# Benchmarking Ceph for Real World Scenarios

David Byte
Sr. Technical Strategist
SUSE

Matthew Curley
Sr. Technologist
HPE

# Agenda

**Problem**

**Use cases and configurations**

- **Object with & Without Journals**
- **Block with & without Journals**
- **File**

**Benchmarking methodologies**

**OS & Ceph Tuning**

# Why Benchmark at all?

To understand the ability of the cluster to meet your performance requirements

To establish a baseline performance that allows for tuning improvement measurements

Provides a baseline for future component testing for inclusion into the cluster and understanding how it may affect the overall cluster performance

# The Problem – Lack of Clarity

**Most storage requirements are expressed in nebulous terms that likely don't apply well to the use case being explored**

- **IOPS**
- **GB/s**

**Should be expressed in**

- **Protocol type with specifics if known**
  - Block, File, or Object
- **IO size**
  - 64k, 1MB, etc
- **Read/Write Mix with type of IO**
  - 60% Sequential Write with 40% random reads
  - Include the throughput requirement

# Protocols & Use Cases

# OBJECT

RADOS Native

S3

Swift

NFS to S3

Useful for:

- Backup
- Cloud Storage
- Large Data store for applications

# OBJECT – Characteristics

WAN friendly

High latency tolerant

Cloud Native Apps

Usually MB and larger size

Scales well with large number of users

# OBJECT – When to use journals

**There are occasions that journals make sense in object scenarios today**

- **Smaller clusters that may receive high bursts of write traffic**
  - Data Center Backups
  - Smaller Service Providers
- **Use cases where there may be a high number of small objects written**
- **Rebuild Requirements – Journals reduce time for the cluster to fully rebalance after an event**
- **Burst Ingest of large objects. Bursty writes of large objects can tie up a cluster without journals much easier**

# BLOCK

## RBD

## iSCSI

## Use Cases:

- Virtual Machine Storage
- D2D Backups
- Bulk storage location
- Warm Archives

# File

**CephFS is a Linux native, distributed filesystem**

- Will eventually support sharding and scaling of MDS nodes

**Today, SUSE Recommends the following usage scenarios**

- Application Home

# Should I Use Journals?

**What exactly are the journals?**

- Ceph OSDs use a journal for two reasons: speed and consistency. The journal enables the Ceph OSD Daemon to commit small writes quickly and guarantee atomic compound operations.

**Journals are usually recommended for Block and File use cases**

**There are a few cases where they are not needed**

- All Flash
- Where responsiveness and throughput are not a concern

**You don’t need journals when trying to gain read performance, no effect there.**

# Benchmarking

# Benchmarking the right thing

## Understand your needs

- Do you care more about bandwidth, latency or high operations per second?

## Understand the workload

- Is it sequential or random?
- Read, Write, or Mixed?
- Large or small I/O?
- Type of connectivity?

# Watch for the bottlenecks

**Bottlenecks in the wrong places can create a false result**

- **Resource Bound on the Testing Nodes?**
  - Network, RAM, CPU
- **Cluster Network Maxed Out?**
  - Uplinks maxed
  - testing nodes links maxed
  - switch cpu maxed
- **Old drivers?**

# Block & File

# Benchmarking Tools - Block & File

FIO - current and most commonly used

iometer - old and not well maintained

iozone - also old and not a lot of wide usage

Spec.org - industry standard audited benchmarks, specSFS is for network file systems. fee based

spc - another industry standard, used heavily by SAN providers, fee based

# Block - FIO

**FIO is used to benchmark block i/o and has a pluggable storage engine, meaning it works well with iSCSI, RBD, and CephFS with the ability to use an optimized storage engine.**

- Has a client/server mode for multi-host testing
- Included with SES

- Info found at: http://git.kernel.dk/?p=fio.git;a=summary

- sample command & common options
- fio --filename=/dev/rbd0 --direct=1 --sync=1 --rw=write --bs=1M --numjobs=16 --iodepth=16 --runtime=300 --time_based --group_reporting --name=bigtest

# FIO Setup

**Install**

- *zypper in fio*

**Single client**

- Use cli
- fio

**Multiple clients**

- one client (think console), multiple servers
- use job files
- fio --client=server --client=server

fio_job_file.fio

```
[writer]
ioengine=rbd
pool=test2x
rbdname=2x.lun
rw=write
bs=1M
size=10240M
direct=0
```

