Optimizing Ceph for All-Flash Architectures

Vijayendra Shamanna (Viju)
Senior Manager
Systems and Software Solutions

SanDisk

Forward-Looking Statements

During our meeting today we may make forward-looking statements.

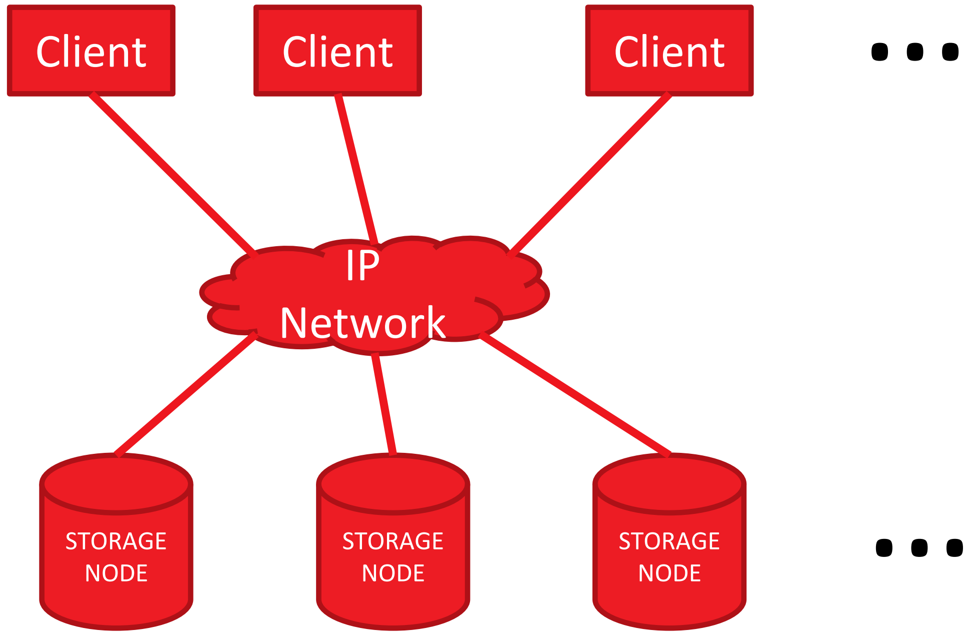
Any statement that refers to expectations, projections or other characterizations of future events or circumstances is a forward-looking statement, including those relating to future technologies, products and applications. Actual results may differ materially from those expressed in these forward-looking statements due to factors detailed under the caption “Risk Factors” and elsewhere in the documents we file from time to time with the SEC, including our annual and quarterly reports.

We undertake no obligation to update these forward-looking statements, which speak only as of the date hereof

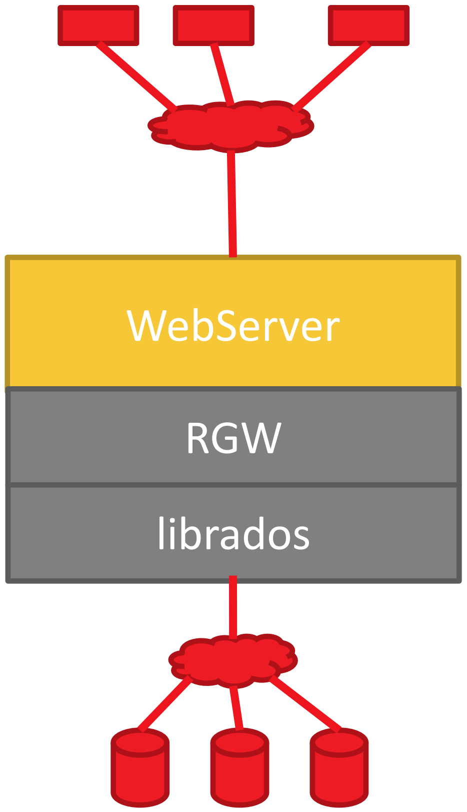
What is Ceph?

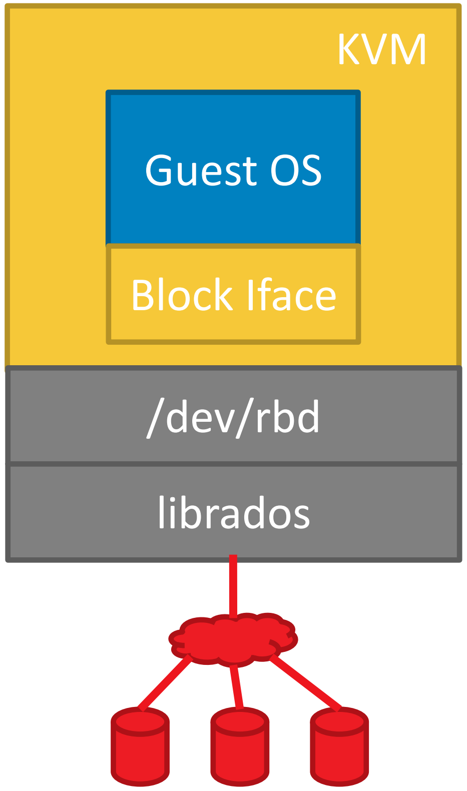
- A Distributed/Clustered Storage System
- File, Object and Block interfaces to common storage substrate
- Ceph is a mostly LGPL open source project
  - Part of Linux kernel and most distros since 2.6.x

