CLOUD COMPUTING STUDY

By

Group E

Cloud Computing

UT DALLAS

RAJA JAYA CHANDRA MANNEM
mrjayachandra@gmail.com 972.730.4304

ceph

CEPH Is ...

- Open Sourced
  - git@github.com:ceph/ceph.git
- Distributed File System
- Infinitely Scalable (can handle petabytes or even exabytes of data)
- Highly Reliable
- Fault Tolerant
- Based on RADOS storage service
- Dependent on the CRUSH Algorithm

Ceph File System Keywords

- Ceph Clients - Ceph Object Gateway, Ceph Block Device, Ceph File System, Custom Apps
- Ceph Storage Cluster - OSDs, Monitors, MDSs
- Object Storage Device (OSD) - Physical or Logical Storage Unit
- Objects - how all data is stored by the OSD Daemons
- Ceph Monitor - Maintains the Cluster Map, and listens for server heartbeat
- Ceph Metadata Server (MDS) - manages the file system namespace, coordinates access
- Ceph Cluster Map - cluster topology used
- CRUSH - Algorithm for computing object storage locations

Keywords (cont.)

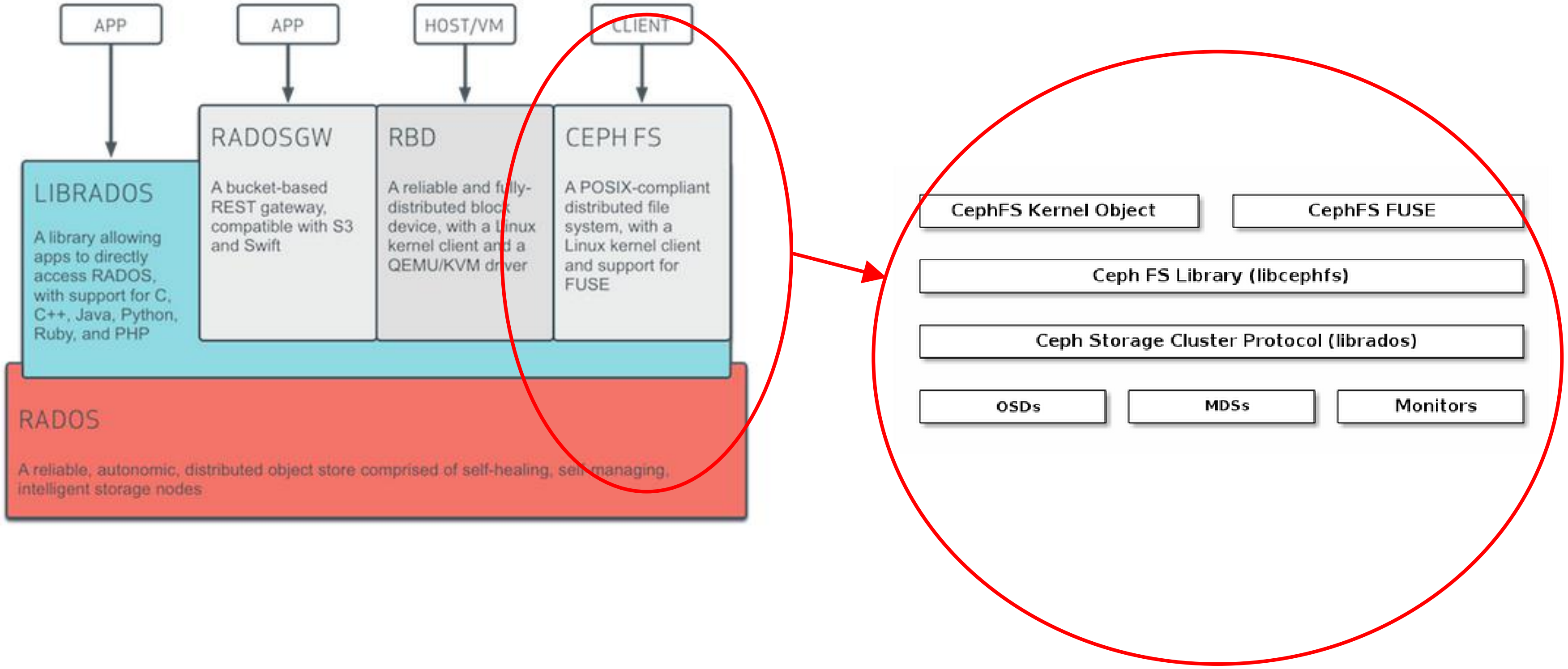
Pools - Logical partitions for storing objects

- Set Ownership and Access to objects
- Set the number of replicas of an object
- Set the number of Placement Groups available
- Set the CRUSH ruleset to be used

Placement Groups (PGs)

- Aggregates a series of objects into a single group
- Layer of abstraction to allow for highly scalable clusters

Architecture


APP
APP
HOST/VM
CLIENT
RADOSGW
A bucket-based REST gateway, compatible with S3 and Swift
RBD
A reliable and fully-distributed block device, with a Linux kernel client and a QEMU/KVM driver
CEPH FS
A POSIX-compliant distributed file system, with a Linux kernel client and support for FUSE
LIBRADOS
A library allowing apps to directly access RADOS, with support for C, C++, Java, Python, Ruby, and PHP
RADOS
A reliable, autonomic, distributed object store comprised of self-healing, self-managing, intelligent storage nodes
CephFS Kernel Object
CephFS FUSE
Ceph FS Library (libcephfs)
Ceph Storage Cluster Protocol (librados)
OSDs
MDSs
Monitors