# FIO – How to read the output

## Tips

- **FIO is powerful – lots of information.  Start with summary data**
- **Watch early runs to sample performance, help adjust testing**

## Run Results

- **Breakdown information per job/workload**
  - Detailed latency info
  - Host CPU impact
  - Load on target storage
- **Summary on overall performance and storage behavior**

# FIO – Output example

**Before and during the run**

| | |
|---|---|
| **Summary information about the running test** | samplesmall: (g=0): rw=randwrite, bs=4K-4K/4K-4K/4K-4K, ioengine=libaio, iodepth=8<br>fio-2.1.10<br>Starting 1 process<br>samplesmall: Laying out IO file(s) (100 file(s) / 100MB) |
| **Current/final status of IO and run completion.** | Jobs: 1 (f=100): [w] [100.0% done] [0KB/1400KB/0KB /s] [0/350/0 iops] [eta 00m:00s] |

# FIO – Output example

## Detailed Breakout

| | |
|---|---|
| **Per Job IO workload** | samplesmall: (groupid=0, jobs=1): err= 0: pid=12451: Wed Oct 5 15:54:02 2016<br>write: io=84252KB, bw=1403.3KB/s, iops=350, runt= 60041msec |
| **Latency to submit & complete IO** | slat (usec): min=3, max=154, avg=12.15, stdev= 4.69<br>clat (msec): min=2, max=309, avg=22.80, stdev=21.14<br>lat (msec): min=2, max=309, avg=22.81, stdev=21.14 |
| **Latency histogram** | clat percentiles (msec):<br>\| 1.00th=[ 5], 5.00th=[ 7], 10.00th=[ 8], 20.00th=[ 10],<br>\| 30.00th=[ 12], 40.00th=[ 13], 50.00th=[ 16], 60.00th=[ 19],<br>\| 70.00th=[ 24], 80.00th=[ 32], 90.00th=[ 47], 95.00th=[ 63],<br>\| 99.00th=[ 111], 99.50th=[ 130], 99.90th=[ 184], 99.95th=[ 196],<br>\| 99.99th=[ 227] |
| **Bandwidth data & latency distribution** | bw (KB /s): min= 0, max= 1547, per=99.32%, avg=1393.47, stdev=168.47<br>lat (msec) : 4=0.63%, 10=22.43%, 20=39.57%, 50=28.72%, 100=7.28%<br>lat (msec) : 250=1.41%, 500=0.01% |

# FIO – Output example

## Detailed Breakout, Continued

| | |
|---|---|
| **System CPU %, context switches, page faults** | cpu : usr=0.19%, sys=0.84%, ctx=26119, majf=0, minf=31 |
| **Outstanding I/O statistics** | IO depths : 1=0.1%, 2=0.1%, 4=0.1%, 8=125.1%, 16=0.0%, 32=0.0%, >=64=0.0%<br>submit : 0=0.0%, 4=100.0%, 8=0.0%, 16=0.0%, 32=0.0%, 64=0.0%, >=64=0.0%<br>complete : 0=0.0%, 4=100.0%, 8=0.1%, 16=0.0%, 32=0.0%, 64=0.0%, >=64=0.0% |
| **IO Count** | issued : total=r=0/w=21056/d=0, short=r=0/w=0/d=0 |
| **FIO latency target stats** | latency : target=0, window=0, percentile=100.00%, depth=8 |

# FIO – Output example

## Run Results

| | |
|---|---|
| **Summary status for run** | Run status group 0 (all jobs):<br>WRITE: io=84252KB, aggrb=1403KB/s, minb=1403KB/s, maxb=1403KB/s, mint=60041msec, maxt=60041msec |
| **Linux target block device stats** | Disk stats (read/write):<br>dm-0: ios=0/26354, merge=0/0, ticks=0/602824, in_queue=602950, util=99.91%, aggrios=0/26367, aggrmerge=0/11, aggrticks=0/602309, aggrin_queue=602300, aggrutil=99.87%<br>sda: ios=0/26367, merge=0/11, ticks=0/602309, in_queue=602300, util=99.87% |

# Object

# Benchmarking Tools - Object

**Cosbench - COSBench - Cloud Object Storage Benchmark**

COSBench is a benchmarking tool to measure the performance of Cloud Object Storage services. Object storage is an emerging technology that is different from traditional file systems (e.g., NFS) or block device systems (e.g., iSCSI). Amazon S3 and Openstack* swift are well-known object storage solutions.