Ceph Architecture

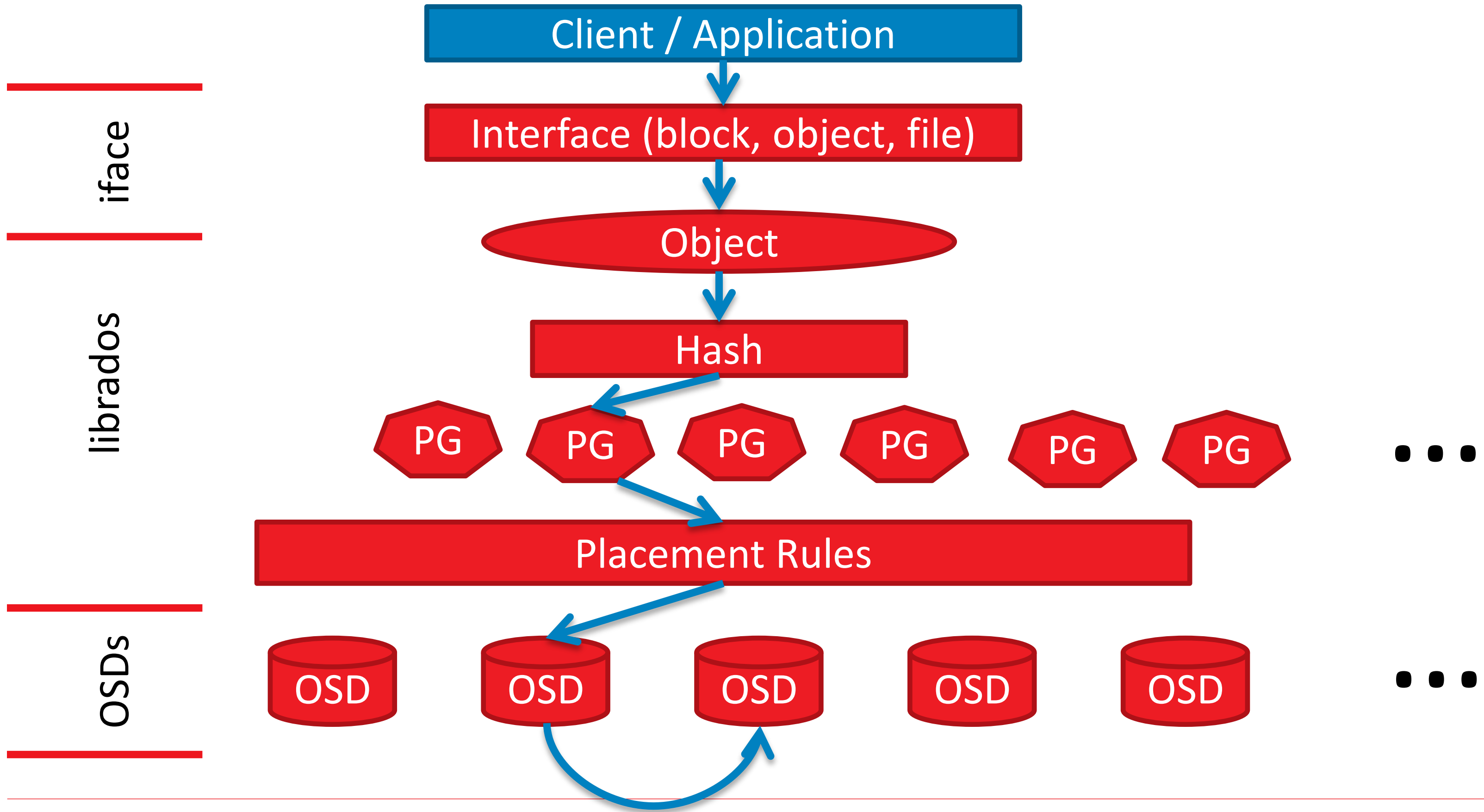


Ceph Deployment Modes

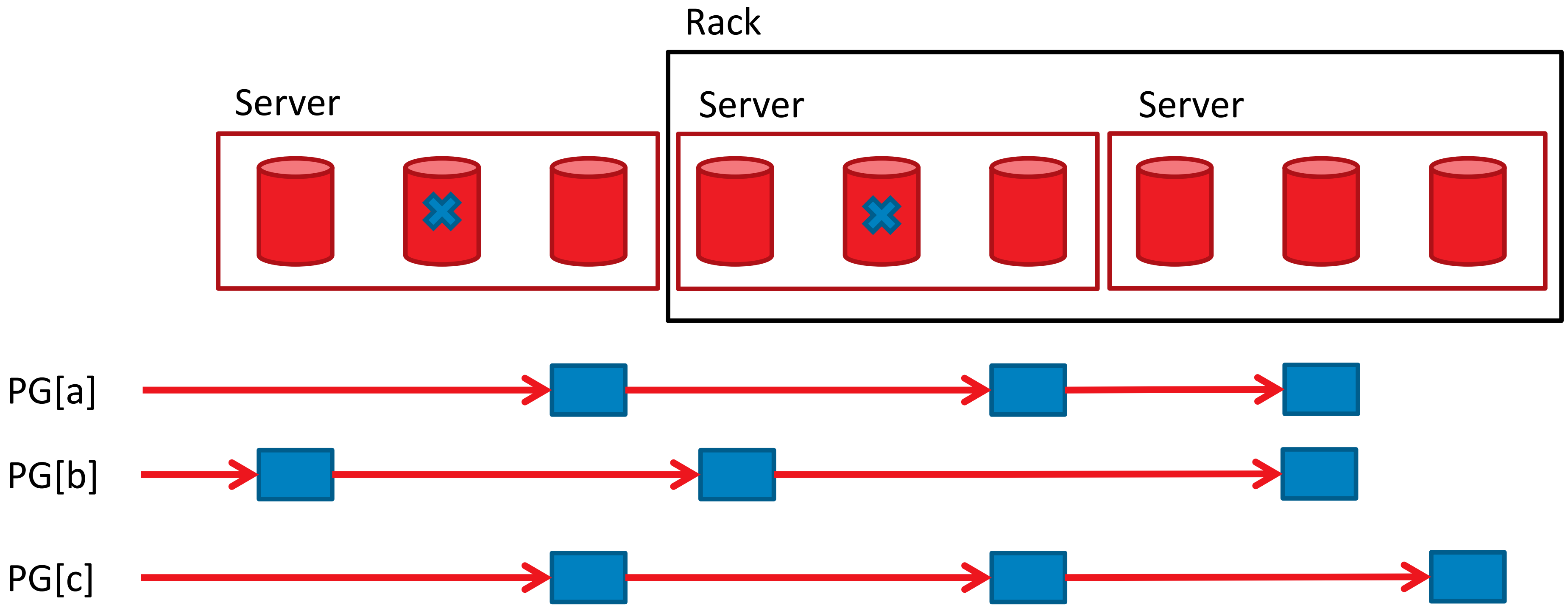




Ceph Operation



Ceph Placement Rules and Data Protection

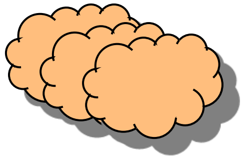


Ceph's built in Data Integrity & Recovery

- Checksum based "patrol reads"
  - Local read of object, exchange of checksum over network
- Journal-based Recovery
  - Short down time (tunable)
  - Re-syncs only changed objects
- Object-based Recovery
  - Long down time OR partial storage loss
  - Object checksums exchanged over network