http://ceph.com/

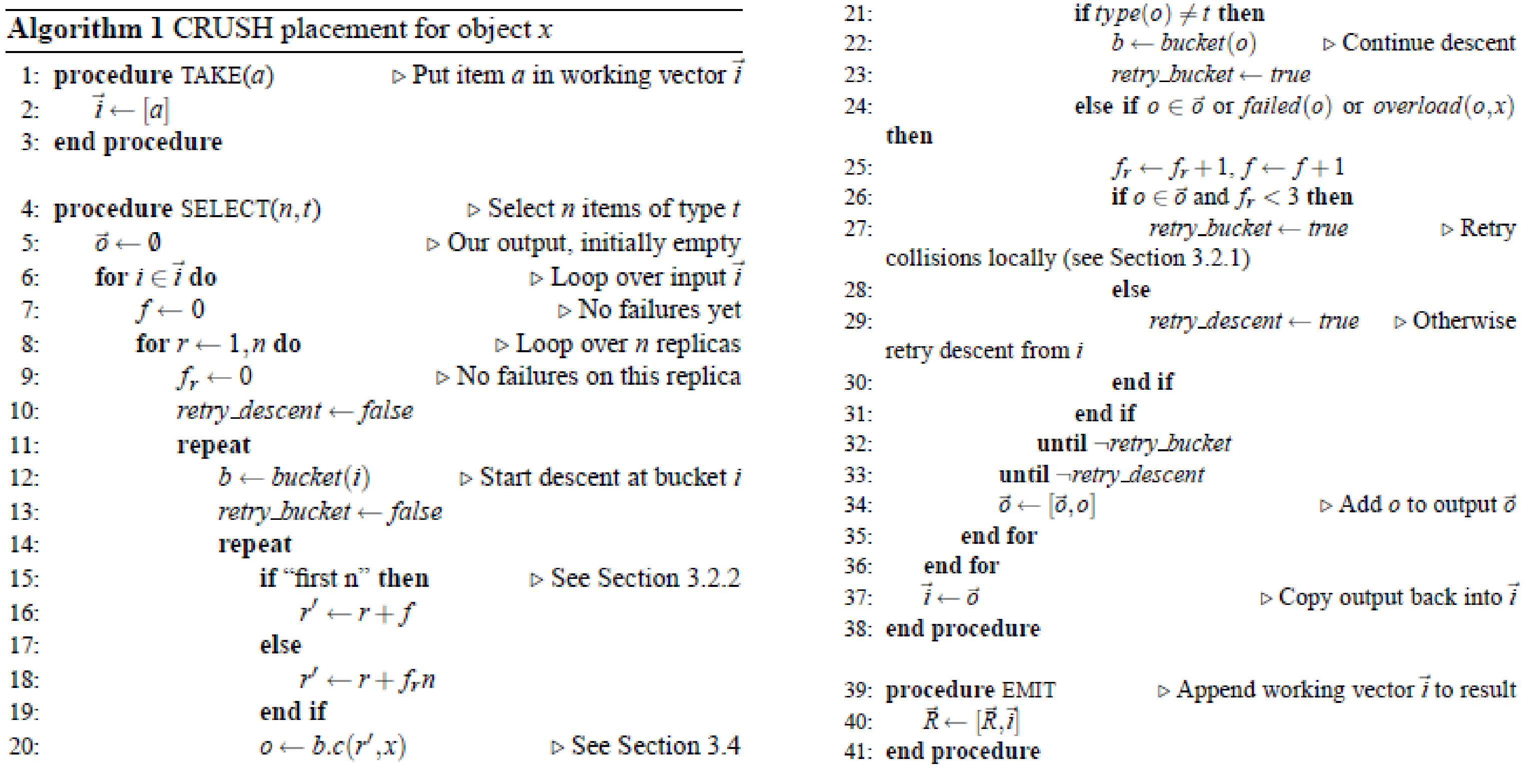
RADOS

- Storage service (Petabyte-scale)
- Scalable
  - Can scale to 1000x
- Reliable
  - Preserves consistent data access
- Allows nodes to self manage:
  - Replication
  - Failure detection
  - Recovery
- Use of a Cluster Map

CRUSH

- Distributes data objects among storage devices
- Data Distribution Policy - Placement Rules
  - How many replica targets are chosen
  - Restrictions on replica placement
    - (i.e. number of replicas in a device, host, rack, room …)
- Utilizes a strong multi-input integer hash function
  - mapping is completely deterministic and calculable
  - Given the cluster map, placement rules, and x (object)
  - Pseudo-random distribution across the Ceph Storage Cluster
    - The hashing function + inputs causes this

CRUSH Algorithm

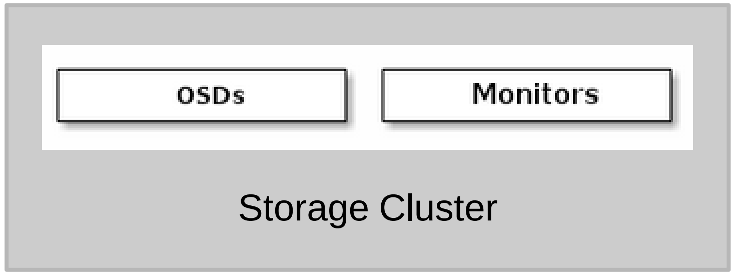


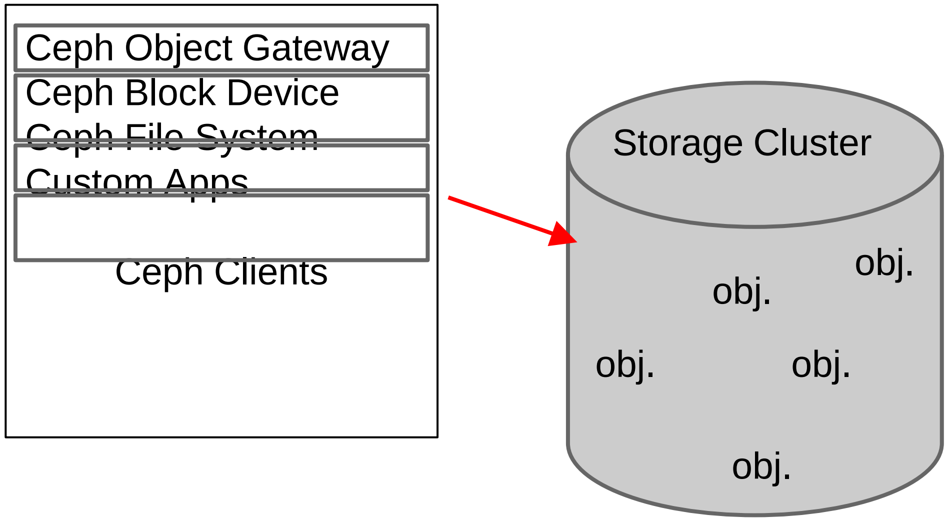
Algorithm 1 CRUSH placement for object x

```
procedure TAKE($a$)                                  ▷ Put item $a$ in working vector $\vec{i}$
    $\vec{i} \leftarrow [a]$
end procedure

procedure SELECT($n,t$)                              ▷ Select $n$ items of type $t$
    $\vec{o} \leftarrow \emptyset$                   ▷ Our output, initially empty
    for $i \in \vec{i}$ do                           ▷ Loop over input $\vec{i}$
        $f \leftarrow 0$                             ▷ No failures yet
        for $r \leftarrow 1,n$ do                    ▷ Loop over $n$ replicas
            $f_r \leftarrow 0$                       ▷ No failures on this replica
            $retry\_descent \leftarrow false$
            repeat
                $b \leftarrow bucket(i)$             ▷ Start descent at bucket $i$
                $retry\_bucket \leftarrow false$
                repeat
                    if "first n" then                ▷ See Section 3.2.2
                        $r' \leftarrow r + f$
                    else
                        $r' \leftarrow r + f_r n$
                    end if
                    $o \leftarrow b.c(r',x)$         ▷ See Section 3.4
                    if $type(o) \neq t$ then
                        $b \leftarrow bucket(o)$     ▷ Continue descent
                        $retry\_bucket \leftarrow true$
                    else if $o \in \vec{o}$ or $failed(o)$ or $overload(o,x)$ then
                        $f_r \leftarrow f_r + 1$, $f \leftarrow f + 1$
                        if $o \in \vec{o}$ and $f_r < 3$ then
                            $retry\_bucket \leftarrow true$      ▷ Retry collisions locally (see Section 3.2.1)
                        else
                            $retry\_descent \leftarrow true$     ▷ Otherwise retry descent from $i$
                        end if
                    end if
                until $\neg retry\_bucket$
            until $\neg retry\_descent$
            $\vec{o} \leftarrow [\vec{o}, o]$       ▷ Add $o$ to output $\vec{o}$
        end for
    end for
    $\vec{i} \leftarrow \vec{o}$                     ▷ Copy output back into $\vec{i}$
end procedure

procedure EMIT                                       ▷ Append working vector $\vec{i}$ to result
    $\vec{R} \leftarrow [\vec{R}, \vec{i}]$
end procedure
```

Ceph Storage Cluster

- Receives data from Ceph Clients
- All clients and Ceph OSD Daemons use CRUSH for file locations
- librados is provided as a native interface to the Ceph Storage Cluster
- Ceph OSD Daemons handle the read/write on the storage disks (OSDs)

