SDC 19

September 23-26, 2019
Santa Clara, CA

Object Storage Workload Tools

John Harrigan
Performance and Scale Engineering, Red Hat

Presentation Take-Aways

Know how to:

- Design the Workload
- Use the Object Storage Workload Tools
- Analyze the Results

Agenda

- Motivation and Testing Methodology
- Designing Workloads
- Workload Test Tools (Usage and Results)
- Demo
- Questions
- Backup Slides

Motivation to Develop

Need to incorporate workload-driven requirements into Red Hat Storage development, documentation, test, and release processes. Continuously test and validate those storage workloads going forward.

Object Storage Workload Testing

- Simulate customer production environments
- Scale-out workloads sampled from key customers
- Record client I/O and system resource utilization statistics

Testing Methodology

Comprehensive workload profile:

- Clusters pre-filled to simulate customer production environments
- Ceph RGW scale-out workload sampled from key customers
- Automated failure injection with in-house tooling
- Recording client I/O throughput and latency statistics
- Log Ceph radosgw system resource utilization (cpu, memory)
- Additional logging: fill rates; RGW garbage collection

Workload generated with COSBench
https://github.com/intel-cloud/cosbench

Designing Workloads

Workload Modeling

- Layout
  - Number of Buckets
  - Number of Objects (per Bucket)
- Object sizes
- Operation types/mixture
- Throughput per day
  - Number of objects accessed
  - Number of objects modified

Considerations

- Micro benchmarks vs. Production simulation
- Ensure workload generator is not a bottleneck

Designing Workloads

Workload Specifications

- Sizing for Capacity = number of objects
  - Test Conditions
    - Available capacity: cap (factor in replication type)
    - Percent cluster fill: %fill
    - object size: objsz
    - number of buckets/containers: numbuckets
  - numobj = ((cap * %fill) / objsz) / numbuckets
- Sizing for Performance = number of workers
  - Target average latency (QoS)
  - Perform sizing runs to determine cluster performance level
  - Adjust number of workers: numworkers

Micro-benchmarks

Object sizes = Fixed (constant) sizes

- 64K - Small size (Thumbnail images, small files etc.)
- 1M - Medium size (Images, text files etc.)
- 32M - Large size (Data Analytics, HD images, log files, backup etc.)

Test Selection (single operation type)

- Sequential: 100% Sequential Read, then 100% Sequential Write
- Random: 100% Random Read, then 100% Random Write

Typically single operation type and single object size
Short runtime duration in lab environment

Production Simulation ‘hybrid’

Based on 48 hour production monitoring

Object size Histogram

- 1K - 50%
- 64K, 8M, 64M - 15% each
- 1G - 5%

Operation types and mix

- 60% Read
- 10% List
- 16% Delete
- 14% Write

Multiple operation types and object sizes

Extended runtimes with operational events

Workload Generator Considerations

Scaling factors

- Network performance - Driver to Object Storage
- Number of Drivers
- Workload definition
  - Object sizes
  - Operation type
  - Number of workers/threads (per Driver)

Workload generator limits

- Driver processing overhead (*'mock' driver info follows)*

Workload Generator Limits

COSbench Scaling Measurements:

- Use ‘mock’ storage driver

```
<!-- Mock Auth / Storage -->
<auth type="mock" config="delay=0"/>
```

Observations:

- Throughput limiting operation type is Write
- Read, List and Delete deliver higher throughput, but don’t scale with number of workers
- Optimal ratio: 4 Drivers per Client node, each with 3 workers
- On larger configurations use load balancer (HAproxy) between Drivers and Storage

Scaling COSbench

Per Driver Resources (increases with number of workers)

- 1-2x CPU
- 1-2GB Memory RSS

Scaling Number of Drivers

- One Controller with one Driver
  ./conf/controller.conf ← one driver section (default)
  driver./start-all.sh
- One Controller with X Drivers
  Edit ./conf/controller.conf ← lists X driver sections
  ./start-driver.sh X
  ./start-controller.sh