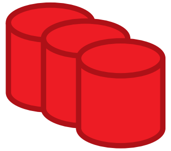
Enterprise Storage Features

* Future



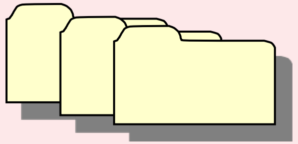
OBJECT STORAGE

- RESTful Interface
- Multi-tenant
- S3 & Swift
- Striped Objects
- Erasure Coding
- Keystone authentication
- Geo-Replication



BLOCK STORAGE

- iSCSI
- Snapshots
- Copy-on Write Clones
- Thin Provisioning
- Configurable Striping
- Incremental backups
- Open Stack integration



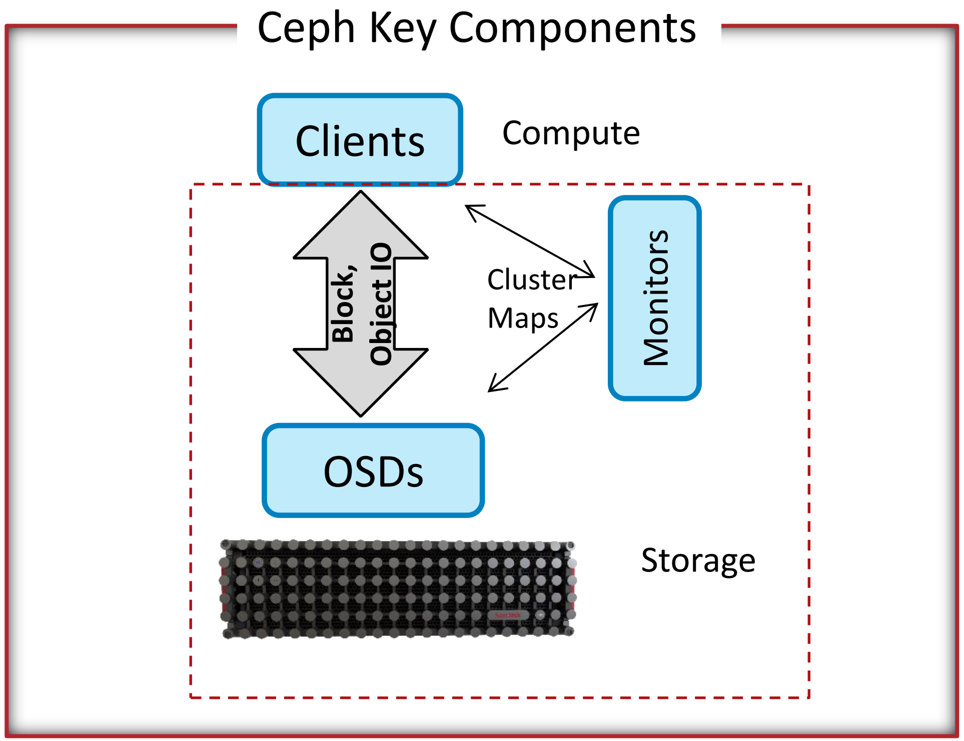
FILE SYSTEM *

- CIFS/NFS
- POSIX compliance
- Distributed Metadata
- Dynamic Rebalancing
- Configurable Striping
- Linux Kernel

