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

Jiangang.duan@intel.com

11/2013

The OpenStack Summit

Join Us In Hong Kong November 5-8, 2013

Agenda

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

Self introduction

- Jiangang Duan (段建钢)
- Open source Cloud Software team in Intel @ Shanghai
- OpenStack related performance
- Blog: http://software.intel.com/en-us/user/335367/track

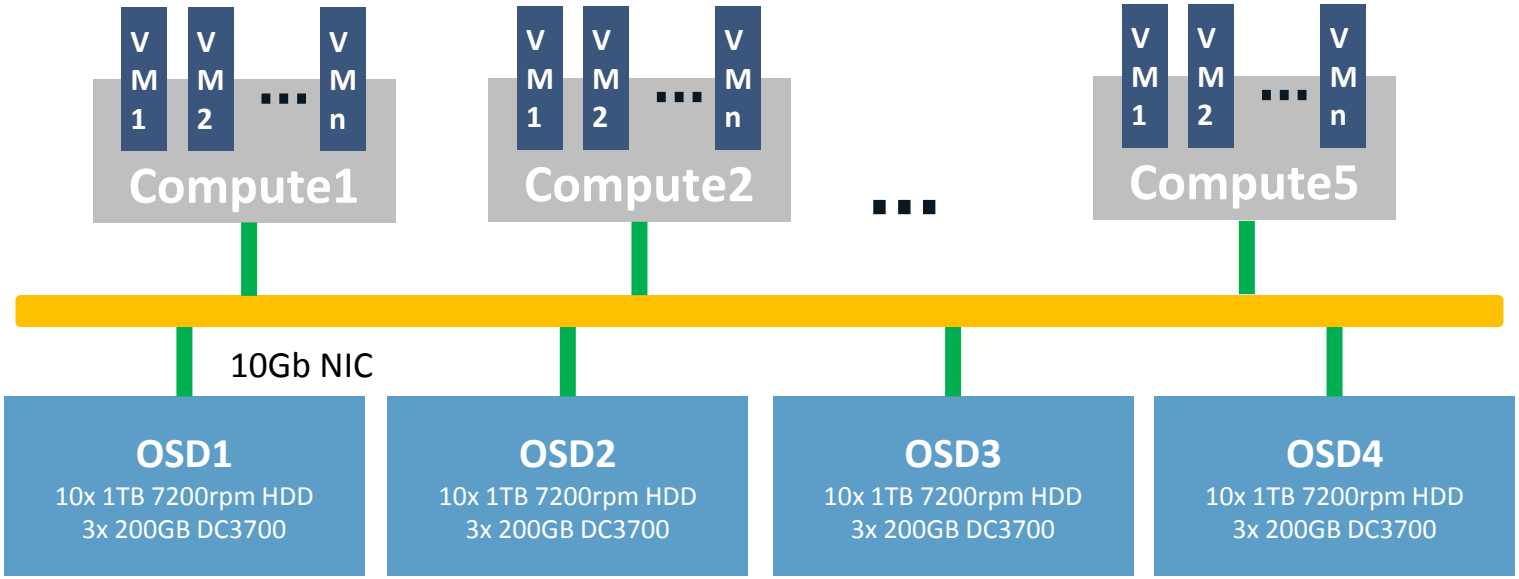
This is the team work:

- *Chendi Xue, Chen Zhang, Jian Zhang, Thomas Barnes, Xinxin Shu, Xiaoxi Chen, Yuan Zhou, Ziyuan Wang, etc.*

Agenda

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

Ceph Test Environment



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

Ceph Test-Bed Configuration: Software

Ceph cluster	
OS	Ubuntu 12.10
Kernel	3.6.6
Ceph	Cuttle Fish 0.61.2
FileSystem	XFS

Client host	
OS	Ubuntu 12.10
Kernel	3.5.0-22
OpenStack	Folsom (2012.2)