Scaling COSbench

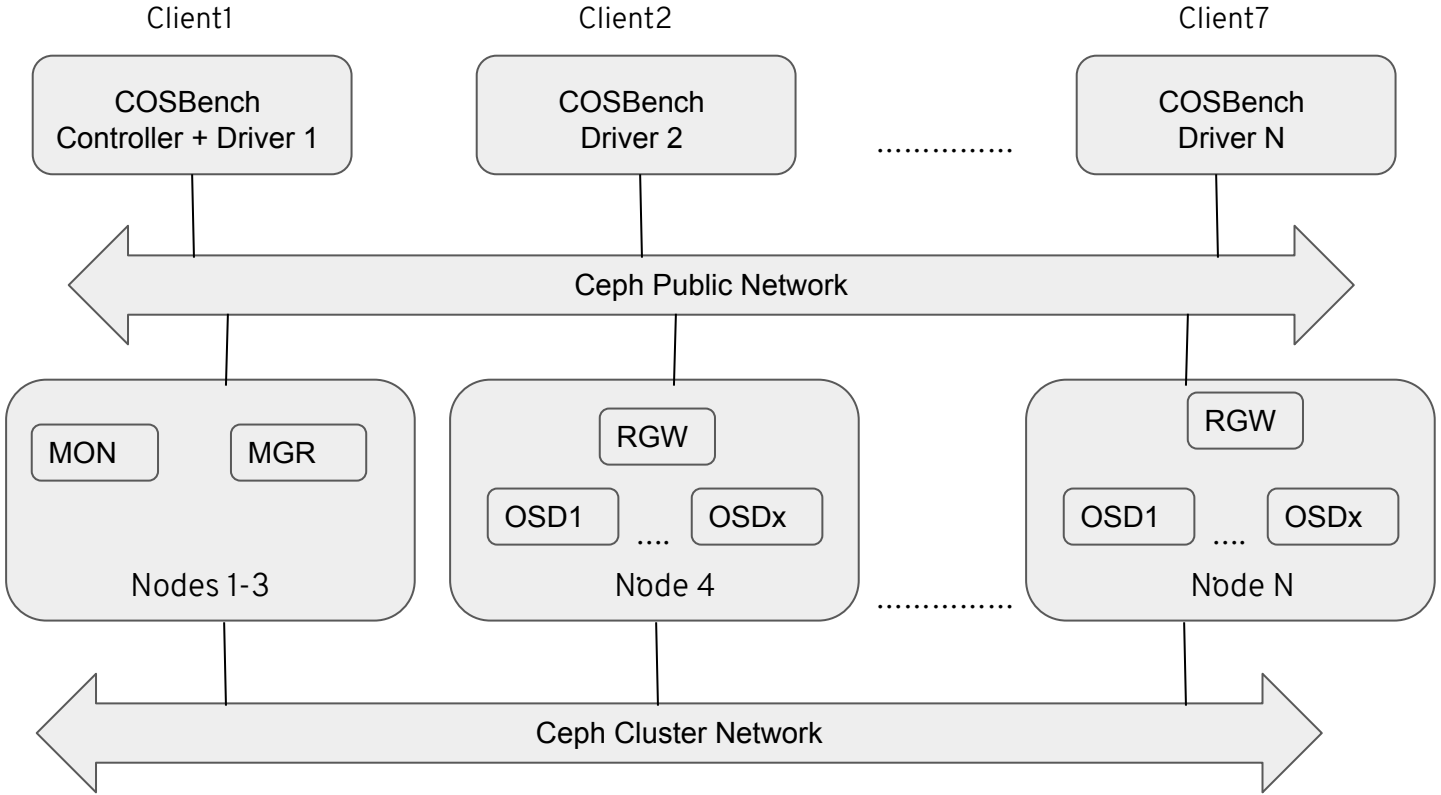
Example: two Drivers - ./conf/controller.conf