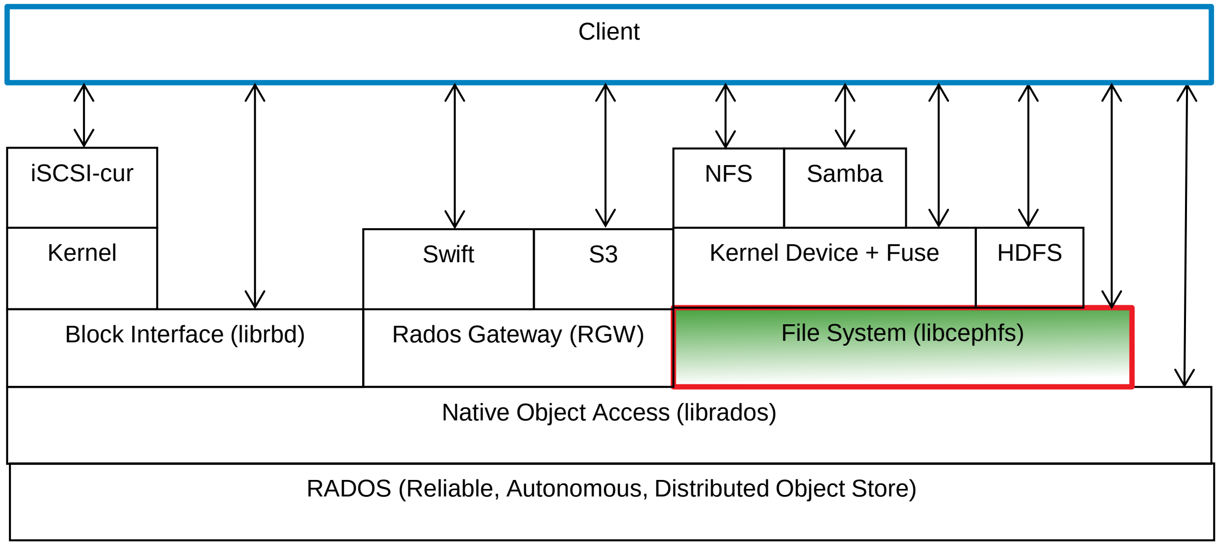
Software Architecture

– Key Components of System built for Hyper Scale

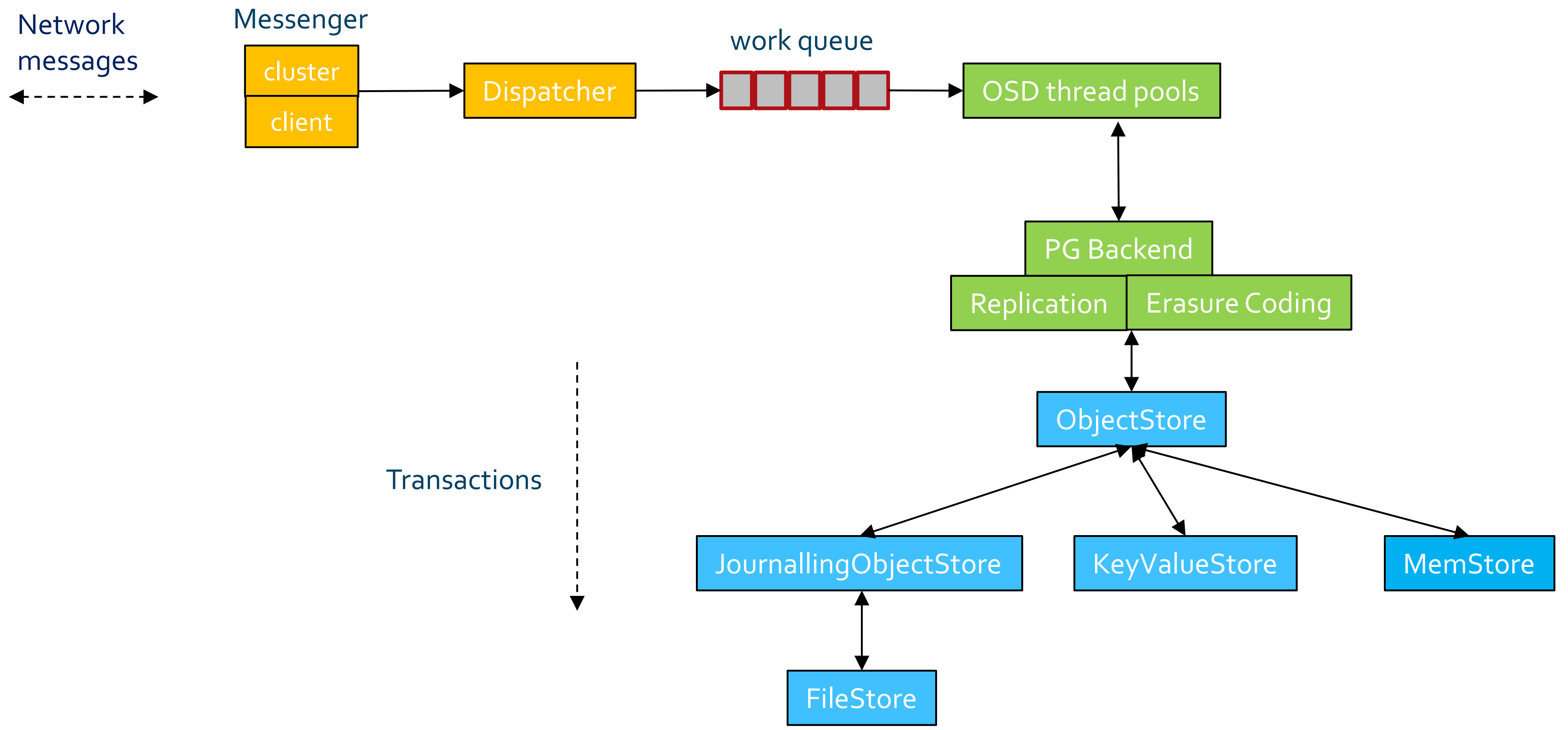
- Storage Clients
  - Standard Interface to use the data (POSIX, Device, Swift/S3…)
  - Transparent for Applications
  - Intelligent, Coordinates with peers
- Object Storage Cluster (OSDs)
  - Stores all data and metadata into flexible-sized containers – Objects
- Cluster Monitors
  - lightweight process for Authentication, Cluster Membership, Critical Cluster State
- Clients authenticates with monitors, direct IO to OSDs for scalable performance
  - No gateways, brokers, lookup tables, indirection …



Ceph Interface Eco System



OSD Architecture



Messenger layer

- Removed Dispatcher and introduced a “fast path” mechanism for read/write requests
  - Same mechanism is now present on client side (librados) as well
- Fine grained locking in message transmit path
- Introduced an efficient buffering mechanism for improved throughput
- Configuration options to disable message signing, CRC check etc

OSD Request Processing

- Running with Memstore backend revealed bottleneck in OSD thread pool code
- OSD worker thread pool mutex heavily contended
- Implemented a sharded worker thread pool. Requests sharded based on their pg (placement group) identifier
- Configuration options to set number of shards and number of worker threads per shard
- Optimized OpTracking path (Sharded Queue and removed redundant locks)