Client VM	
OS	Ubuntu 12.10
Kernel	3.5.0-22

Network tuning

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

XFS tuning

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

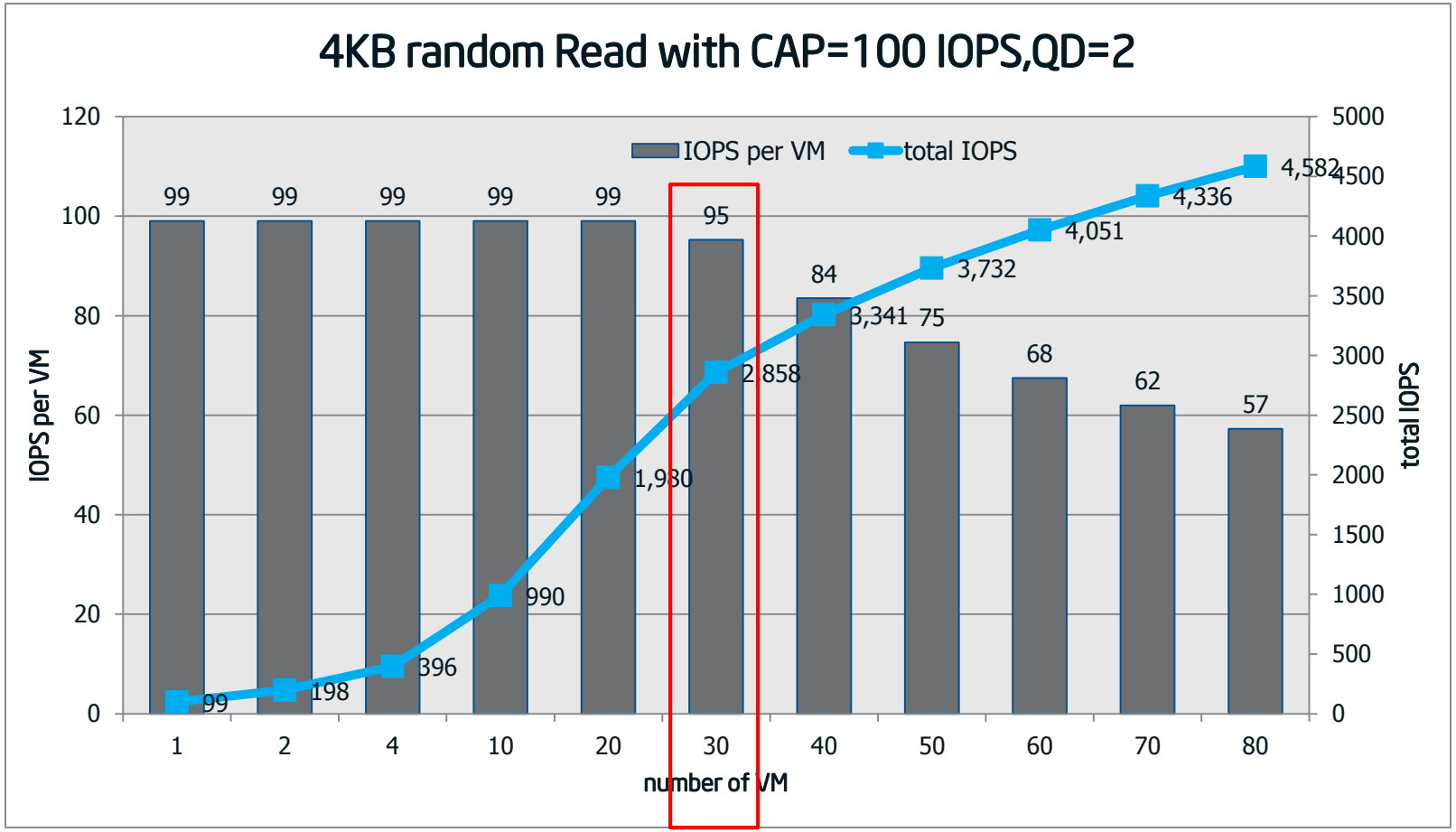
Ceph Tuning

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

Methodology

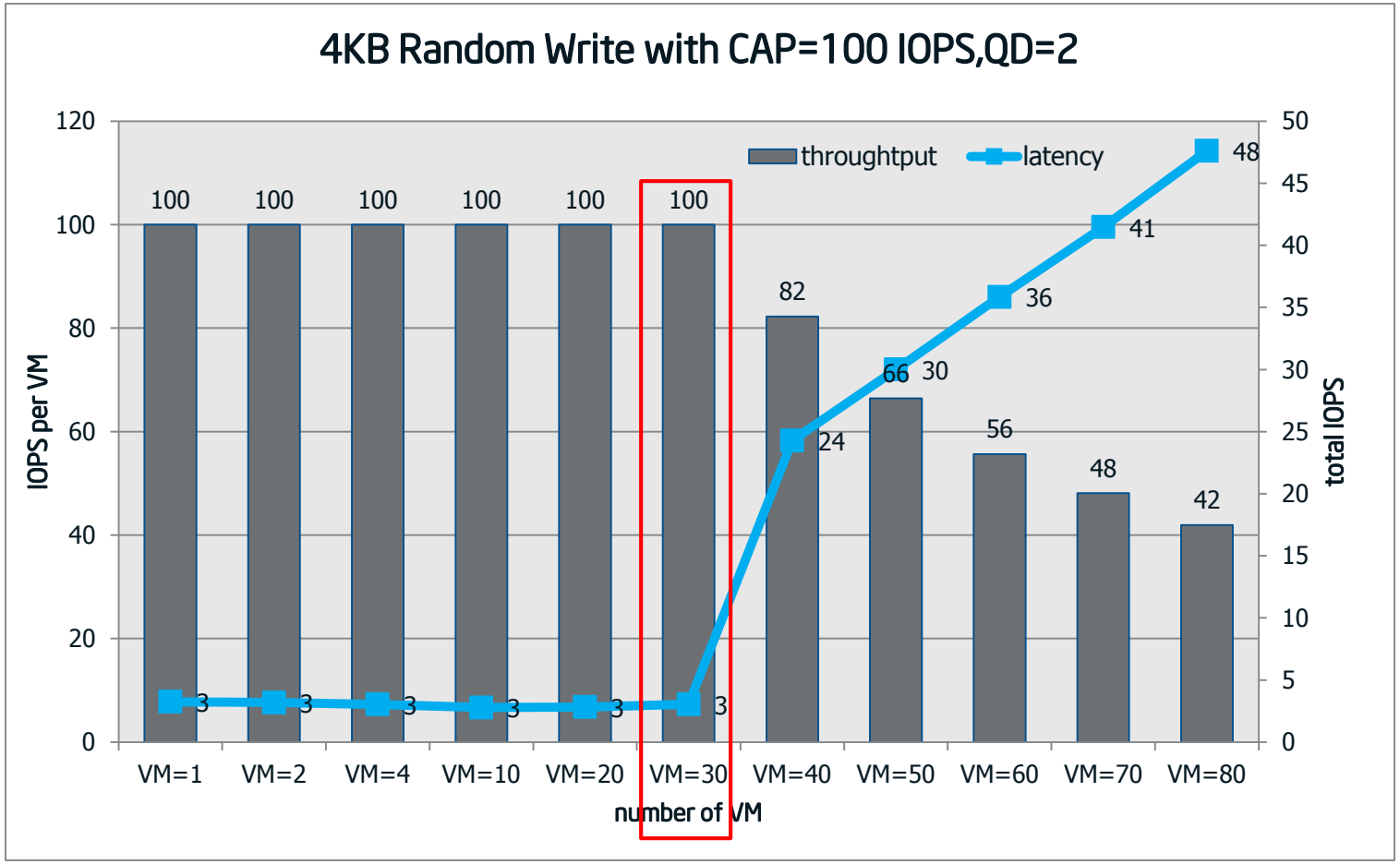
- Storage Interface:
  - RBD (aka volume) is attached to one virtual machine via *QEMU RBD* driver
- Workload
  - FIO to simulate 4 IO pattern:
    - *64k Sequential Read/Write (SR and SW)* with 60MB/sec max per VM
    - *4k Random Read/Write (RR and RW)* with 100IOPS max per VM
- Run Rules
  - Volume is *pre-allocated* before testing with dd
  - *Drop OSD page cache* before each measurement
  - Duration *400* secs warm-up, *600* secs data collection
  - 60GB per Volume, increase Stress VM number to get full load line
- QoS requirement:
  - Less than 20ms latency for random read/write
  - Per VM throughput larger than 90% of the predefine threshold
    - 54MB/sec for SR/SW and 90IOPS for RR/RW