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

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

[driver2]
name = driver2
url = http://127.0.0.1:18188/driver
```

Object Storage Testbed Components



Object Storage Workload Testing Tools

- genXMLs - generates COSbench workload files
- RGWtest - Ceph RGW testing automation
- RGWOCP - deploys Ceph RGW & COSbench on kubernetes
- OSDfailure - injects failures and measures Client I/O impact

https://github.com/jharriga/

genXMLs - Overview

Purpose

- Generates workload files and text COSbench results

Automation Capabilities

- Generates workload files (genXMLs.sh)
- Produces COSbench ‘General Report’ (cbparser.py)

Repo URL

- https://github.com/jharriga/genXMLs

genXMLs - Usage

Procedure

- Install
  - git clone https://github.com/jharriga/genXMLs
- Configure
  - Edit genXMLs.sh
    - akey, skey, endpt ← S3 AUTH
    - testname, runtime
    - objSIZES, numCONT, numOBJ
- Generate workload files
  - ./genXMLs.sh

See next slide for workload file inventory

genXMLs - Workload Files

FILENAME	DESCRIPTION
fill.xml	cluster fill workload (creates buckets and objects) - first workload
empty.xml	empty cluster (removes objects and buckets) - last workload
seqops.xml	performs sequential reads and writes (two workstages)
randomops.xml	performs random reads and writes (two workstages)
mixedops.xml	performs mixture of read, list, write and delete operations

RGWtest - Overview

Purpose

- Automates Ceph RGW performance testing

Automation Capabilities

- Generates workload files
- Configures RGW for test runs
  - Creates (swift) user and pools
  - Inserts (swift) user credentials in workload files
- Executes and monitors workloads
  - logs system resource utilization
  - logs Ceph stats

https://github.com/jharriga/RGWtest

RGWtest - Usage

- Install
  - git clone https://github.com/jharriga/RGWtest
- Configure
  - Edit vars.shinc
- Run tests
  - writeXML.sh
  - resetRGW.sh
  - runIOworkload.sh <workload.xml>
- Review results
  - logfile (RGWtest/<RESULTSDIR>/<LOGFILE>)
  - client performance (COSbench controller)

RGWtest - Workload Files

FILENAME	DESCRIPTION
fillWorkload.xml	cluster fill workload (creates buckets and objects) - first workload
emptyWorkload.xml	empty cluster (removes objects and buckets) - last workload
ioWorkload.xml	User defined workload built from ‘XMLtemplates’ files

Execute a workload:

runIOworkload.sh <workload.xml>

Can also run workloads created with genXML.sh

RGWtest - Configuration (1 of 2)

Edit vars.shinc - Runtime Environment

Variable Name	Default Value	Definition
MONhostname	pcloud10	Ceph MON hostname/ip address
RGWhostname	pcloud08	Ceph radosgw hostname/ip address
cosPATH	/root/v0.4.2	Path to locally installed COSbench
rgwUSER	johndoe:swift	Object storage username
rgwURL	localhost:5000	Object storage endpoint
preparePTYPE	ec	Replication type (ec or 3-way)
pg_data, pg_index, pg	4096, 256, 256	determined by PGCALC (backup slides)

RGWtest- Configuration (2 of 2)

Edit vars.shinc - Workload definition

Variable Name	Default Value	Definition
objSIZES	histogram	1KB/50%, 8KB/15%, 65MB/15%, 1GB/5%
numCONT	5	number of containers (or buckets)
numOBJ	232000	number of objects per container
numOBJmax	numOBJ	useful for aging runs - (numOBJ * 2)
runtime_sec	3600	1 hour runtime (in seconds)
fillWORKERS	40	number of workers/threads - fillWorkload.xml