FileStore improvements

- Eliminated backend storage from picture by using a small workload (FileStore served data from page cache)
- Severe lock contention in LRU FD (file descriptor) cache. Implemented a sharded version of LRU cache
- CollectionIndex (per-PG) object was being created upon every IO request. Implemented a cache for the same as PG info doesn’t change often
- Optimized “Object-name to XFS file name” mapping function
- Removed redundant snapshot related checks in parent read processing path

Inconsistent Performance Observation

- Large performance variations on different pools across multiple clients
- First client after cluster restart gets maximum performance irrespective of the pool
- Continued degraded performance from clients starting later
- Issue also observed on read I/O with unpopulated RBD images – Ruled out FS issues
- Performance counters show up to 3x increase in latency through the I/O path with no particular bottleneck

Issue with TCmalloc

- Perf top shows rapid increase in time spent in TCmalloc functions

```
14.75%  libtcmalloc.so.4.1.2     [.] tcmalloc::CentralFreeList::FetchFromSpans()
7.46%  libtcmalloc.so.4.1.2     [.] tcmalloc::ThreadCache::ReleaseToCentralCache(tcmalloc::ThreadCache::FreeList*, unsigned
long, int)
```

- I/O from different client causing new threads in sharded thread pool to process I/O
- Causing memory movement from thread caches and increasing alloc/free latency
- JEmalloc and Glibc malloc do not exhibit this behavior
- JEmalloc build option added to Ceph Hammer
- Setting TCmalloc tunable 'TCMALLOC_MAX_TOTAL_THREAD_CACHE_BYTES’ to larger value (64M) alleviates the issue

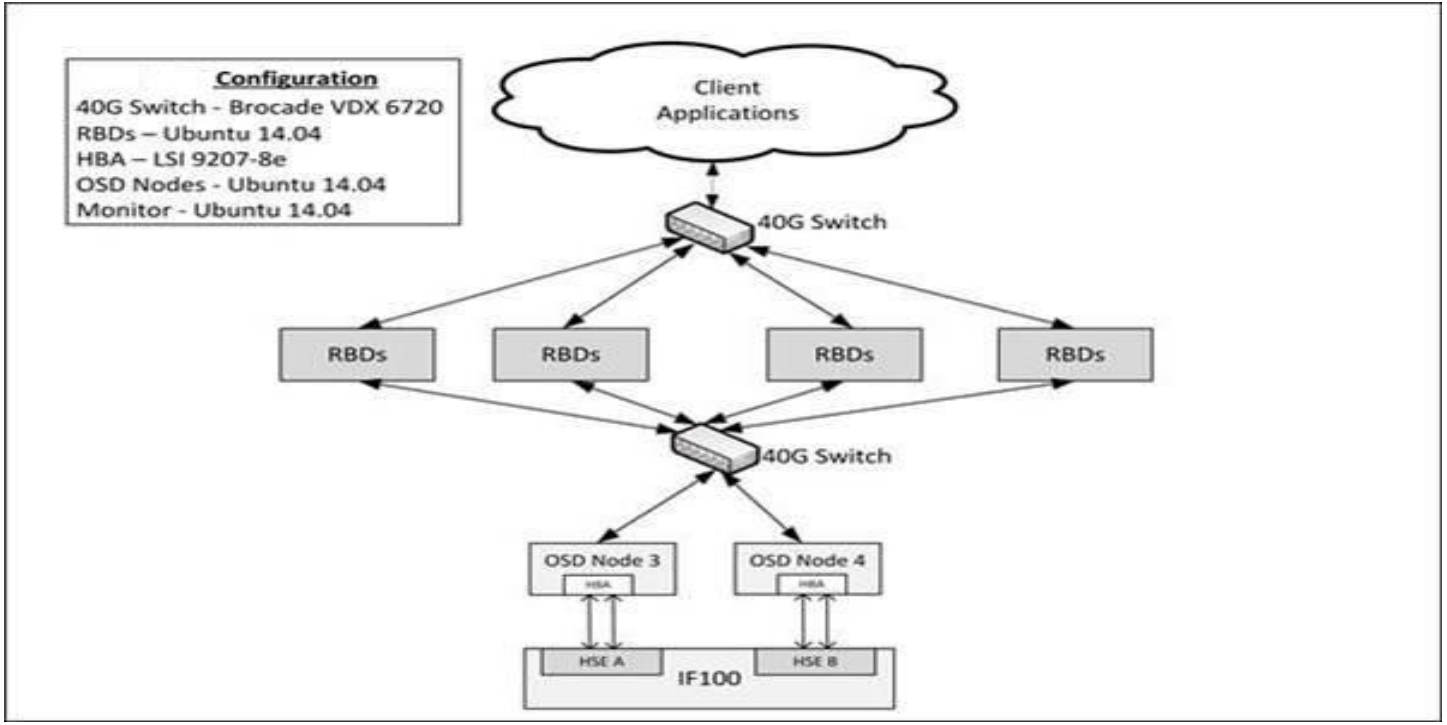
Client Optimizations

- Ceph by default, turns Nagle's algorithm OFF
- RBD kernel driver ignored TCP_NODELAY setting
- Large latency variations at lower queue depths
- Changes to RBD driver submitted upstream