Random read performance



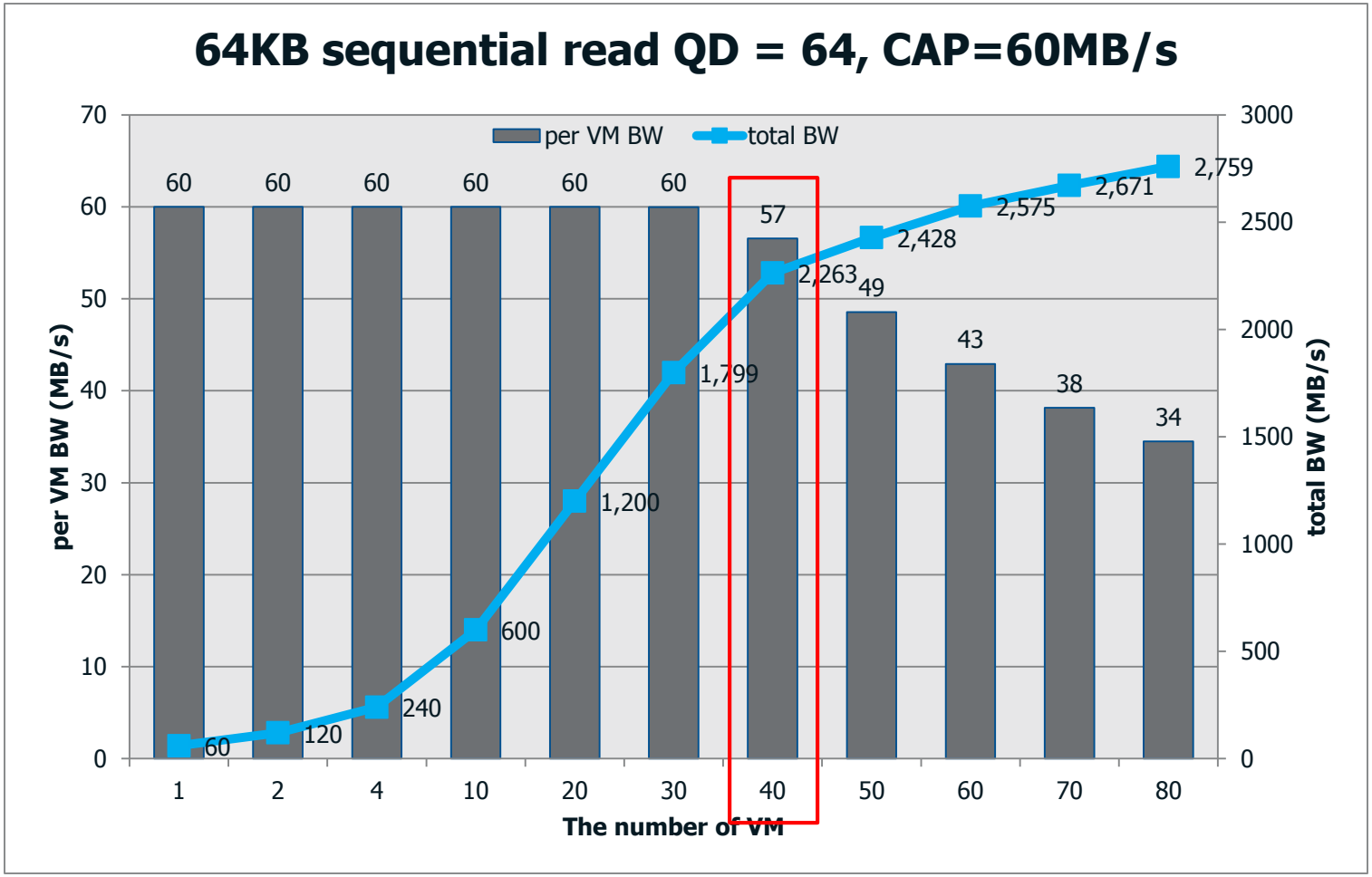
- Max measured value at ~4600 IOPS
- When scale to 30VM, per VM IOPS drop to 95

Random write performance



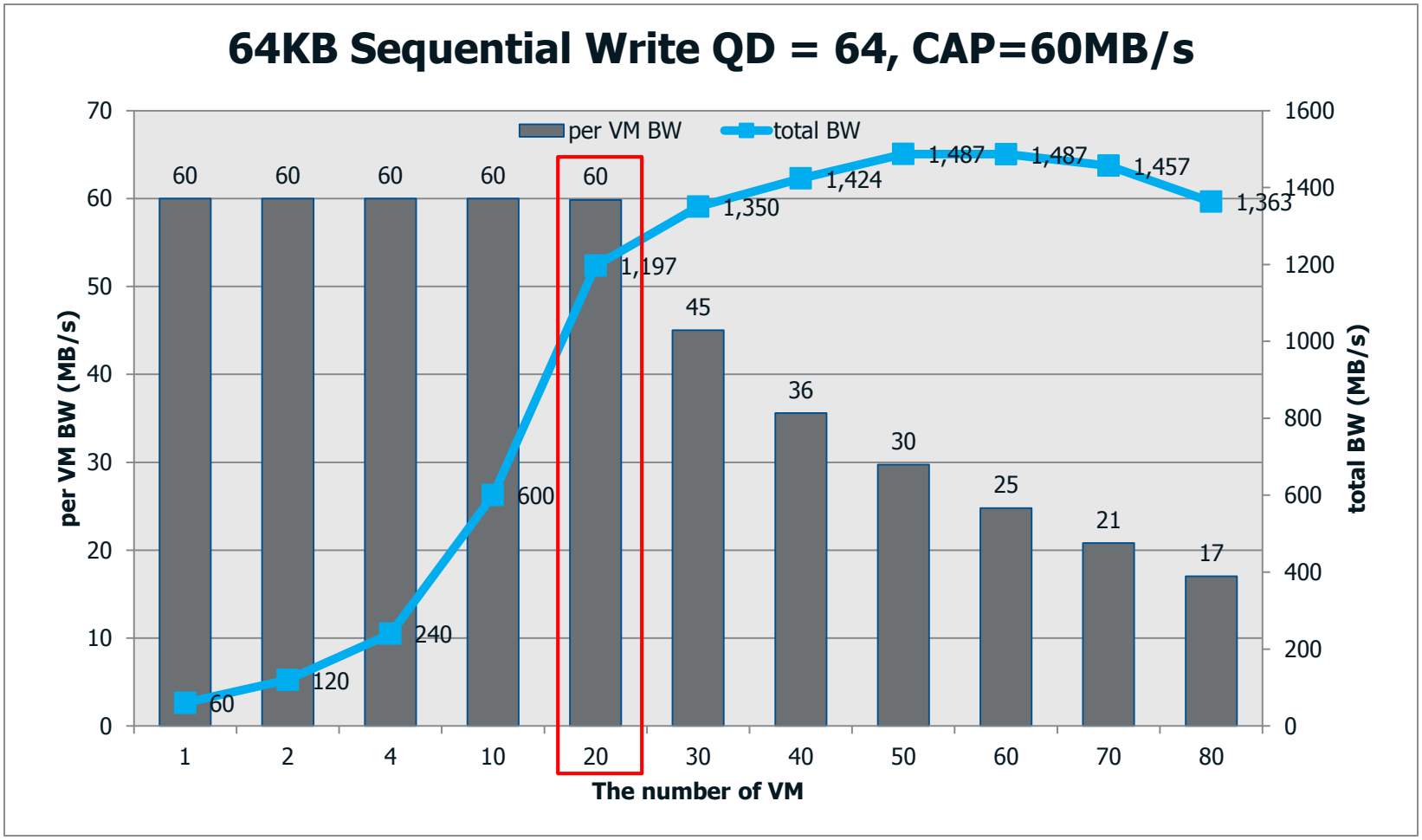
- Peak at ~3291 IOPS
- When scale to 40VM, per VM VW drop to 82 IOPS per VM