runtestWORKERS	40	number of workers/threads - ioWorkload.xml

RGWtest - Workload Specifications

Define new workload specifications (advanced)

- RGWtest/Templates contains the template workload files
- Values from vars.shinc are inserted by ‘writeXML.sh’

EXAMPLE - use alternate template

Edit vars.sh → RUNTESTtemplate=”TMPL_deleteWrite.tmpl”

writeXML.sh
runIOworkload ioWorkload.xml

Users can modify existing or create new templates

RGWtest
Review Results

RGWtest - logfile

Default location ← RESULTSDIR="./RESULTS"

- LOGFILE="${RESULTSDIR}/${PROGNAME}_${ts}.log"

Collected statistics ($pollinterval=”1m”)

- ‘ceph df’ capacity - RAW and default.rgw.buckets.data
- System resource utilization
  - OSD/RGW system load average
  - radosgw process and memory stats
  - ceph-osd process and memory stats
- Ceph RGW garbage collection
- Ceph RGW resharding activity

All log entries are timestamped

RGWtest - Logfile excerpt

Logfile excerpt - Fill Cluster 30%

Start

GLOBAL:

SIZE	AVAIL	RAW USED	%RAW USED
524TiB	523TiB	308GiB	0.06

POOLS:

NAME	ID	USED	%USED	MAX AVAIL	OBJECTS
default.rgw.buckets.data	10	0B	0	332TiB	0

End

GLOBAL:

SIZE	AVAIL	RAW USED	%RAW USED
524TiB	379TiB	145TiB	27.62

POOLS:

NAME	ID	USED	%USED	MAX AVAIL	OBJECTS
default.rgw.buckets.data	10	95.9TiB	32.46	199TiB	26573298

RGWtest - Review Results

Test Description	Conditions	Avg Thruput	Avg Latency	99% Latency
fillCluster 30%	3.1 filestore	147 op/s	466 ms	8640 ms
fillCluster 30%	3.2 bluestore	164 op/s	416 ms	8260 ms
PERCENT CHANGE	N/A	+11%	-11%	-4%

Logfile Statistics:

- Load average: 15 min avg - filestore=23.2 ; bluestore=8.7
- CPU usage (PCPU) and memory usage (VSZ and RSS)
  - Ceph-osd
  - Radosgw
- Workload runtime: start and end timestamp

RGWtest - Review Results

Test Description	Conditions	Avg Thruput	Avg Latency	99% Latency
hybridSS initial (read stats, 1hr)	3.1 filestore	85 op/s	644 ms	10080 ms
hybridSS initial (read stats, 1hr)	3.2 bluestore	142 op/s	358 ms	6540 ms
PERCENT CHANGE	N/A	+67%	-45%	-35%

Logfile Statistics:

- Load average: 15 min avg - filestore=27.2 ; bluestore=7.3
- CPU usage (PCPU) and memory usage (VSZ and RSS)
  - Ceph-osd
  - Radosgw
- Workload runtime: start and end timestamp

Tools Demo

Presentation Take Away's

Know how to:

- Design the Workload
- Use the Object Storage Workload Tools
- Analyze the Results

Questions/Discussion

BACKUP SLIDES

RGWtest - Procedure

- Calculate pg_num values for RGW pools
  - https://access.redhat.com/labsinfo/cephpgc (downstream - supports EC)
  - https://ceph.io/pgcalc/ (upstream)
- Edit RGWtest/vars.shinc
- Create pools and user (resetRGW.sh)
- Write workload files (writeXML.sh)
- Workload run sequence (runIOworkload.sh)
  1. Fill cluster to a predetermined % RAW USAGE
  2. Run hybridSS workload as initial measurement run
  3. Run hybrid2x workload to age the cluster
  4. Run hybridSS workload as aged measurement run
  5. Empty cluster
- Review Results

RGWtest - Test Configuration

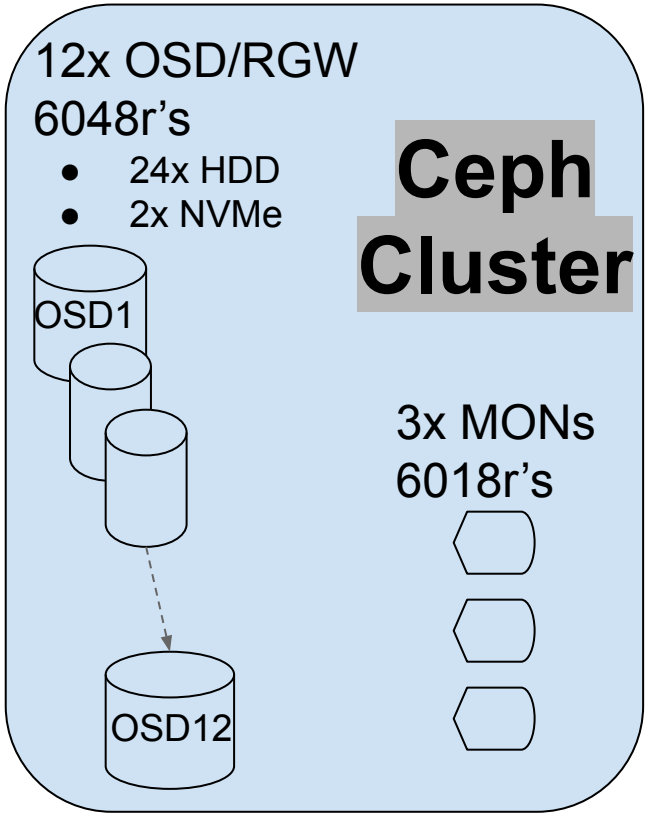
Hardware

- 12x OSD nodes (312 OSDs with 500TB RAW)
- 3x MON nodes (one serving as MGR node)
- 17x Client nodes (COSbench drivers)
- 1x Admin node (RGWtest, COSbench controller and ceph-ansible)

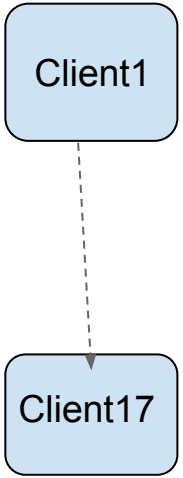
Software

- RHEL 7.6
- RHCS 3.2
  - bluestore w/osd_scenario=noncollocated
  - default WAL and DB sizes