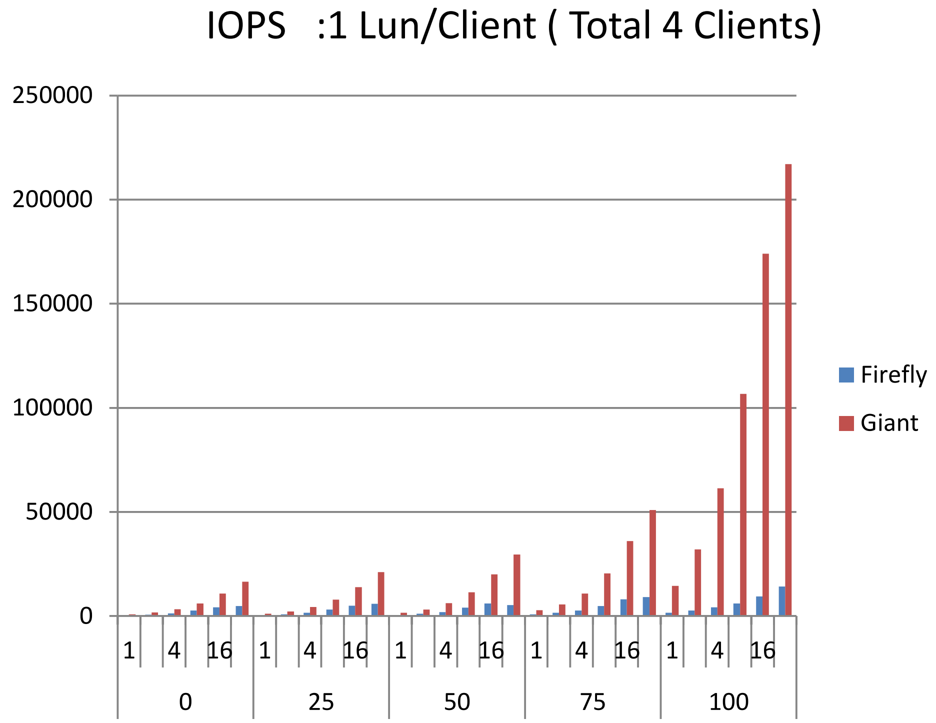
Results!

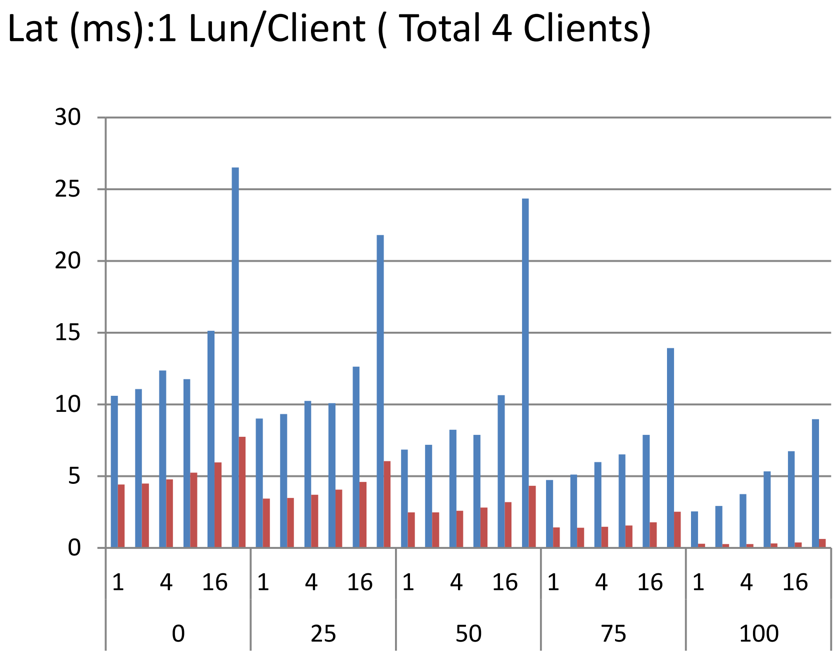
SanDisk®

Hardware Topology


Configuration
40G Switch - Brocade VDX 6720
RBDs – Ubuntu 14.04
HBA – LSI 9207-8e
OSD Nodes - Ubuntu 14.04
Monitor - Ubuntu 14.04

