.000
        item osd.19 weight 1.000
        item osd.20 weight 1.000
        item osd.21 weight 1.000
}
host tick-oss3 {
        id -5               # do not change unnecessarily
        # weight 11.000
        alg straw
        hash 0        # rjenkins1
        item osd.22 weight 1.000
        item osd.23 weight 1.000
        item osd.24 weight 1.000
        item osd.25 weight 1.000
        item osd.26 weight 1.000
        item osd.27 weight 1.000
        item osd.28 weight 1.000
        item osd.29 weight 1.000
        item osd.30 weight 1.000
        item osd.31 weight 1.000
        item osd.32 weight 1.000
}
host tick-oss4 {
        id -6               # do not change unnecessarily
        # weight 11.000
        alg straw
        hash 0        # rjenkins1
        item osd.33 weight 1.000
        item osd.34 weight 1.000
```

```
        item osd.35 weight 1.000
        item osd.36 weight 1.000
        item osd.37 weight 1.000
        item osd.38 weight 1.000
        item osd.39 weight 1.000
        item osd.40 weight 1.000
        item osd.41 weight 1.000
        item osd.42 weight 1.000
        item osd.43 weight 1.000
}
rack unknownrack {
        id -3                # do not change unnecessarily
        # weight 44.000
        alg straw
        hash 0        # rjenkins1
        item tick-oss1 weight 11.000
        item tick-oss2 weight 11.000
        item tick-oss3 weight 11.000
        item tick-oss4 weight 11.000
}
root default {
        id -1                # do not change unnecessarily
        # weight 44.000
        alg straw
        hash 0        # rjenkins1
        item unknownrack weight 44.000
}

# rules
rule data {
        ruleset 0
        type replicated
        min_size 1
        max_size 10
        step take default
        step chooseleaf firstn 0 type host
        step emit
}
rule metadata {
        ruleset 1
```

```
        type replicated
        min_size 1
        max_size 10
        step take default
        step chooseleaf firstn 0 type host
        step emit
}
rule rbd {
        ruleset 2
        type replicated
        min_size 1
        max_size 10
        step take default
        step chooseleaf firstn 0 type host
        step emit
}

# end crush map
```

Appendix D - Final ceph.conf

```
[global]
        # Disable authentication (for testing)
        auth service required = none
        auth cluster required = none
        auth client required = none

        # Disable syslog logging
        log to syslog = false

        # use leveldb for certain xattr attributes.  Needed for EXT4.
        filestore xattr use omap = true

        # Use aio for the journal.  Enabled by default in recent versions of Ceph.
        journal aio = true

        # Disable crc checksums for the messenger.  Small Gain, may not be
        # necessary with SSE4 CRC32.
        ms nocrc = true

```

```
        # set the tcp rcvbuf size.  Workaround for Kernel TCP autotuning issues.
        ms tcp rcvbuf = 262144
        # Implemented to deal with thread timeouts.  May no longer be necessary.
        osd_op_thread_timeout = 30
        filestore_op_thread_timeout = 600

        # Use the Infiniband network
        public network = 10.37.0.0/16
        cluster network = 10.37.0.0/16

        # export logs to a specific home directory.
        log file = /chexport/users/nhm/ceph/$name.log

[mon]
        mon data = /tmp/mon.$id

[mon.a]
        mon addr = 10.37.248.43:6789
        host = spoon41

[mds.a]
        host = spoon41

[osd]
        osd journal size = 10240

[osd.0]
        host = tick-oss1
        osd data = /tmp/mnt/osd-device-0-data
        osd journal = /dev/mapper/tick-oss1-sata-l11

```

... Abbreviated for saving space

Appendix E - Tuning Parameters

For parameter sweep, here are the settings we iterated through. Some of these are "compound" settings where we increased or decreased multiple things in the ceph.conf file at the same time to reduce the number of permutations that would need to be tested.

```
Setting: big bytes (compound setting)
     # Increase various queue byte limits over the defaults
     filestore_queue_max_bytes: 1048576000
     filestore_queue_committing_max_bytes: 1048576000
     journal_max_write_bytes: 1048576000
     journal_queue_max_bytes: 1048576000
     ms_dispatch_throttle_bytes: 1048576000
     objecter_infilght_op_bytes: 1048576000


Setting: big ops (compound setting)
     # Increase various queue op limits over the defaults
     filestore_queue_max_ops: 5000
     filestore_queue_committing_max_ops: 5000
     journal_max_write_entries: 1000
     journal_queue_max_ops: 5000
     objecter_inflight_ops: 8192


Setting: debugging off (compound setting)
     # disable all debugging
     debug_lockdep: ”0/0”
     debug_context: ”0/0”
     debug_crush: ”0/0”
     debug_mds: ”0/0”
     debug_mds_balancer: ”0/0”
     debug_mds_locker: ”0/0”
     debug_mds_log: ”0/0”
     debug_mds_log_expire: ”0/0”
     debug_mds_migrator: ”0/0”
     debug_buffer: ”0/0”
     debug_timer: ”0/0”
     debug_filer: ”0/0”
     debug_objecter: ”0/0”
     debug_rados: ”0/0”
     debug_rbd: ”0/0”
     debug_journaler: ”0/0”
     debug_objectcacher: ”0/0”
     debug_client: ”0/0”
```

```
     debug_osd: ”0/0”
     debug_optracker: ”0/0”
     debug_objclass: ”0/0”
     debug_filestore: ”0/0”
     debug_journal: ”0/0”
     debug_ms: ”0/0”
     debug_mon: ”0/0”
     debug_monc: ”0/0”
     debug_paxos: ”0/0”
     debug_tp: ”0/0”
     debug_auth: ”0/0”
     debug_finisher: ”0/0”
     debug_heartbeatmap: ”0/0”
     debug_perfcounter: ”0/0”
     debug_rgw: ”0/0”
     debug_hadoop: ”0/0”
     debug_asok: ”0/0”
     debug_throttle: ”0/0”


Setting: default
     # Dummy setting to just use the ”base” config.

Setting: filestore_op_threads
     # Increase the number of filestore op threads (used for
reading/writing data)

Setting: flush_false (compound setting)
     # Disable the filestore flusher.  (Flushes happen less often but
are bigger)
     filestore_flush_min: 0
     filestore_flusher: ”false”

Setting: flush_true (compound setting)
     # Enable the filestore flusher (Flushes happen more often, but are
bigger)
     filestore_flush_min: 0
     filestore_flusher: ”true”

Setting: journal_aio_true
```

```
     # Enable asynchronous IO for journal writes

Setting: ms_nocrc_true
     # Disable CRC checks in the messenger (IE for network transfers,
but not for filestore)

Setting: osd_disk_threads
     # Number of threads used for background processes like scrubbing
and snap trimming.

Setting: osd_op_threads
     # Number of threads to use for OSD Daemon operations

Setting: small bytes (compound setting)
     # Decrease various queue byte limits vs the defaults
     filestore_queue_max_bytes: 10485760
     filestore_queue_committing_max_bytes: 10485760
     journal_max_write_bytes: 10485760
     journal_queue_max_bytes: 10485760
     ms_dispatch_throttle_bytes: 10485760
     objecter_infilght_op_bytes: 10485760

Setting: small_ops (compound setting)
     # Decrease various queue op limits vs the defaults
     filestore_queue_max_ops: 50
     filestore_queue_committing_max_ops: 50
     journal_max_write_entries: 10
     journal_queue_max_ops: 50
     objecter_inflight_ops: 128
```