**https://github.com/intel-cloud/cosbench**

# Object - Cosbench

**Supports multiple object interfaces including S3 and Swift**

**Supports use from CLI or web GUI**

**Capable of building and executing jobs using multiple nodes with multiple workers per node**

**Can really hammer the resources available on a radosgw**

| PID | USER | PR | NI | VIRT | RES | SHR | S | %CPU | %MEM | TIME+ | COMMAND |
|---|---|---|---|---|---|---|---|---|---|---|---|
| 47705 | ceph | 20 | 0 | 16.955g | 258432 | 12808 | S | 743.23 | 0.078 | 42:04.55 | radosgw |

**And on the testing node**

| PID | USER | PR | NI | VIRT | RES | SHR | S | %CPU | %MEM | TIME+ | COMMAND |
|---|---|---|---|---|---|---|---|---|---|---|---|
| 39176 | root | 20 | 0 | 25.969g | 523028 | 15136 | S | 304.64 | 0.797 | 27:31.48 | java |

# Cosbench Setup

**Download from: https://github.com/intel-cloud/cosbench/releases or get my appliance on SUSEStudio.com https://susestudio.com/a/8Kp374/cosbench**

**If installing by hand, add java 1.8 and which to your install**

**make sure to chmod a+x *.sh in the directory**

**Job setup can be done via GUI or jumpstarted from templates in conf/ directory**

**conf/controller.conf**

```
[controller]
drivers = 2
log_level = INFO
log_file = log/system.log
archive_dir = archive

[driver1]
name = testnode1
url = http://127.0.0.1:18088/driver

[driver2]
name=testnode2
url=http://192.168.10.2:18088/driver
```

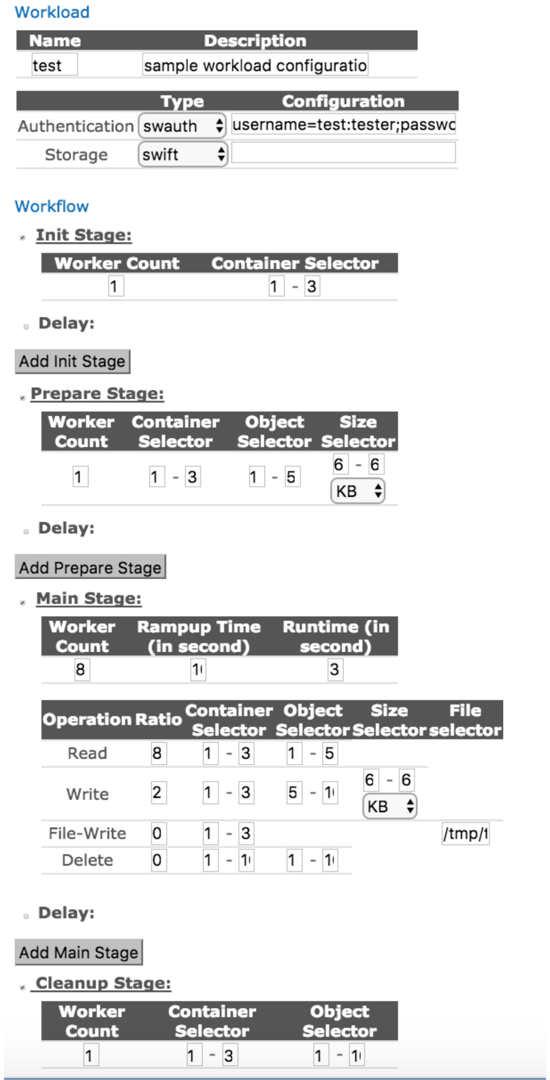
**conf/driver.conf**

```
[driver]
name=testnode1
url=http://127.0.0.1:18088/driver
```

# Cosbench Job Setup

## The GUI is the easy way to setup jobs.

## Define things like number of containers, number of objects, size of objects, number of workers, etc.



# Reading Cosbench Output

**General Report**

| Op-Type | Op-Count | Byte-Count | Avg-ResTime | Avg-ProcTime | Throughput | Bandwidth | Succ-Ratio |
|---|---|---|---|---|---|---|---|
| op1: init -write | 0 ops | 0 B | N/A | N/A | 0 op/s | 0 B/S | N/A |
| op1: prepare -write | 1.6 kops | 3.98 GB | 29.62 ms | 24.3 ms | 33.56 op/s | 83.37 MB/S | 100% |
| op1: read | 120.79 kops | 299.92 GB | 11.4 ms | 4.47 ms | 402.67 op/s | 999.81 MB/S | 100% |
| op2: write | 30.27 kops | 75.7 GB | 33.61 ms | 28.5 ms | 100.89 op/s | 252.33 MB/S | 100% |
| op1: cleanup -delete | 3.2 kops | 0 B | 7.72 ms | 7.72 ms | 129.23 op/s | 0 B/S | 100% |
| op1: dispose -delete | 0 ops | 0 B | N/A | N/A | 0 op/s | 0 B/S | N/A |