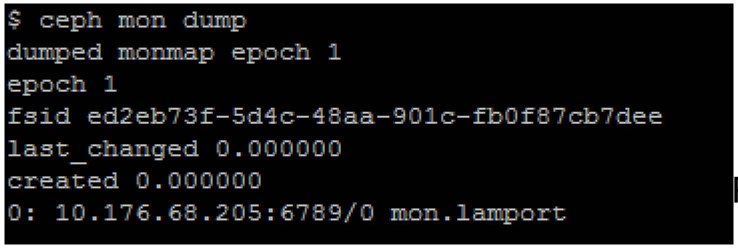




http://ceph.com/

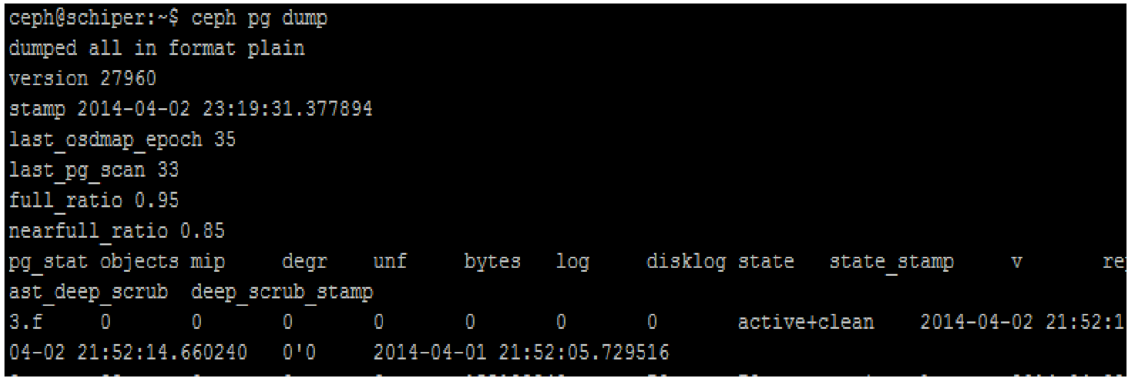
Cluster Map

- Used in conjunction with the CRUSH algorithm to calculate object location
- Cluster Topology (Collection of 5 maps)
  1. Monitor Map - Cluster fsid, Position, Name/Address and Port of each monitor



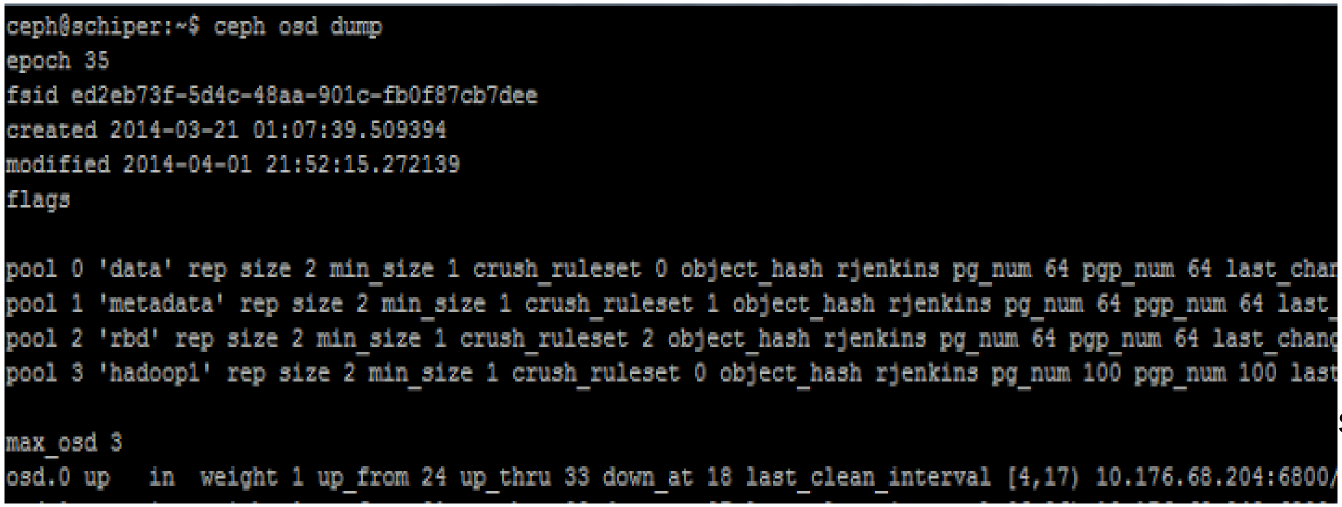
```
$ ceph mon dump
dumped monmap epoch 1
epoch 1
fsid ed2eb73f-5d4c-48aa-901c-fb0f87cb7dee
last_changed 0.000000
created 0.000000
0: 10.176.68.205:6789/0 mon.lamport
```

  1. PG Map - PG Full Ratios, PG details



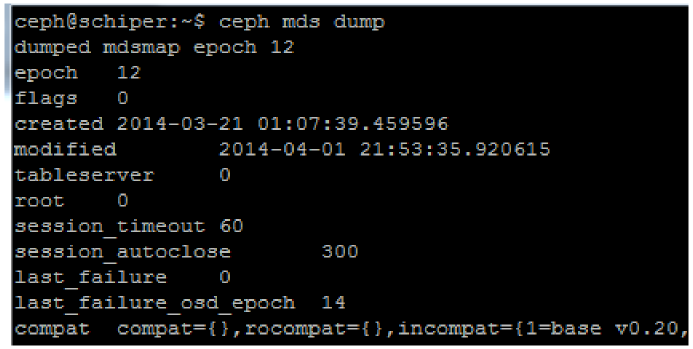
```
ceph@schiper:~$ ceph pg dump
dumped all in format plain
version 27960
stamp 2014-04-02 23:19:31.377894
last_osdmap_epoch 35
last_pg_scan 33
full_ratio 0.95
nearfull_ratio 0.85
pg_stat objects mip     degr    unf     bytes   log     disklog state   state_stamp     v       re
ast_deep_scrub  deep_scrub_stamp
3.f     0       0       0       0       0       0       0       active+clean    2014-04-02 21:52:1
04-02 21:52:14.660240   0'0     2014-04-01 21:52:05.729516
```

3. OSD Map - Cluster fsid, Create/Modify Dates, List of Pools, Replica Sizes, PG numbers, List of all OSDs and their status



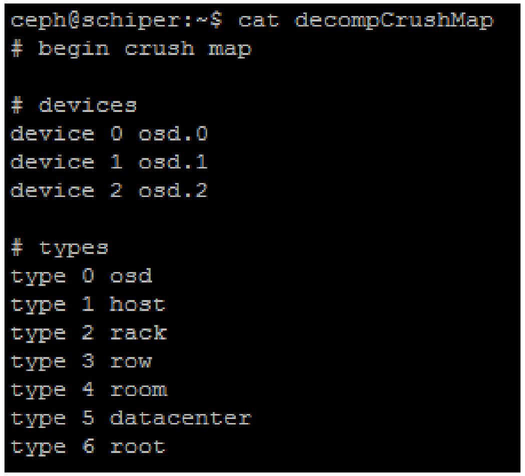
```
ceph@schiper:~$ ceph osd dump
epoch 35
fsid ed2eb73f-5d4c-48aa-901c-fb0f87cb7dee
created 2014-03-21 01:07:39.509394
modified 2014-04-01 21:52:15.272139
flags

pool 0 'data' rep size 2 min_size 1 crush_ruleset 0 object_hash rjenkins pg_num 64 pgp_num 64 last_char
pool 1 'metadata' rep size 2 min_size 1 crush_ruleset 1 object_hash rjenkins pg_num 64 pgp_num 64 last_
pool 2 'rbd' rep size 2 min_size 1 crush_ruleset 2 object_hash rjenkins pg_num 64 pgp_num 64 last_chang
pool 3 'hadoop1' rep size 2 min_size 1 crush_ruleset 0 object_hash rjenkins pg_num 100 pgp_num 100 last

max_osd 3
osd.0 up   in  weight 1 up_from 24 up_thru 33 down_at 18 last_clean_interval [4,17) 10.176.68.204:6800/
```