Sequential read performance



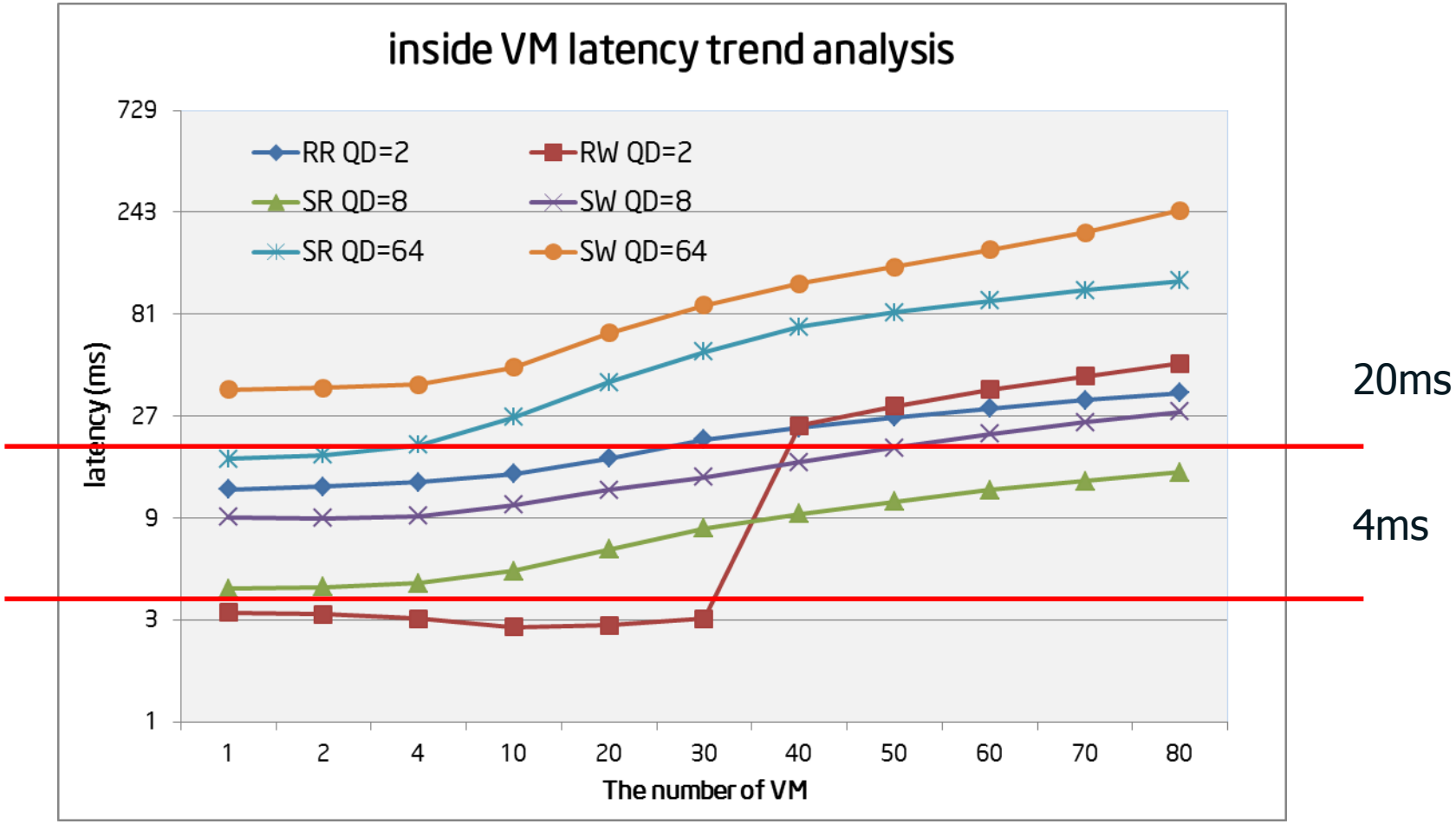
- Max measured value at ~2759MB/s
- When scale to 40VM, per VM VW drop to 57MB/sec

Sequential write performance



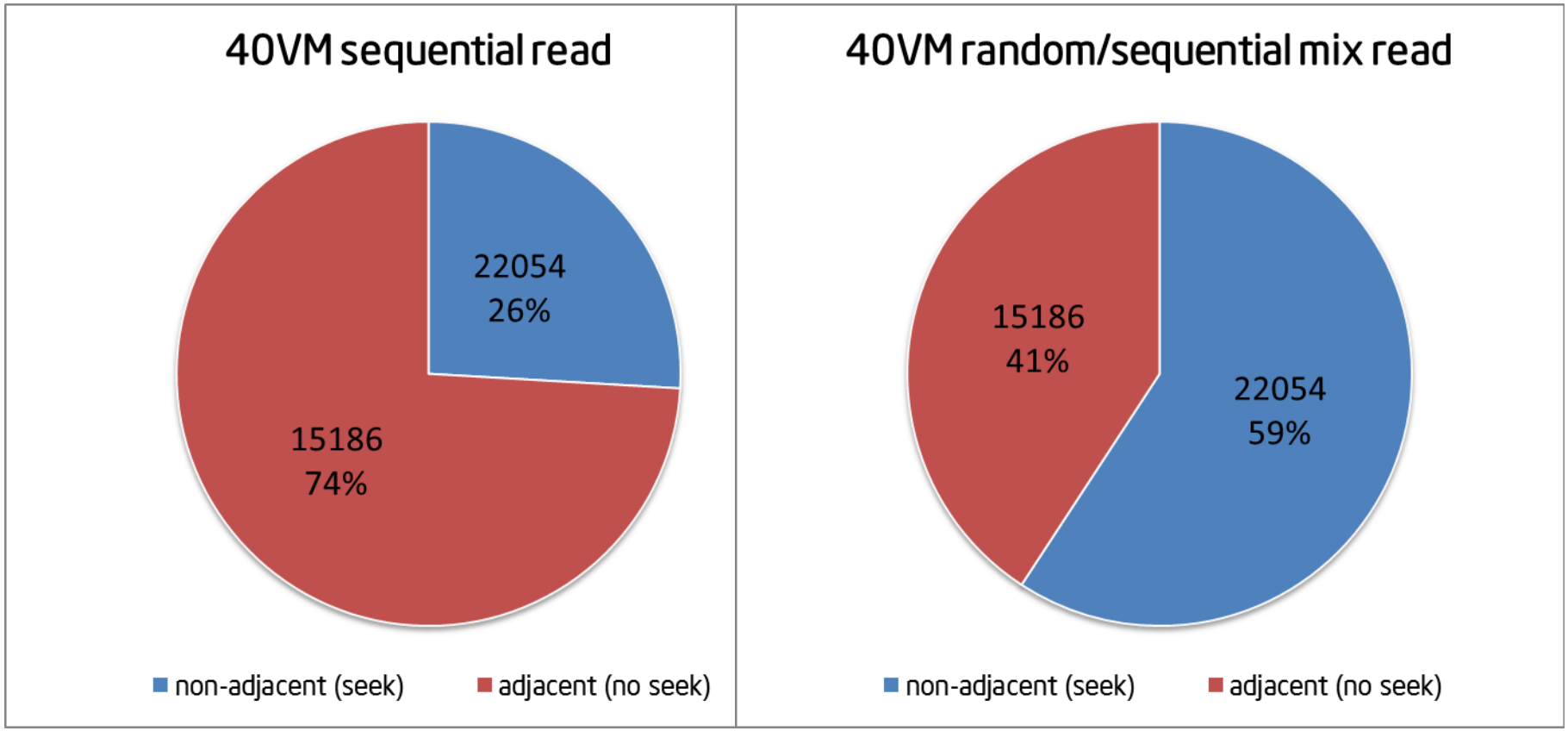
- Max at 1487MB/sec
- When scale to 30VM, per VM BW drop to 45MB/sec per VM