8K Random IOPs Performance -1 RBD /Client

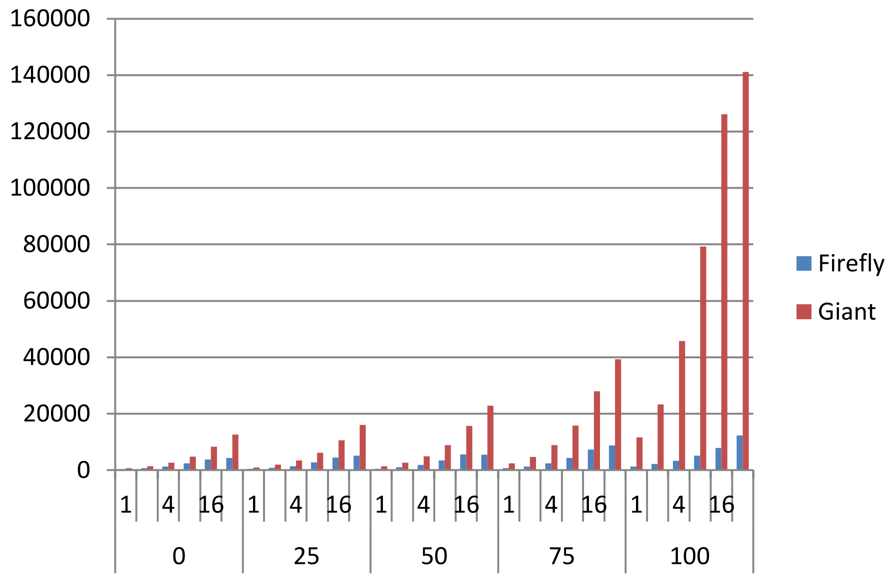




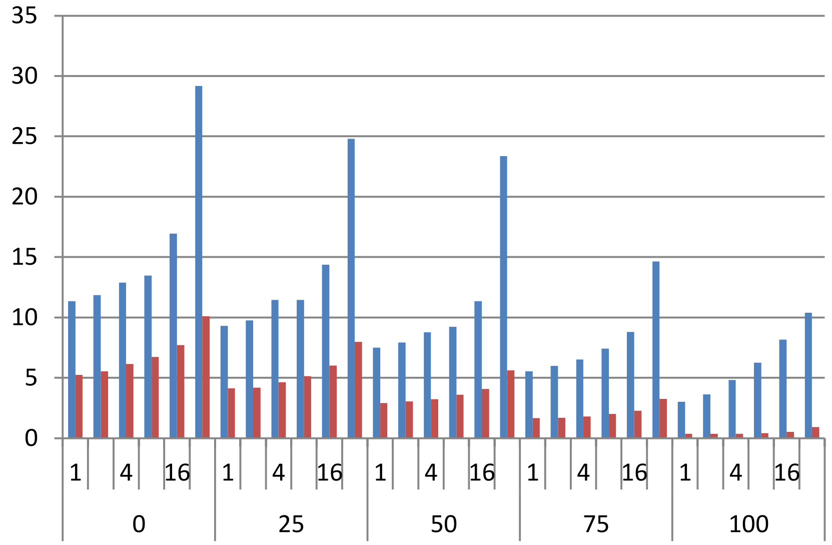
[Queue Depth]
Read Percent

64K Random IOPs Performance -1 RBD /Client

IOPS :1 Lun/Client (Total 4 Clients)

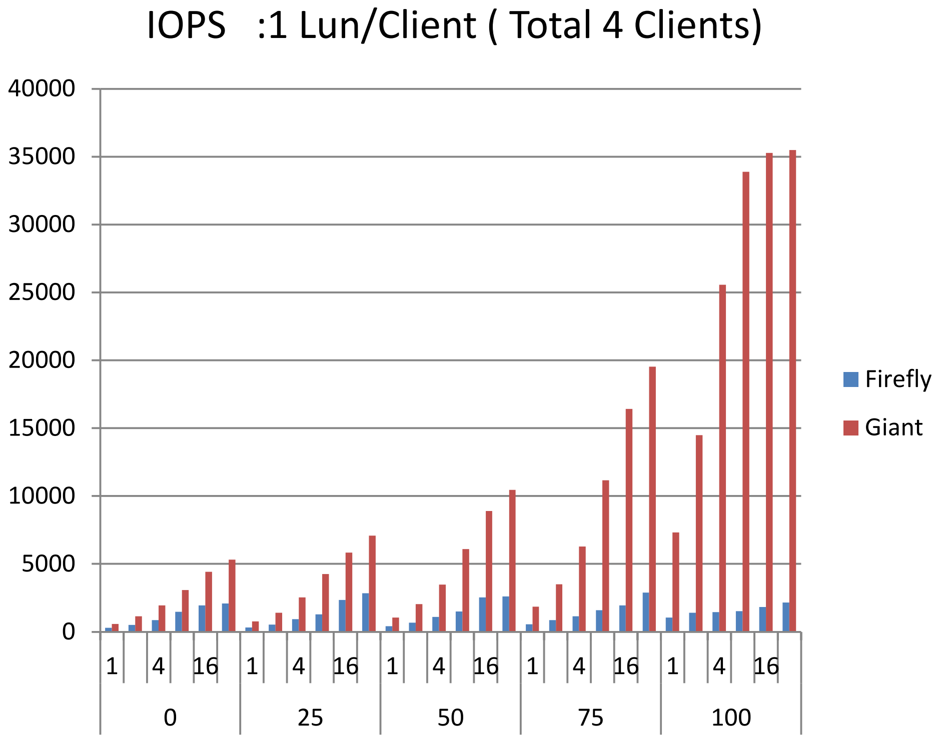


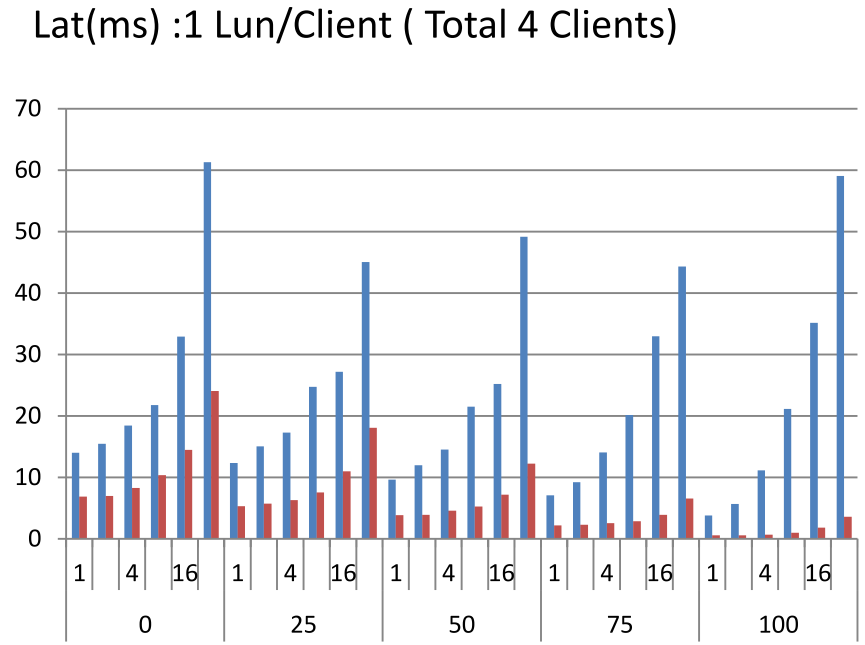
Lat(ms) :1 Lun/Client (Total 4 Clients)