**ResTime (RT) Details**

| Op-Type | 60%-RT | 80%-RT | 90%-RT | 95%-RT | 99%-RT | 100%-RT |
|---|---|---|---|---|---|---|
| init-write | N/A | N/A | N/A | N/A | N/A | N/A |
| prepare-write | < 40 ms | < 40 ms | < 40 ms | < 40 ms | < 40 ms | < 1,970 ms |
| read | < 20 ms | < 20 ms | < 20 ms | < 20 ms | < 30 ms | < 360 ms |
| write | < 40 ms | < 40 ms | < 50 ms | < 50 ms | < 60 ms | < 400 ms |
| cleanup-delete | < 10 ms | < 10 ms | < 10 ms | < 10 ms | < 20 ms | < 30 ms |
| dispose-delete | N/A | N/A | N/A | N/A | N/A | N/A |

hide peformance details

# Reading Cosbench Output

## The section below gives information about the stages of the test from the config file.

### Stages

| Current Stage | Stages completed | Stages remaining | Start Time | End Time | Time Remaining |
|---|---|---|---|---|---|
| | | | | | |

| ID | Name | Works | Workers | Op-Info | State | Link |
|---|---|---|---|---|---|---|
| w2-s1-init | init | 1 wks | 1 wkrs | init | completed | view details |
| w2-s2-prepare | prepare | 1 wks | 1 wkrs | prepare | completed | view details |
| w2-s3-normal | normal | 1 wks | 8 wkrs | read, write | completed | view details |
| w2-s4-cleanup | cleanup | 1 wks | 1 wkrs | cleanup | completed | view details |
| w2-s5-dispose | dispose | 1 wks | 1 wkrs | dispose | completed | view details |

There are 5 stages in this workload.