Latency analysis



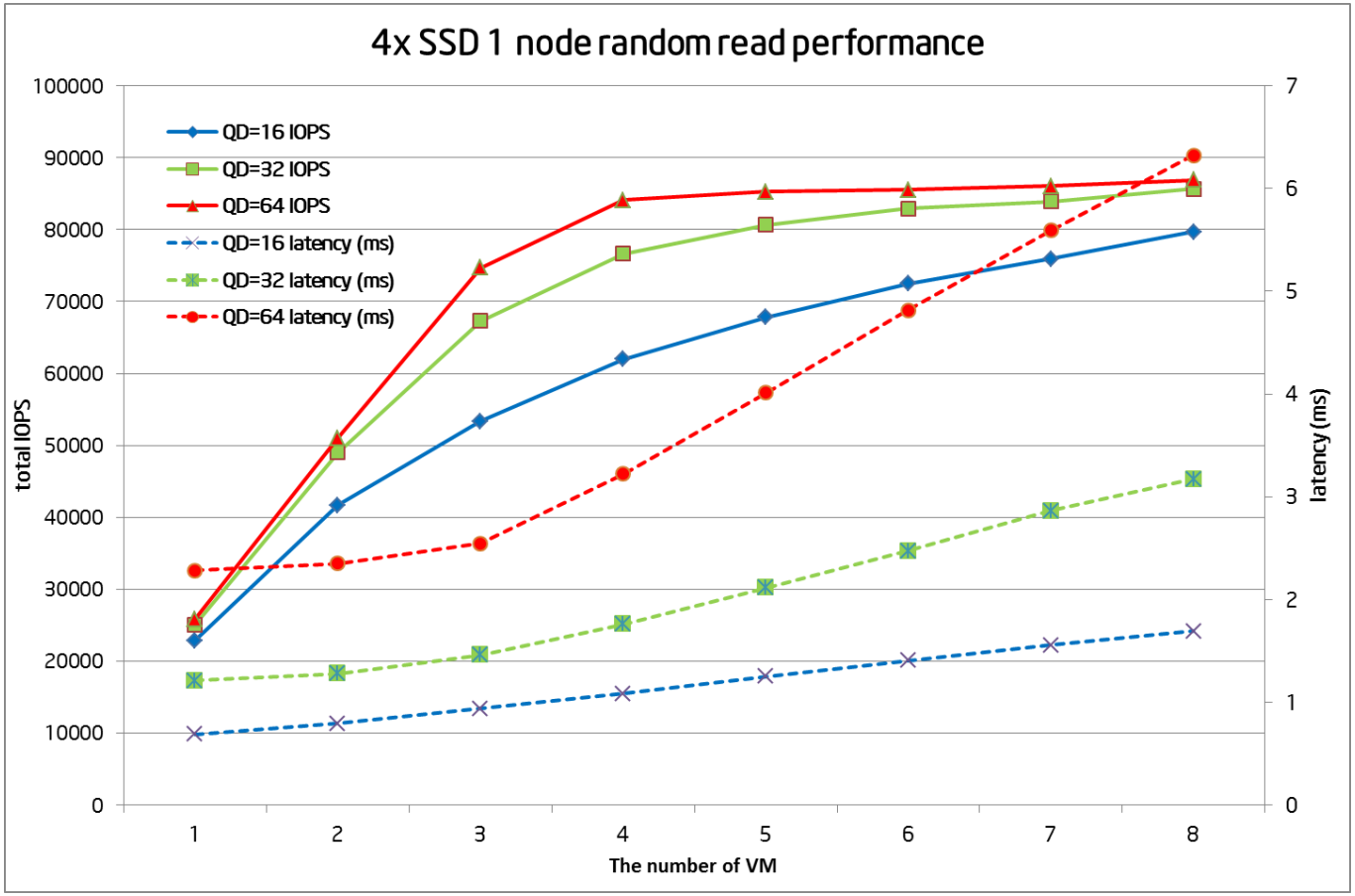
- Latency is strongly related to size of QD
- Random is good enough. Sequential still has improve room.