3. st of

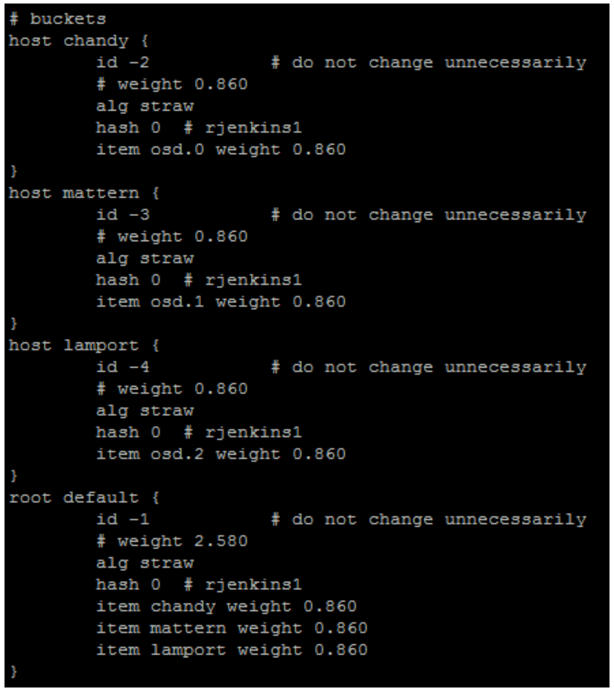


```
ceph@schiper:~$ ceph mds dump
dumped mdsmap epoch 12
epoch   12
flags   0
created 2014-03-21 01:07:39.459596
modified        2014-04-01 21:53:35.920615
tableserver     0
root    0
session_timeout 60
session_autoclose       300
last_failure    0
last_failure_osd_epoch  14
compat  compat={},rocompat={},incompat={1=base v0.20,
```

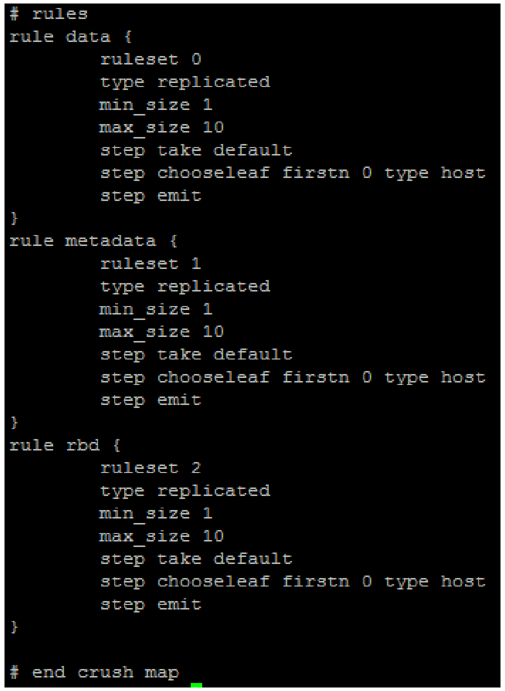
5. CRUSH Map - List of OSDs, the Failure Domain Hierarchy (i.e. device, host, rack, room …), rules for traversing the hierarchy



```
ceph@schiper:~$ cat decompCrushMap
# begin crush map

# devices
device 0 osd.0
device 1 osd.1
device 2 osd.2

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



```
# buckets
host chandy {
        id -2           # do not change unnecessarily
        # weight 0.860
        alg straw
        hash 0  # rjenkins1
        item osd.0 weight 0.860
}
host mattern {
        id -3           # do not change unnecessarily
        # weight 0.860
        alg straw
        hash 0  # rjenkins1
        item osd.1 weight 0.860
}
host lamport {
        id -4           # do not change unnecessarily
        # weight 0.860
        alg straw
        hash 0  # rjenkins1
        item osd.2 weight 0.860
}
root default {
        id -1           # do not change unnecessarily
        # weight 2.580
        alg straw
        hash 0  # rjenkins1
        item chandy weight 0.860
        item mattern weight 0.860
        item lamport weight 0.860
}
```

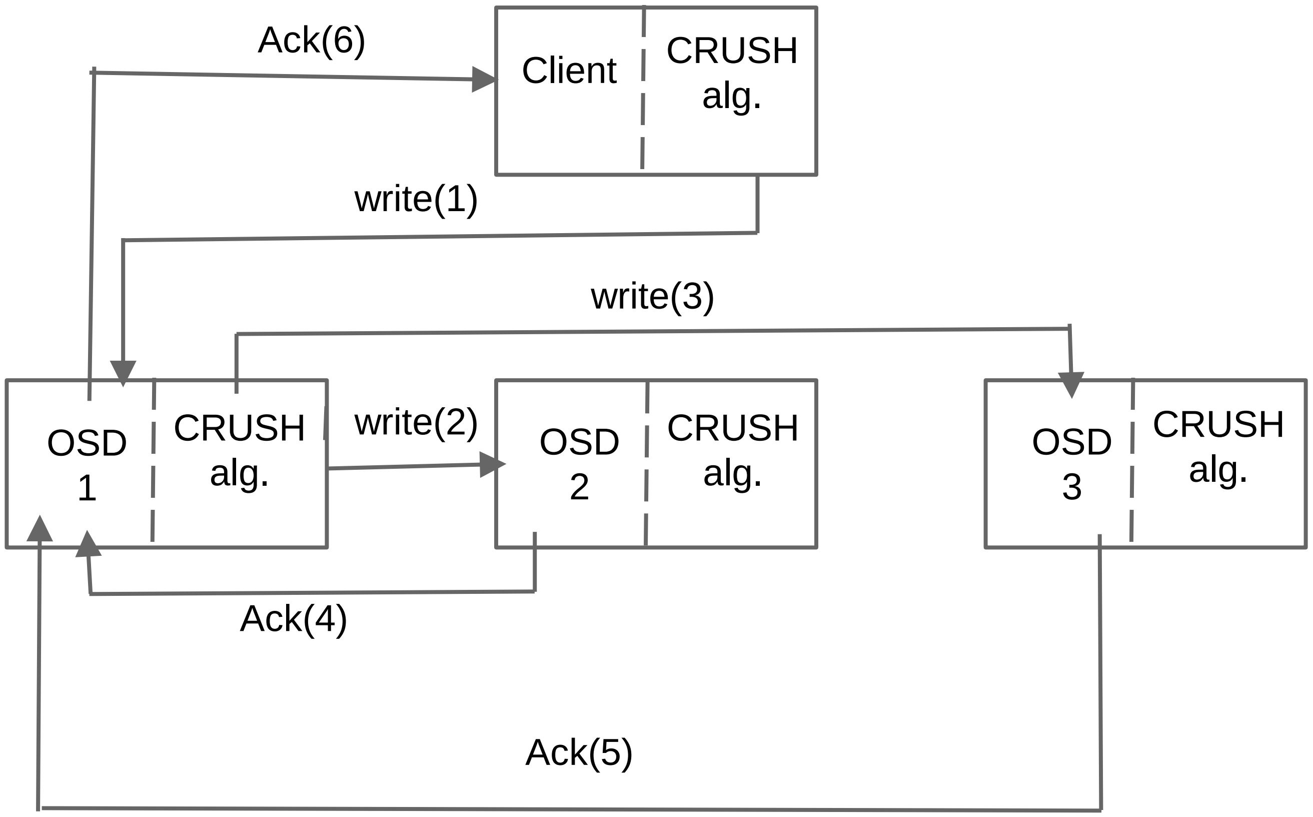


```
# rules
rule data {
        ruleset 0
        type replicated
        min_size 1
        max_size 10
        step take default
        step chooseleaf firstn 0 type host
        step emit
}
rule metadata {
        ruleset 1
        type replicated
        min_size 1
        max_size 10
        step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

Cluster Aware

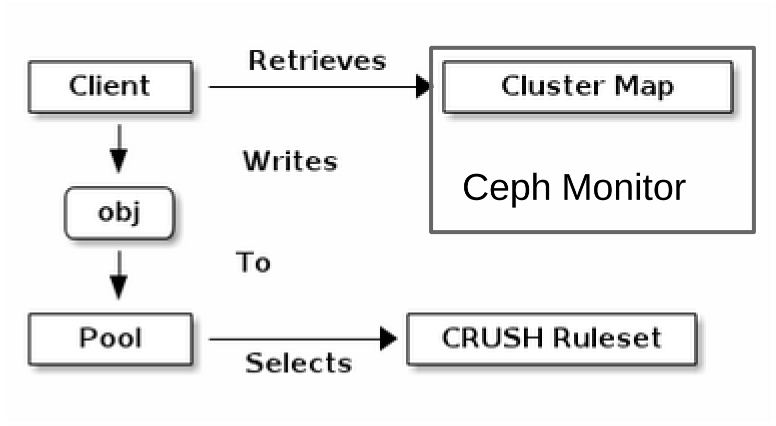
- Clients and OSDs know of all the OSDs (known cluster topology)
- Benefits:
    1. OSDs service clients directly
        a. No single point of failure (unlike traditional centralized unit)
    2. OSD reports status to neighbors and monitors
        a. Failures are detected quickly and reflected in the Cluster Map
        b. Monitors remain lightweight
    3. Data Scrubbing
        a. Daily data comparison across replicas to find bugs or FS errors
        b. Weekly deep data scrubbing - compare replicas bit-by-bit
            i. can determine bad sectors on the disk
    4. Replication
        a. OSDs use CRUSH algorithm to determine replica locations