[Queue Depth]
Read Percent

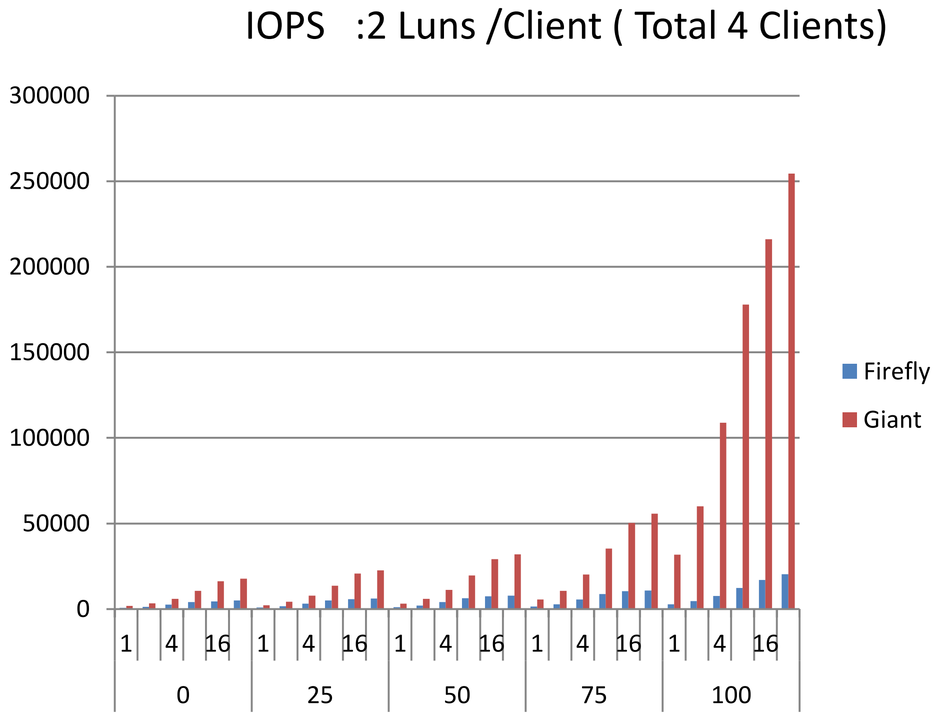
256K Random IOPs Performance -1 RBD /Client

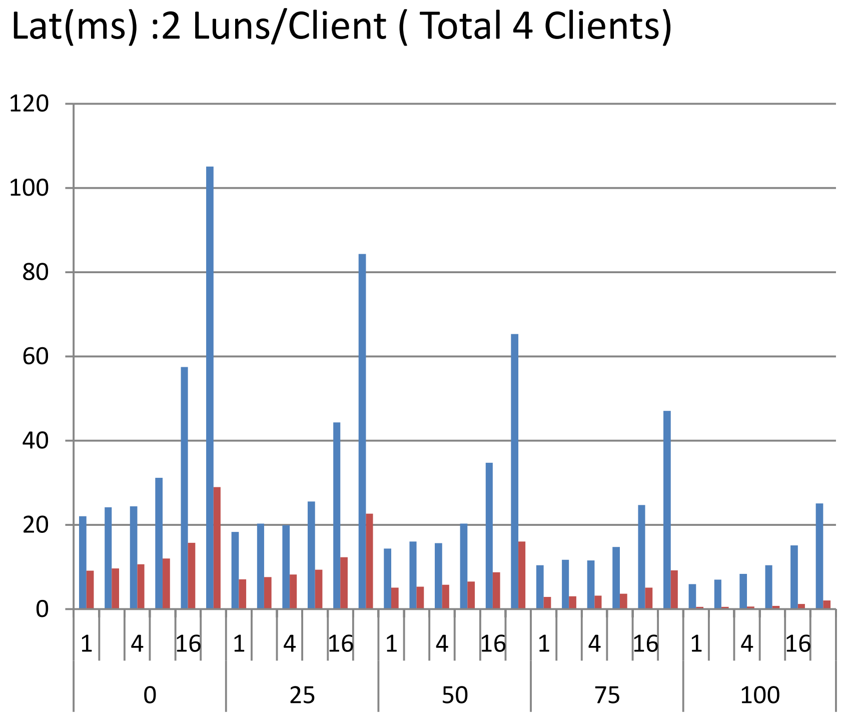




[Queue Depth]
Read Percent

8K Random IOPs Performance -2 RBD /Client





[Queue Depth]
Read Percent

SanDisk Emerging Storage Solutions (EMS)
Our goal for EMS

- Expand the usage of flash for newer enterprise workloads (Ex: Ceph, Hadoop, Content Repositories, Media Streaming, Clustered Database Applications, etc)
- … through disruptive innovation of "disaggregation" of storage from compute
- … by building shared storage solutions
- … thereby allowing customers to lower TCO relative to HDD based offerings through scaling storage and compute separately

Mission: Provide the best building blocks for "shared storage" flash solutions.

SanDisk Systems & Software Solutions

Building Blocks Transforming Hyperscale

Massive Capacity

InfiniFlash™

Capacity & High Density for Big Data Analytics, Media Services, Content Repositories

512TB 3U, 780K IOPS, SAS storage
Compelling TCO– Space/Energy Savings

Extreme Performance for Applications..

ION Accelerator™

Accelerate Database, OLTP & other high performance workloads

1.7M IOPS, 23 GB/s, 56 us latency
Lower costs with IT consolidation

ZetaScale™

Lower TCO while retaining performance for in-memory databases

Run multiple TB data sets on SSDs vs. GB data set on DRAM

Deliver up to 5:1 server consolidation, improving TCO

.. Servers/Virtualization

FlashSoft®

Server-side SSD based caching to reduce I/O Latency

Improve application performance
Maximize storage infrastructure investment

Thank you ! Questions?

vijayendra.shamanna@sandisk.com