IO trace analysis for Sequential read



- Ceph converts the logical sequential to random disk access
- Read ahead can help, but not enough

Full SSD performance



- 4x 200GB SSD with 8 OSD (60GB data + 10GB journal each)
- Good enough full SSD performance.
- ~55K with 1ms latency; ~80K with 2ms, peak 85K with 3ms

Ceph Data Summary

workload pattern	Max measureable througput	throughput Consider QoS	in theory Disk throughput (IOPS and MB/s)	In theory Network throughput (IOPS and MB/s)	Efficiency
4KB random read CAP=100	4582	2858	3600	31250	79%
4KB random write CAP=100	3357	3000	4000	31250	75%
64KB seq read QD=64 CAP=60	2759	2263	6400	4000	57%
64KB seq write QD=64 CAP=60	1487	1197	3200	4000	37%

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

HW	total capacity (GB)	total throughput (IOPS)	60GB Volume number consider capacity	60GB Volume number consider performance
40x 1TB with replica=2	20000	2800	333.3	30
3 x 200GB SSD with replica=2	150	85000	2.5	850

Consider Random IO only

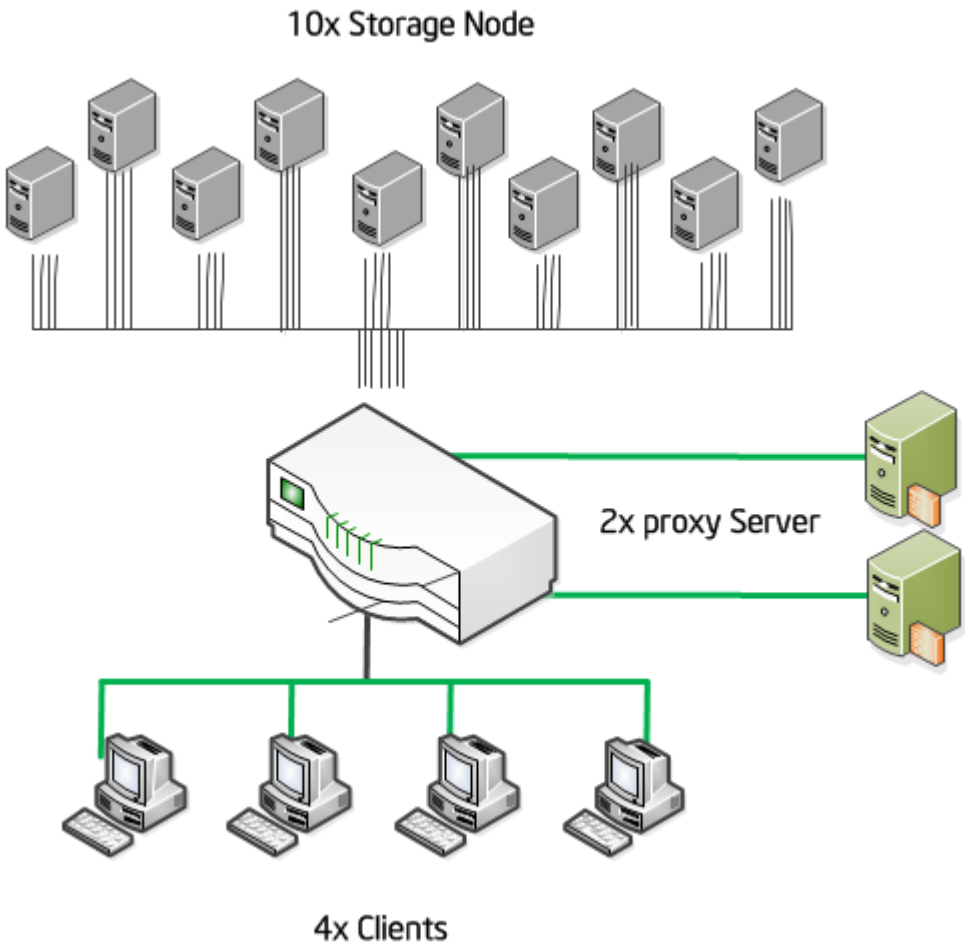
- Random performance is pretty good.
- Sequential performance has room to do better.
- Challenges and opportunities:
  - Throttle both BW and IOPS to better support multiple tenant case
  - Mix SSD and HDD together to achieve a balanced capacity/performance.

Agenda

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

H/W Configuration

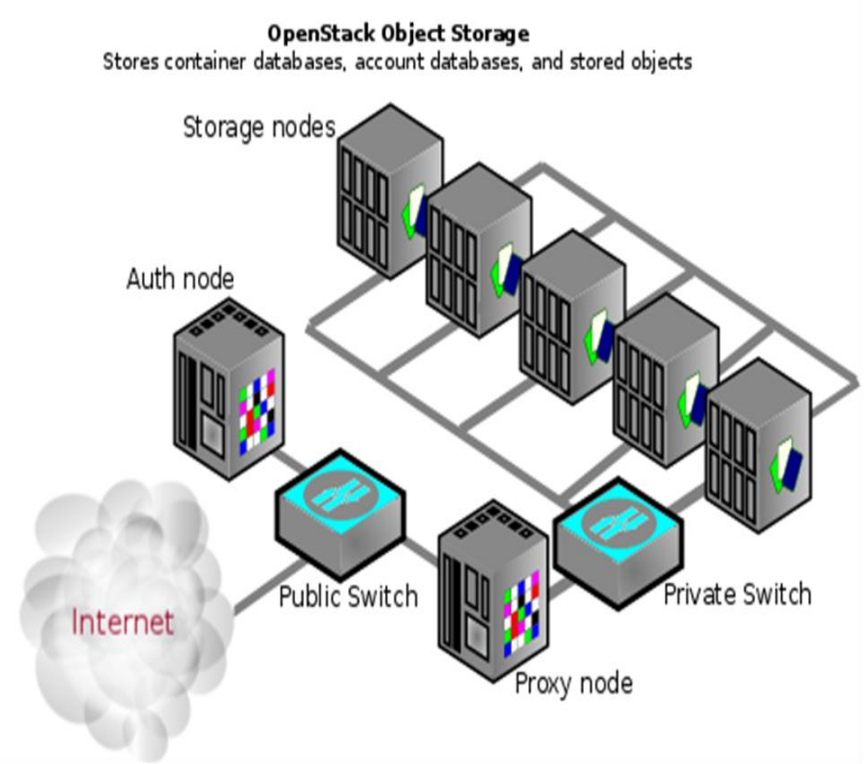
- Proxy Node x2
  - CPU = 2 * Intel Xeon E5 2.6GHz (8C/16T)
  - Memory = 64GB, 8x 8G DDR3 1333MHz
  - NIC = Intel 82599 Dual port 10GbE
  - OS = CentOS 6.4 (2.6.32-358.14.1)
  - BIOS = Turbo on, EIST on, CPU C-state ON
- Storage Node x10
  - CPU = 1* Intel Xeon E3 3.5GHz (4C/8T)
  - Memory =16 GB, 4x 4G DDR3 1333MHz
  - NIC = 1 * Intel 82576 Quad port 1GbE **(4*1Gb)**
  - Disk = 12* 1TB 7200rpm SATA, 1x 64GB SSD (3Gb/s) for account/container zone
  - OS = CentOS 6.4 (2.6.32-358.14.1)
- Network
  - 1x 10Gb to in/out of Proxy
  - 4x 1Gb for each storage node



Software Configuration

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

- Memcached Server
  - cache-size = 2048MB
  - max-concurrent-connections = 20480