Typical Write Protocol



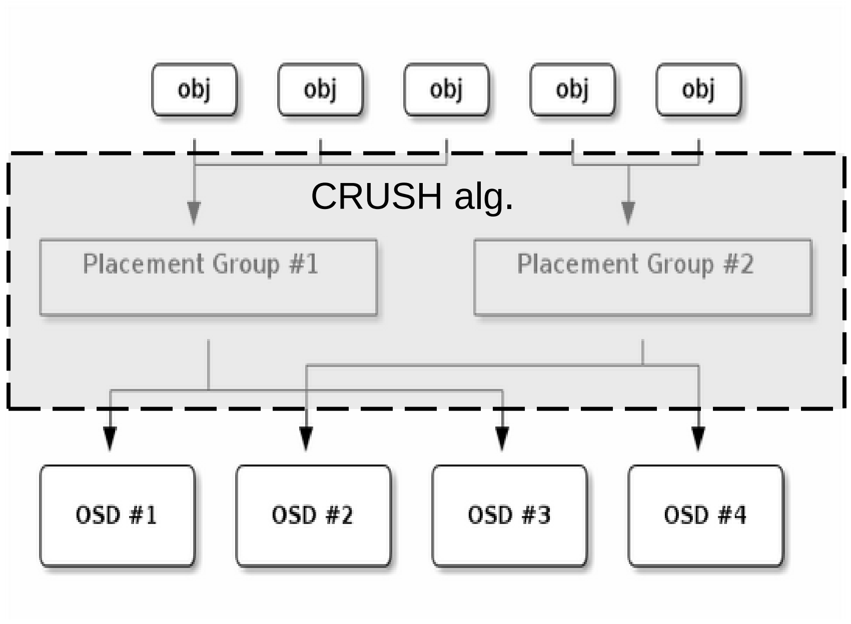
Dynamic Cluster Management

1. Ceph Clients retrieve a current Cluster Map upon binding to a Ceph Monitor
2. Ceph Clients then write objects to pools.
3. Data placement is then determined based on:
   a. Replica count
   b. CRUSH ruleset
   c. Count of Placement Groups (PGs)



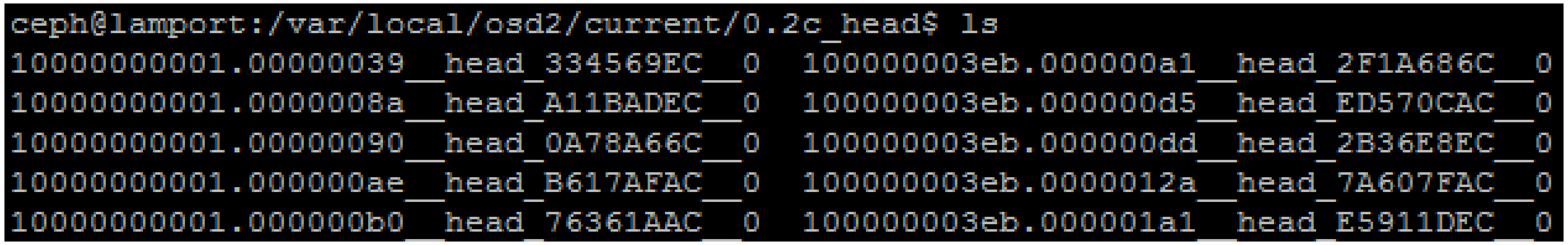
Dynamic Cluster Management

- Pools have multiple PGs
- PGs decouple Ceph Clients from OSDs
    - Allows for dynamic scaling and rebalancing
- Cluster Map + CRUSH alg.then client can compute OSD for R/W
    - All object locations are calculated



File Location Calculation

1. Ceph Client binds to Ceph Monitor, retrieves Cluster Map
2. Client inputs - Object ID and Pool
3. CRUSH alg. hashes the Object ID
4. CRUSH Calculates hash modulo (%) OSD count for the PG ID
5. CRUSH gets the Pool ID given the Pool Name (i.e. Pool1)
6. CRUSH prepends Pool ID to PG ID



```
ceph@lamport:/var/local/osd2/current/0.2c_head$ ls
10000000001.00000039__head_334569EC__0  100000003eb.000000a1__head_2F1A686C__0
10000000001.0000008a__head_A11BADEC__0  100000003eb.000000d5__head_ED570CAC__0
10000000001.00000090__head_0A78A66C__0  100000003eb.000000dd__head_2B36E8EC__0
10000000001.000000ae__head_B617AFAC__0  100000003eb.0000012a__head_7A607FAC__0
10000000001.000000b0__head_76361AAC__0  100000003eb.000001a1__head_E5911DEC__0
```

Summary

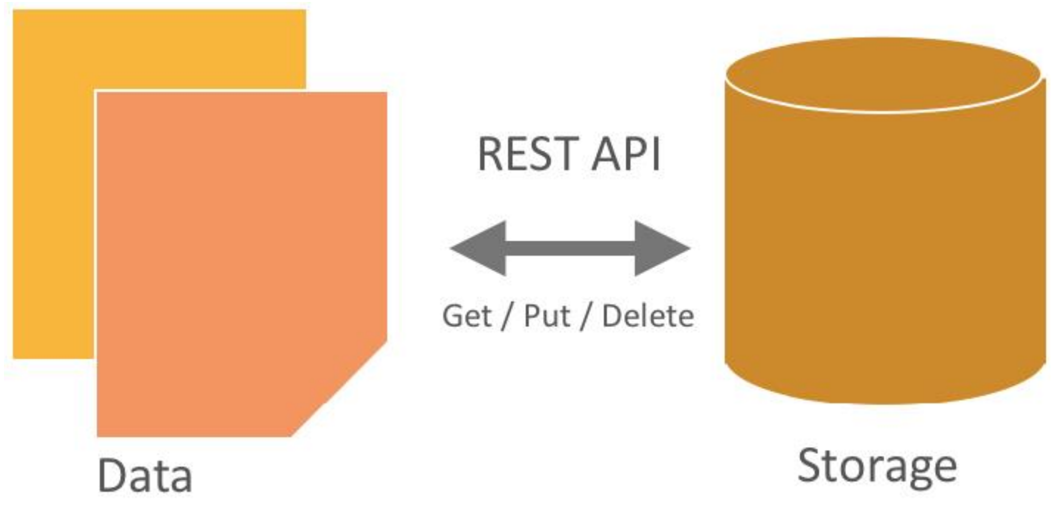
- Reliable/Fault Tolerant
- Self Healing
- Distributed

Working on …

- Testing Ceph compared with HDFS
- Configuring Hadoop to use Ceph FS
- Adjusting Ceph configurations for better performance

Openstack-Swift

- Highly available, distributed, eventually consistent object/blob storage
- RESTful requests sent to Proxy Server
- Proxy Server routes requests to appropriate ring location



	Ceph	**Swift**	**HDFS**
Consistency	Strongly consistent	Eventually consistent	Eventually consistent
Storage	●Block storage, Object storage	Object storage	Object storage
Data placement method	CRUSH (algorithm)	Ring (static mapping structure)	Namenode specifies the location of the data
Latency	No centralized metadata server reducing access latency	centralized metadata server, reason for latency	centralized metadata server, reason for latency
Components	object server, monitor server, RADOS	Account, Container, Object	Namenodes, Datanodes
Replica management	yes(dynamic)	No	No

More Differences

HDFS uses a central system to maintain file metadata (Namenode), where as in Swift the metadata is distributed and replicated across the cluster. Having a central meta-data system is a single point of failure for HDFS, and makes it more difficult to scale to very large sizes. Same comparison holds good for Ceph versus HDFS.