- Ceph pool configuration
  - default.rgw.buckets.data: EC 4+2; pg_data=4096
  - All other Ceph pools: 3-way replication; pg_index=256, pg=256

Test Configuration



17x Clients
Dell r620's
- COSbench Drivers
- HAproxy



1x Admin
Dell r620's
- RGWtest
- COSbench Controller

RGWtest - Variable Settings

Object count calculations (sizing capacity)

- Actual available (factoring replication type)
  - 'ceph df' output , the MAX AVAIL for *default.rgw.buckets.data 332TiB*
- RGW Object size distribution
  objSIZES="h(1|1|50,64|64|15,8192|8192|15,65536|65536|15,1048576|1048576|5)KB"
- 62MB Mean Objsz: (1*.5)+(64*.15)+(8192*.15)+(65536*.15)+(1048576*.05)
- numobj = ((cap * %fill) / objsz) / numbuckets
  - ((332TiB * 0.3) / 62MB) / 5 = ~400k

RGWtest variable settings (vars.shinc)

- preparePTYPE=ec, k=4, m=2 ← EC 4+2 replication
- pg_data=4096, pg_index=256, pg=256 ← https://access.redhat.com/labs/cephpgc/
- numCONT=5, numOBJ=400000 ← 2M Objects total
- fillWORKERS=68 ← 17 driver nodes (four per driver node)
- runtestWORKERS=68 ← 17 driver nodes (four per driver node)

Ceph Placement Groups per Pool Calc.

Step2. Select a Ceph use case

Rados Gateway Only -- Use for S3 and/or Swift workloads only

Step3. Select special conditions (*optional*)

☑ Support Erasure Coding (EC) Supported only for the RGW and native librados object storage.

Step4. Adjust values for pools and PGs

☐ Set values for all pools

Pool Name	Pool Type	Size	K	M	OSD #	%Data	Target PGs per OSD	Suggested PG Count	
.rgw.root	Replicated	3			312	0.10	100	256	
default.rgw.intent-log	Replicated	3			312	0.10	100	256	
default.rgw.log	Replicated	3			312	0.10	100	256	
default.rgw.buckets.data	Erasure Coding		4	2	312	94.80	100	4096	
default.rgw.buckets.extra	Replicated	3			312	1.00	100	256	
default.rgw.buckets.index	Replicated	3			312	3.00	100	256	

Workload

Basic Info

ID: w13 **Name**: hybrid2x **Current State**: finished

Submitted At: Jun 9, 2019 9:19:03 PM **Started At**: Jun 9, 2019 9:19:03 PM **Stopped At**: Jun 11, 2019 9:19:15 PM

more info

Final Result

General Report

Op-Type	Op-Count	Byte-Count	Avg-ResTime	Avg-ProcTime	Throughput	Bandwidth	Succ-Ratio
read	4.46 mops	237.03 TB	616.61 ms	50.02 ms	25.79 op/s	1.37 GB/S	99.49%
list	740.94 kops	0 B	5.95 ms	5.95 ms	4.29 op/s	0 B/S	99.48%
write	1.19 mops	63.26 TB	574.48 ms	78.97 ms	6.91 op/s	366.11 MB/S	100%
delete	1.04 mops	0 B	16.71 ms	16.71 ms	6.05 op/s	0 B/S	100%

ResTime (RT) Details

Op-Type	60%-RT	80%-RT	90%-RT	95%-RT	99%-RT	100%-RT
read	< 80 ms	< 250 ms	< 820 ms	< 1,020 ms	< 12,700 ms	< 16,870 ms
list	< 10 ms	< 10 ms	< 10 ms	< 20 ms	< 50 ms	< 1,530 ms
write	< 70 ms	< 250 ms	< 790 ms	< 980 ms	< 11,870 ms	< 17,500 ms
delete	< 10 ms	< 30 ms	< 50 ms	< 70 ms	< 120 ms	< 10,110 ms

ID: w15 **Name**: delWrite2hr **Current State**: finished

Submitted At: Jun 11, 2019 10:20:35 PM **Started At**: Jun 11, 2019 10:20:35 PM **Stopped At**: Jun 12, 2019 12:20:48 AM

more info

Final Result

General Report

Op-Type	Op-Count	Byte-Count	Avg-ResTime	Avg-ProcTime	Throughput	Bandwidth	Succ-Ratio
write	305.56 kops	16.13 TB	462.57 ms	62.69 ms	42.45 op/s	2.24 GB/S	100%
delete	203.46 kops	0 B	12.7 ms	12.7 ms	28.27 op/s	0 B/S	100%

ResTime (RT) Details

Op-Type	60%-RT	80%-RT	90%-RT	95%-RT	99%-RT	100%-RT
write	< 20 ms	< 240 ms	< 660 ms	< 730 ms	< 8,770 ms	< 12,450 ms
delete	< 20 ms	< 20 ms	< 20 ms	< 20 ms	< 70 ms	< 10,040 ms

END