Test Methodology

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

		Small-Scale Test	Large-Scale Test	
Test operations	Operations	Read/Write	Read/Write	
Swift	Object Size	128K/10M	128K	10M
	# of containers	100	10000	10000
	# of objects /container	100	10000	100
COBbench	# of workers	Start from 5, doubled until 2560	Start from 10, doubled until 2560	
	Ramp up (s)	300	300	
	Run time (s)	300	300	
	Ramp down (s)	100	100	
	Metrics	Operations per second	Operations per second	
	Compliant performance	avg response time < 200ms + object size/2M s	avg response time < 200ms + object size/2M s	
	B/W	MB/s	MB/s	

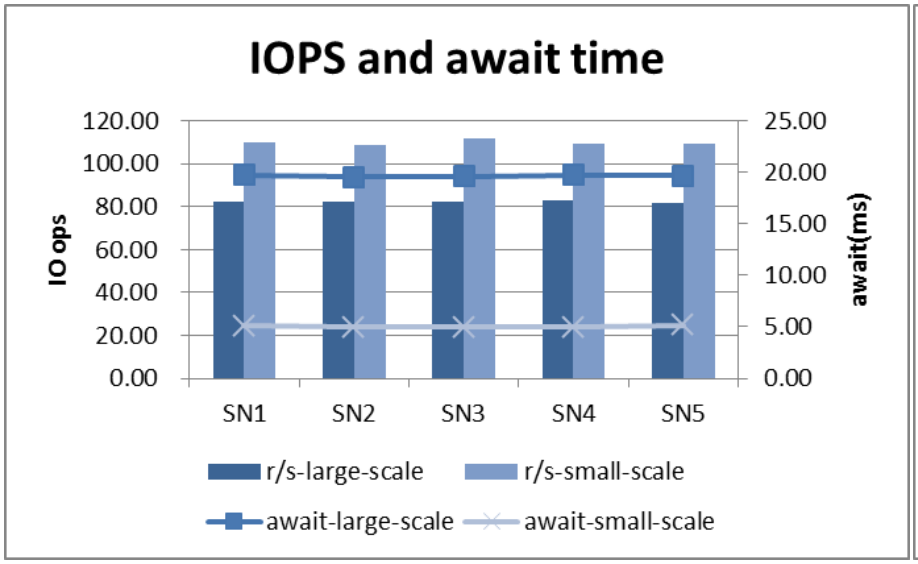
Swift Performance Overview

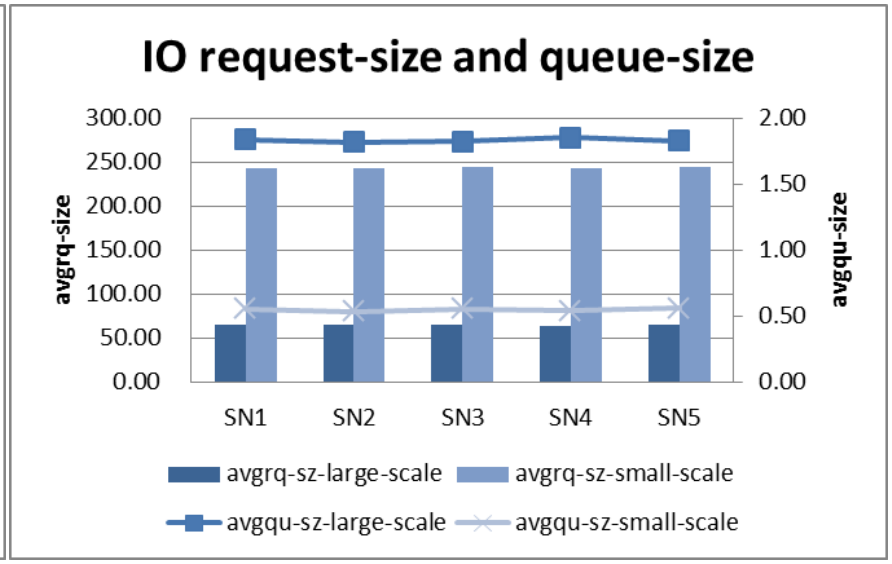
#con x # obj	Object-Size	RW-Mode	Worker-Count	Avg-ResTime	Throughput	Bottleneck
	--	--	--	ms	op/s	--
100x100	128KB	Read	640	51.77	12359.84	Proxy CPU = 100%, SN CPU = 72%
		Write	640	213.12	3001.65	Proxy CPU=47%, SN CPU = 96%
	10MB	Read	160	699.3	228.78	Proxy CPU = 61%, SN CPU = 55%, Proxy NIC 9.6Gb/s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

#con x # obj	Object-Size	RW-Mode	Worker-Count	Avg-ResTime	Throughput	Bottleneck
	--	--	--	ms	op/s	--
10000*10000	128KB	Read	640	248.56	2574.78	Proxy CPU = 25% SN CPU = 52%, disk latency 22ms
		Write	80	62.75	1272.97	Proxy CPU = 21%, SN CPU = 57%, disk latency 18ms
10000*100	10MB	Read	160	699.17	228.83	Proxy CPU = 56%, SN CPU = 62%, Proxy NIC 9.6Gb/s
		Write	320	4096.3	78.12	Proxy CPU = 16%, SN CPU = 81%, Proxy NIC 9.6Gb/s