Swift is designed with multi-tenancy in mind, where HDFS has no notion of multi-tenancy

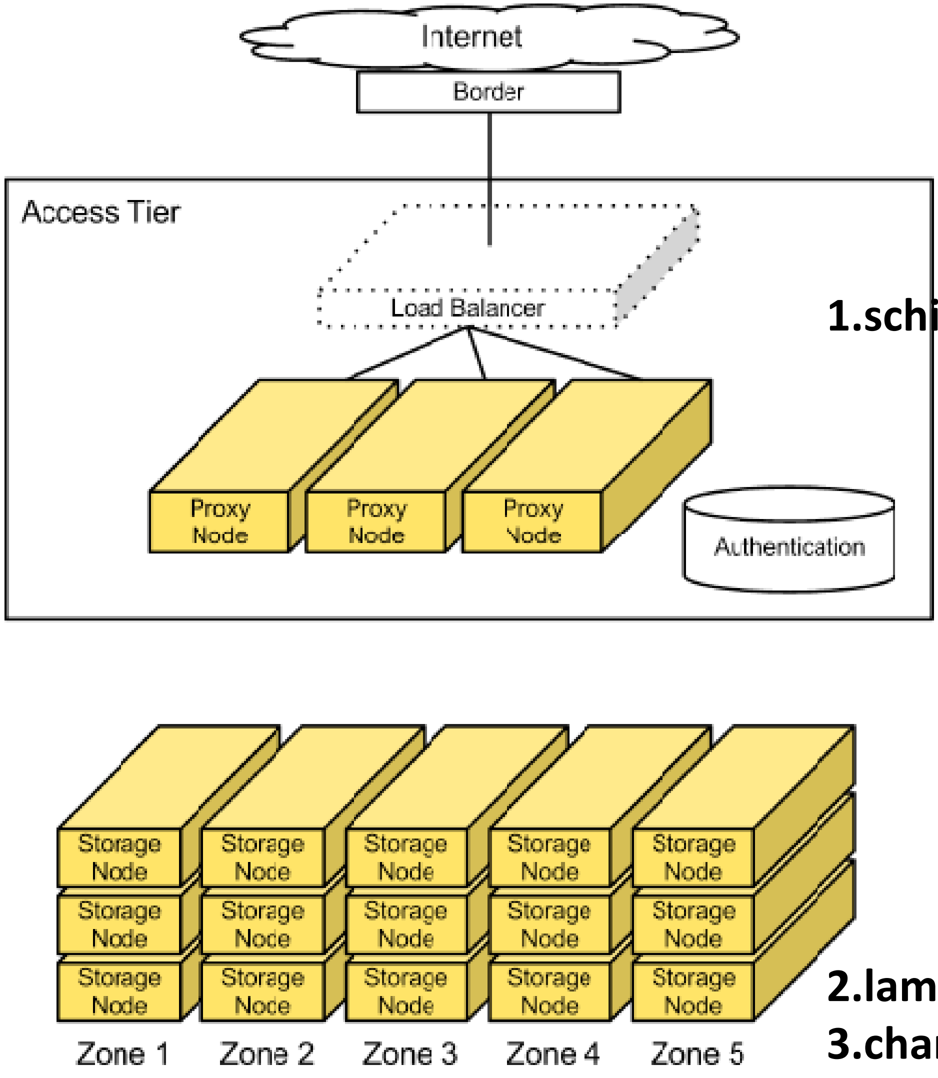
HDFS is optimized for larger files (as is typical for processing data), where Swift is designed to store any sized files.

Files in HDFS are write once, and can only have one writer at a time, in Swift files can be written many times, and under concurrency, the last write wins.

OpenStack Swift Cluster Architecture - Access Tier :

Building Blocks :

Proxy Servers: Handles all incoming API requests.
Rings: Maps logical names of data to locations on particular disks.
Zones: Each Zone isolates data from other Zones.
Accounts & Containers: Each Account and Container are individual databases that are distributed across the cluster.
Objects: The data itself.
Partitions: A Partition stores Objects, Account databases and Container databases.



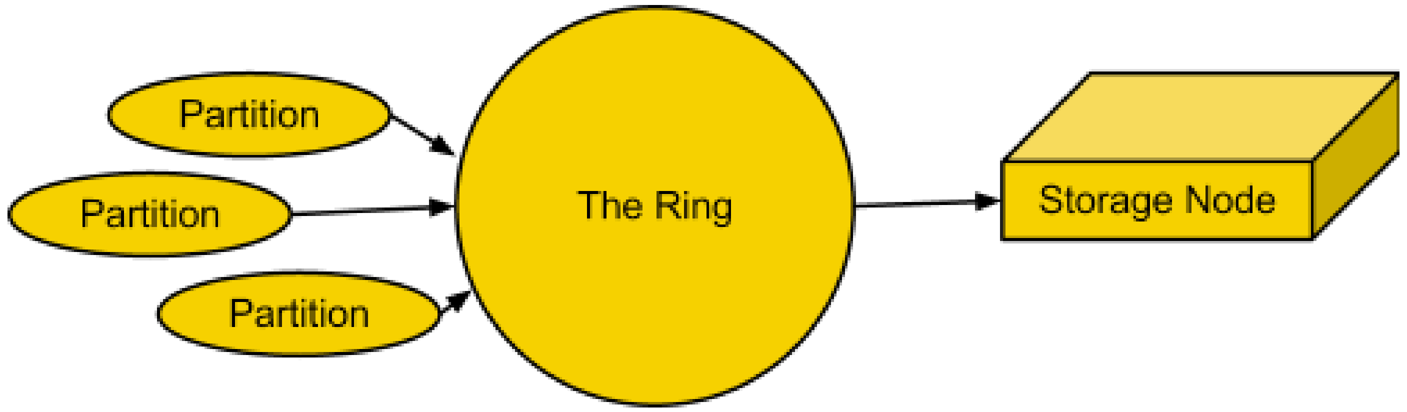
1.schiper

2.lamport
3.chandy
4.mattern

SWIFT USAGE

- Used in small deployments for “just” storing VM images, to mission critical storage clusters for high-volume websites, to mobile application development, custom file-sharing applications, data analytics and private storage infrastructure-as-a-service.
- http://swift.example.com/v1/account/container/object