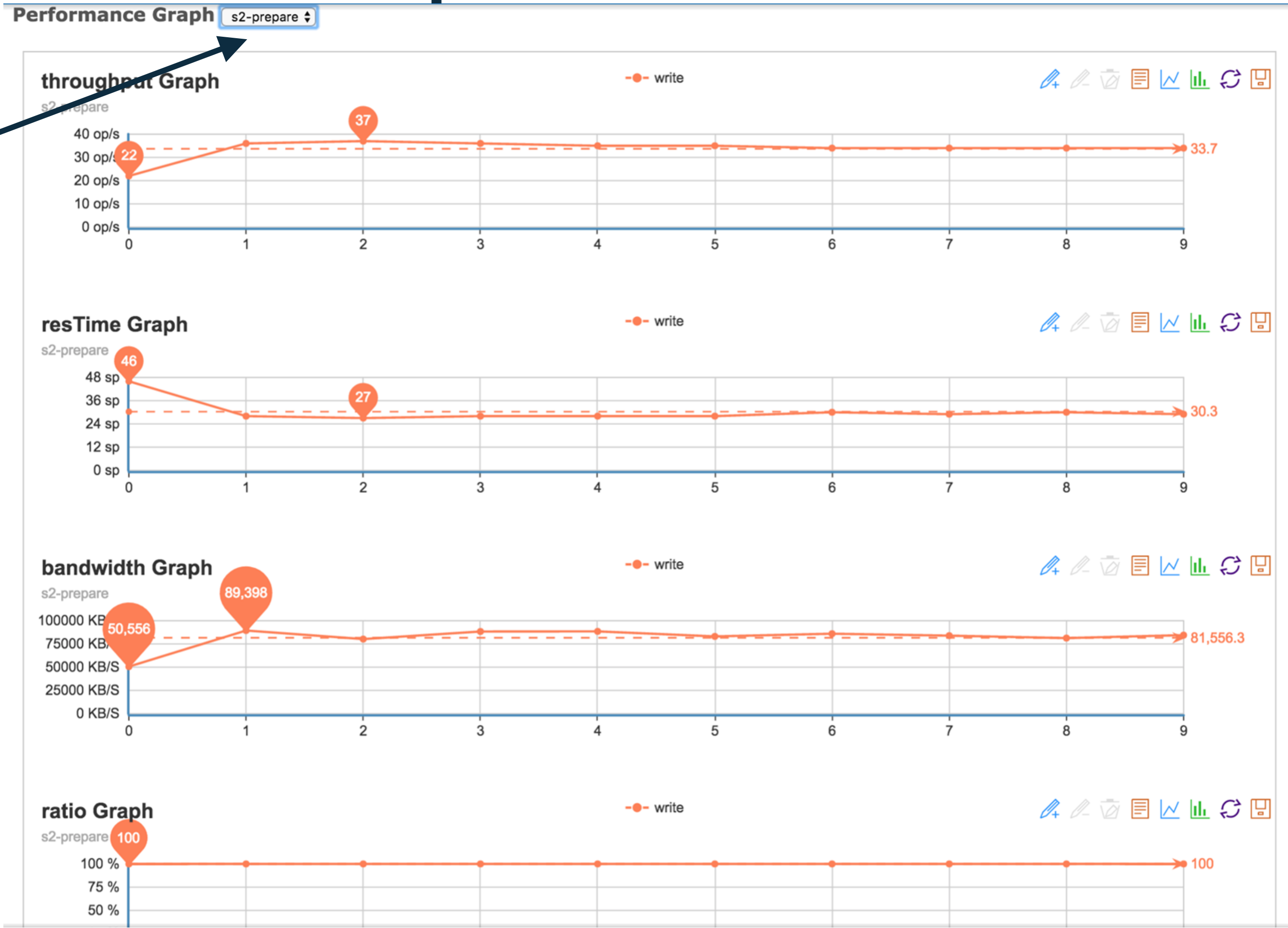
### Error Statistics

| Driver Url | Error Code | Occurrence Number |
|---|---|---|

hide error statistics details

# Reading Cosbench Output

**Note the stage**



# Reading Cosbench Output

**Highs and lows are identified by the bubbles**

# Summary

**Choose the benchmark(s) and data pattern(s) that best fit what you want to learn about the solution.**

- Benchmarking can help determine ‘how much’ a solution can do, but also help understand ‘sweet spots’ for SLA and cost.
- Ceph supports different I/O ingest, so important to cover each type

**Build from benchmark results**

- More complex testing starts with baseline expectations.
- Next steps: canned application workloads, canary/beta deployments

# Tuning

# Hardware Tuning

- Use SSD Journals
- Attach spinners to controllers with battery backed cache in write-back mode
- Set the firmware for performance bias
- Get a block diagram and make sure you aren’t overwhelming the bus

# OS Tuning

- For multi-processor systems, NUMA pinning of soft IRQs can improve CPU efficiency.  Map cores, PCIe Devices, OSD/journals
- With above, try distribute interrupts for controller(s) to match core count for socket
- Network jumbo frame settings can boost performance

# Ceph Tuning

## General

- **Tuning best done against application workloads**
- **Set placement groups counts appropriate for pools**
- **Disable OSD scrub only during performance evaluation**
- **Verify acceptable performance AND acceptable latency**
- **Tune and test at scale**
- **Stay ahead of degraded disks – lowest common denominator performance**
- **Consider whether you are comfortable with the RAID controller battery backed cache. If so, adjust the OSD mount parameters**

  - osd mount options xfs = nobarrier, rw, noatime, inode64, logbufs=8

# Ceph Tuning

## Block

- Multiple OSDs per device may improve performance, but not typically recommended for production
- Ceph Authentication and logging are valuable, but could disable for latency sensitive loads – understand the consequences.
- ‘osd op num shards’ and ‘osd op num threads per shard’ – bumping may improve some workloads on flash, cost more CPU
- VM/librbd use, configure RBD caching appropriate to workload

# Ceph Tuning

## Object

- Adjust ‘filestore merge threshold’ and ‘filestore split multiple’ settings to mitigate performance impact as data grows
- Test with a few variations of EC m & k values
- Use the isa erasure coding library for Intel CPUs
  - `erasure-code-plugin=isa` on the pool creation command line

# In Conclusion

**Ensure you are benchmarking what is really important**

**Use the right tools, the right way**

**If you perform baselines, save the job configuration details for proper future comparison**

**If you tune your config, keep a backup copy of the config file.**

We adapt. You succeed.