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

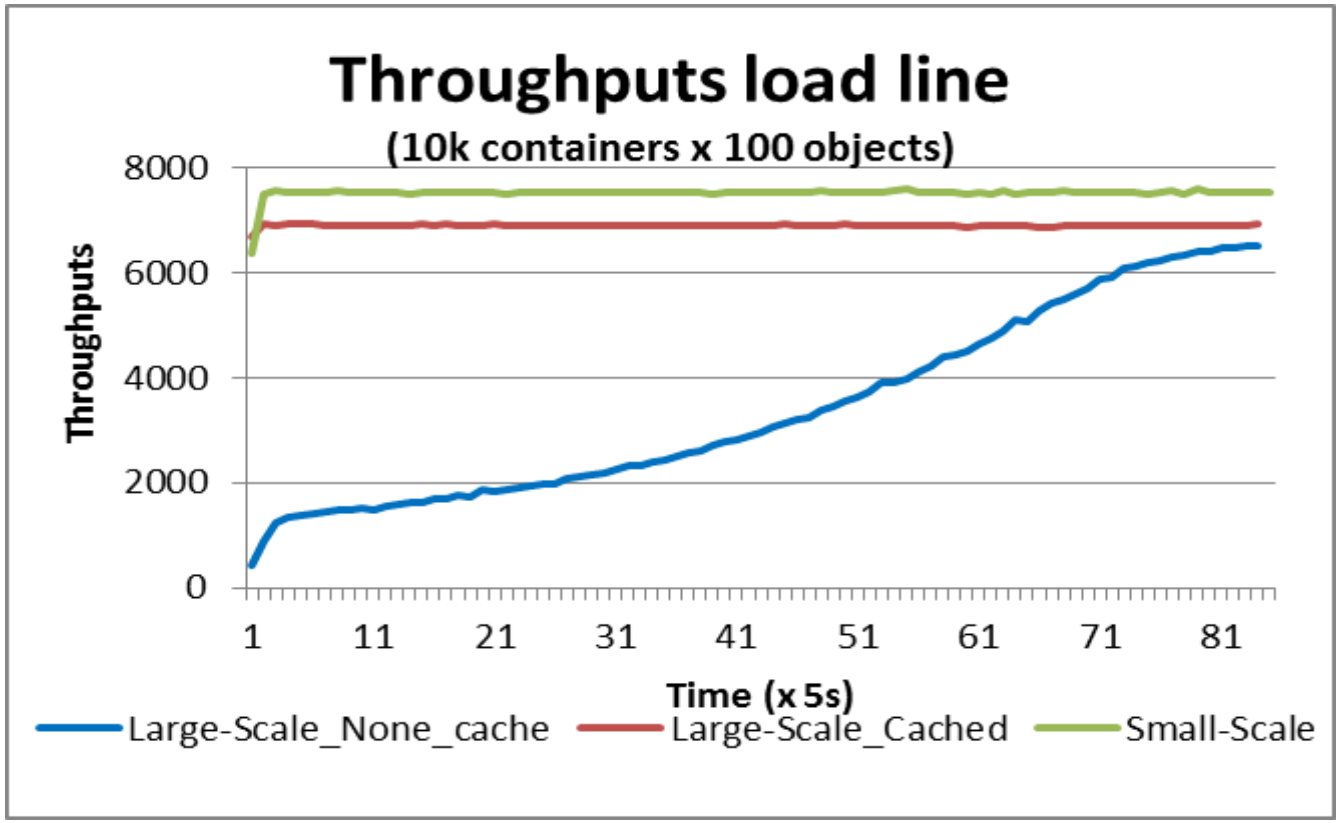
Problem : large-scale test 128K read 80+% drops





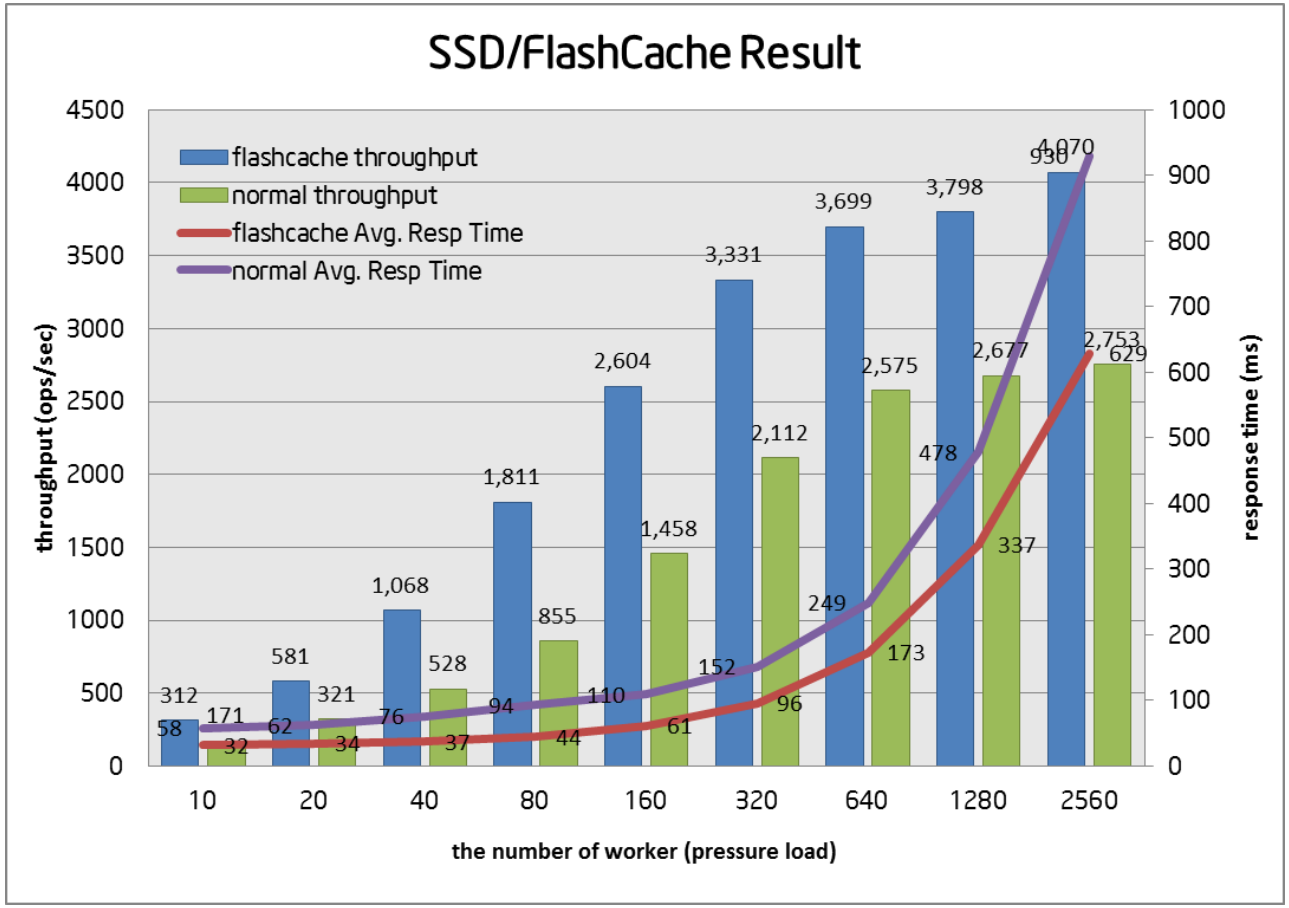
- Different IO pattern
  - large-scale : 34 KB per op
  - small-scale: 122KB per op
- Blktrace shows lots of IO generated by XFS file system
  - Meta data cannot be cached with 10k * 10k objects
  - Frequent disk access for that info

Option 1 : Using larger memory to cache Inodes



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

Option 2 : Using SSD to cache Inodes



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

Alternative Solutions and Next step

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

Summary

- Many emerging open source storage projects to satisfy BIG DATA requirement
- Ceph and swift are widely used in OpenStack environment
- Good result however still improvement areas
- Call action to work together

Disclaimers