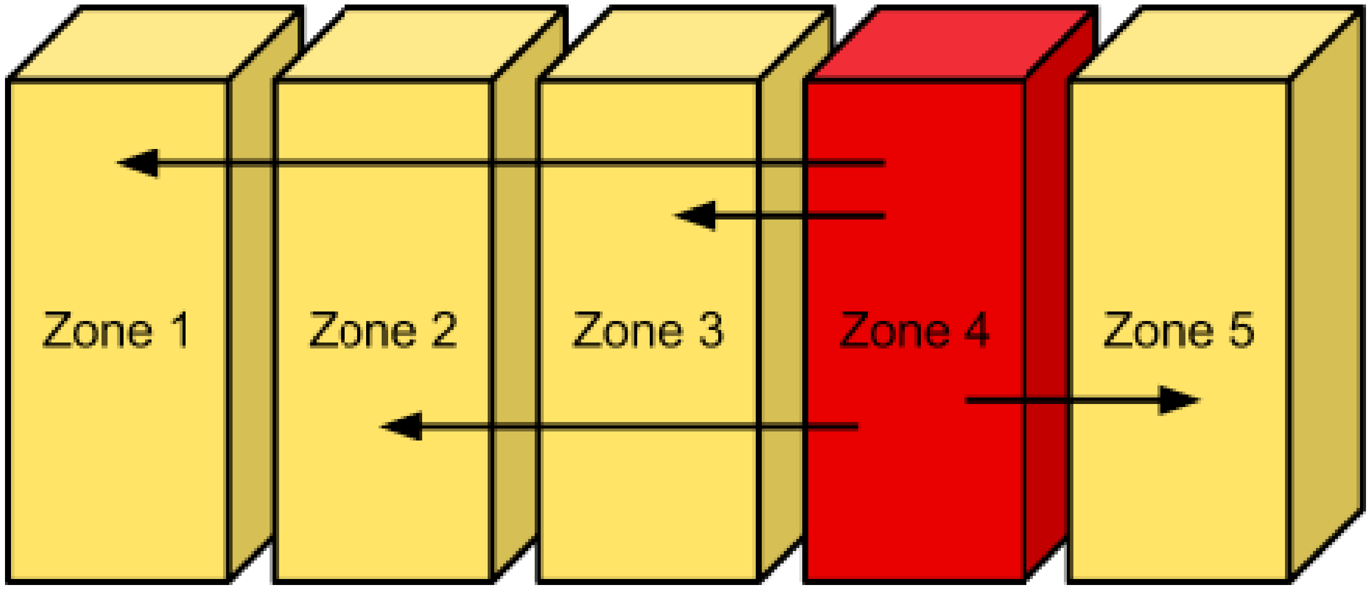
THE RING



The Ring maps partitions to physical locations on disk.

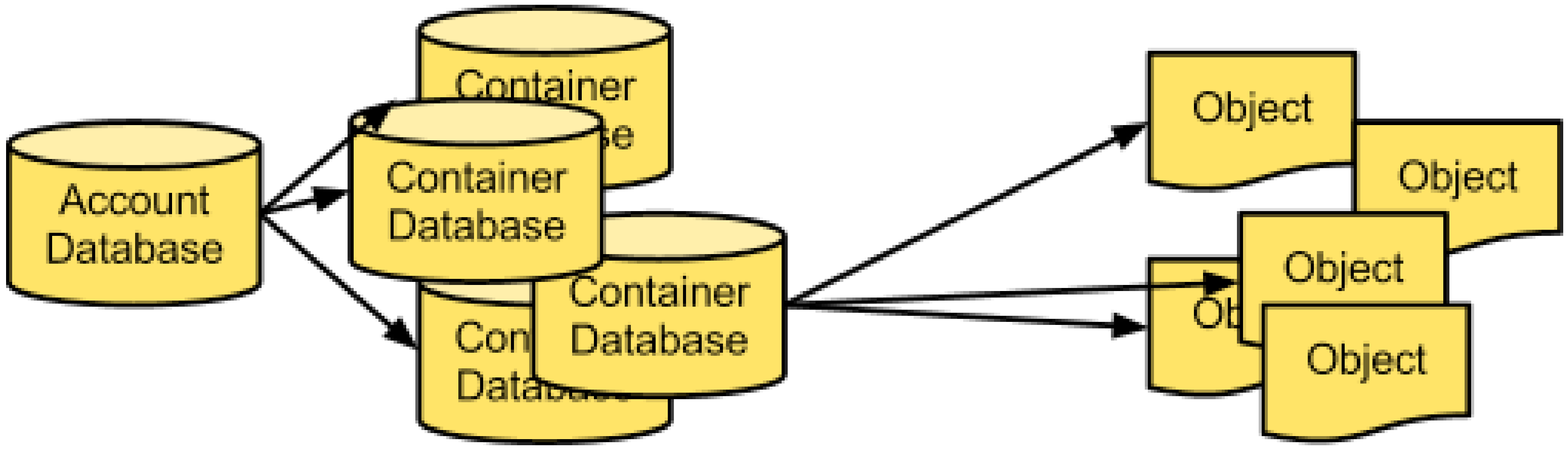
Ring is also responsible for determining which devices are used for handoff should a failure occur.

ZONES : Failure Boundaries



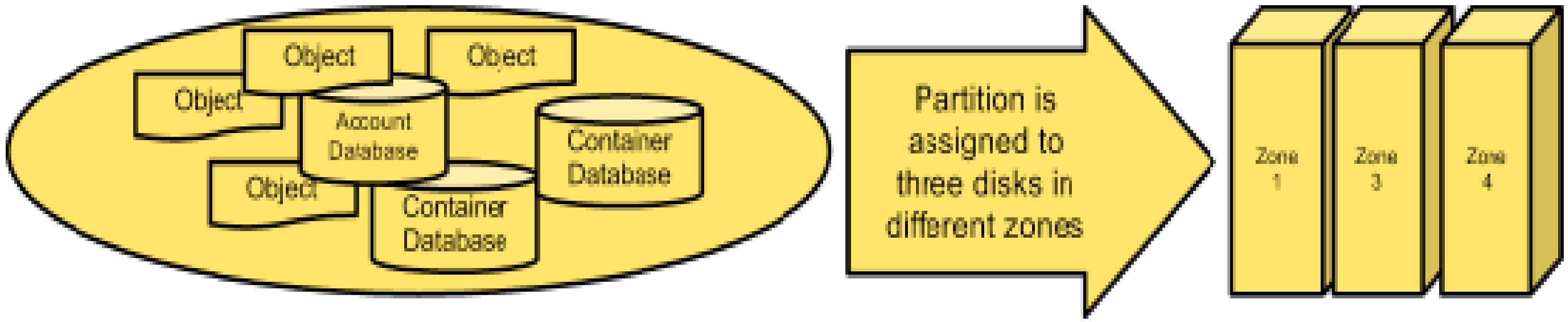
When a disk fails, replica data is automatically distributed to the other zones to ensure there are three copies of the data

Accounts & Containers



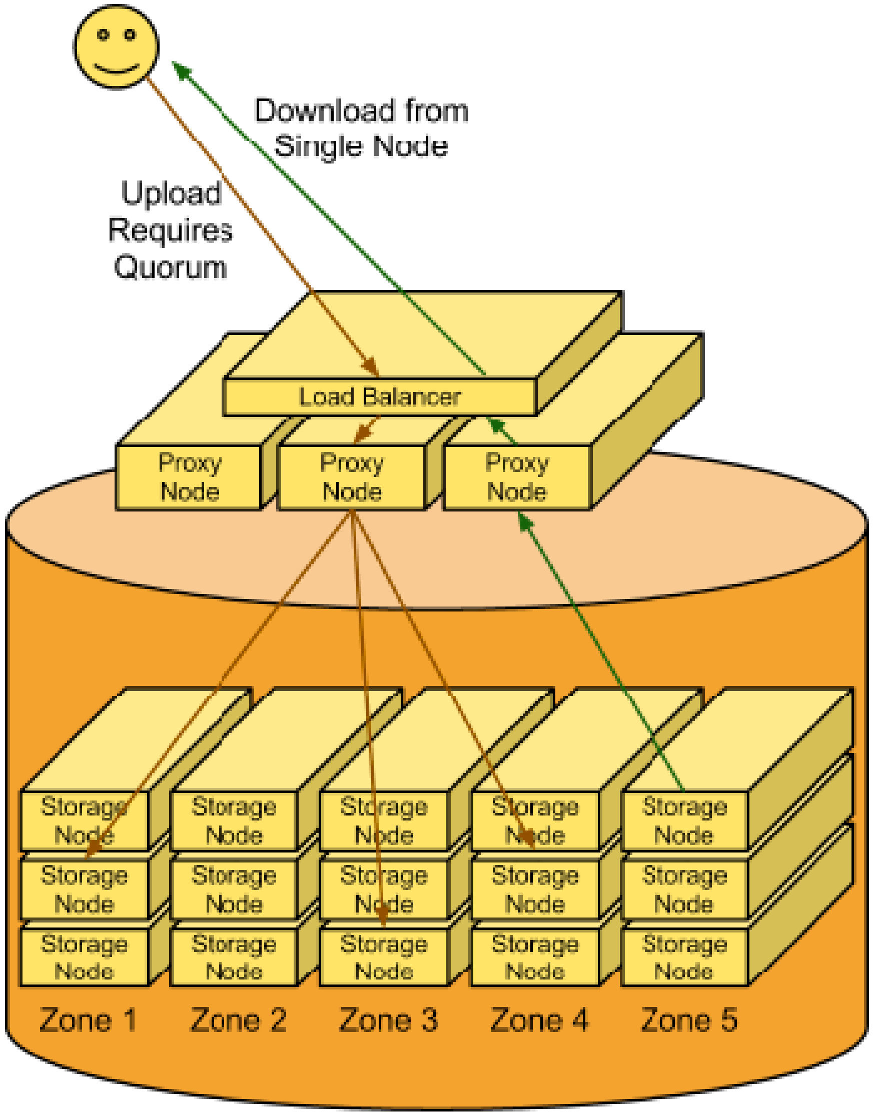
To keep track of object data location, each account in the system has a database that references all its containers, and each container database references each object

Partitions



a partition is just a directory sitting on a disk with a corresponding hash table of what it contains.

Upload



Make a HTTP request to PUT an object

Receive the request.

Figure out where the data is going to go. To do this, the Account name, Container name and Object name are all used to determine the Partition where this object should live.

Then a lookup in the Ring figures out which storage nodes contain the Partitions in question.

The data then is sent to each storage node where it is placed in the appropriate Partition.

Upload

A quorum is required -- at least two of the three writes must be successful before the client is notified that the upload was successful.

Next, the Container database is updated asynchronously to reflect that there is a new object in it.

Download

A request comes in for an Account/Container/object. Using the same consistent hashing, the Partition name is generated.

A lookup in the Ring reveals which storage nodes contain that Partition.

A request is made to one of the storage nodes to fetch the object and if that fails, requests are made to the other nodes.

Benefits of Swift

Feature	Benefit
HDD/node failure agnostic	Self healing Reliability, data redundancy protecting from failures
Unlimited storage	Huge & flat namespace, highly scalable read/write access Ability to serve content directly from storage system
Multi-dimensional scalability (scale out architecture)	Backup and archive large amounts of data with linear performance
Account/Container/Object structure	Optimized for scale Scales to multiple petabytes, billions of objects
Built-in replication 3x+ data redundancy compared to 2x on RAID	Configurable number of accounts, container and object copies for high availability

Benefits of Swift (continued)

Easily add capacity unlike RAID resize	Elastic data scaling with ease
No central database	Higher performance, no bottlenecks
RAID not required	Handle lots of small, random reads and writes efficiently
Built-in management utilities	Account Management: Create, add, verify, delete users Container Management: Upload, download, verify Monitoring: Capacity, host, network, log trawling, cluster health

Object Server

- Blob storage server
- Stores, retrieves, deletes objects on local devices
- Objects stored as binary files
- Last write wins

Deletion

- Treated as a separate version of file
- 0 byte file ending in .ts (tombstone)
- Ensures objects are actually deleted

Container Server

- Handles listings of objects in sqlite database files
- Listings are replicated across the cluster
- Tracks number of objects and storage usage for a container

Account Server

- Similar to the Container Server, but is responsible for listings of containers rather than objects.
- The Account Reaper handles account deletion and deletion of all objects for an account.

Replication

- Keeps system in consistent state during temporary error conditions

- Compares local data with remote copy to ensure latest version kept

- Keeps list of hashes about files; pushes changes when hash changes.

Updaters

- Process failed updates
- Consistency Window is the same as the Updater frequency

Auditors

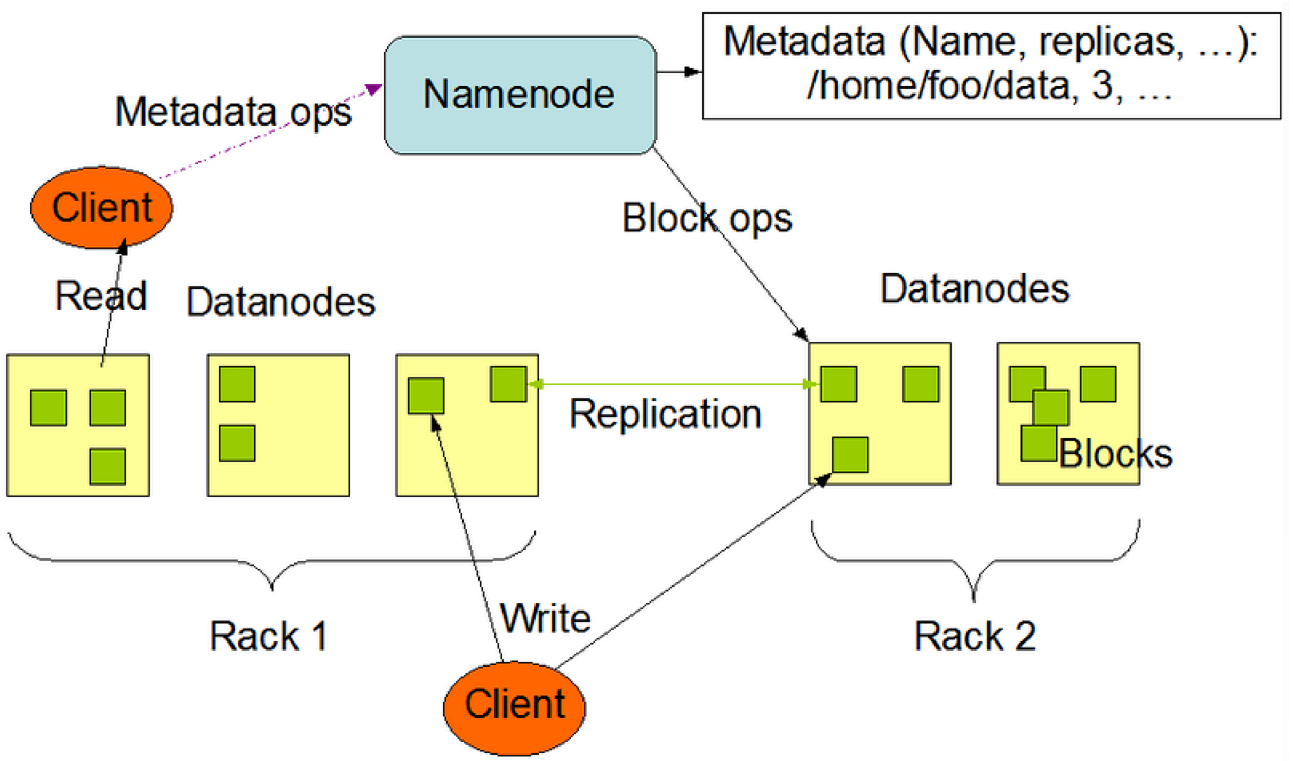
- Check integrity of objects, containers, accounts on local server
- Quarantines corrupted files
- Replaces quarantined files with correct files from other replicas

hadoop

HDFS Is ...

- Open Sourced
  - git://git.apache.org/hadoop-hdfs.git
- Distributed File System
- Provides high throughput access to application data
- Highly fault-tolerant
- For applications which have large data sets, typically GBs to TBs in size
- A write-once-read-many framework (ensures data coherence)
  - files should not be changed after being closed
- Designed for reliability of very large files, by storing files as replicated block sequences.

Architecture



http://hadoop.apache.org/docs/current/hadoop-project-dist/hadoop-hdfs/HdfsDesign.html

Failures

- Heartbeat messages are used to determine Disk Failures by the NameNode (NN)
- NN flags any Data Node (DN) with an absent heartbeat as dead
- The data on the DN is then unavailable moving forward
- The NN will initiate replication if a dead DN causes the replication count to drop for a (or many) block(s)
- Data Integrity is guaranteed through the use of checksums on the data

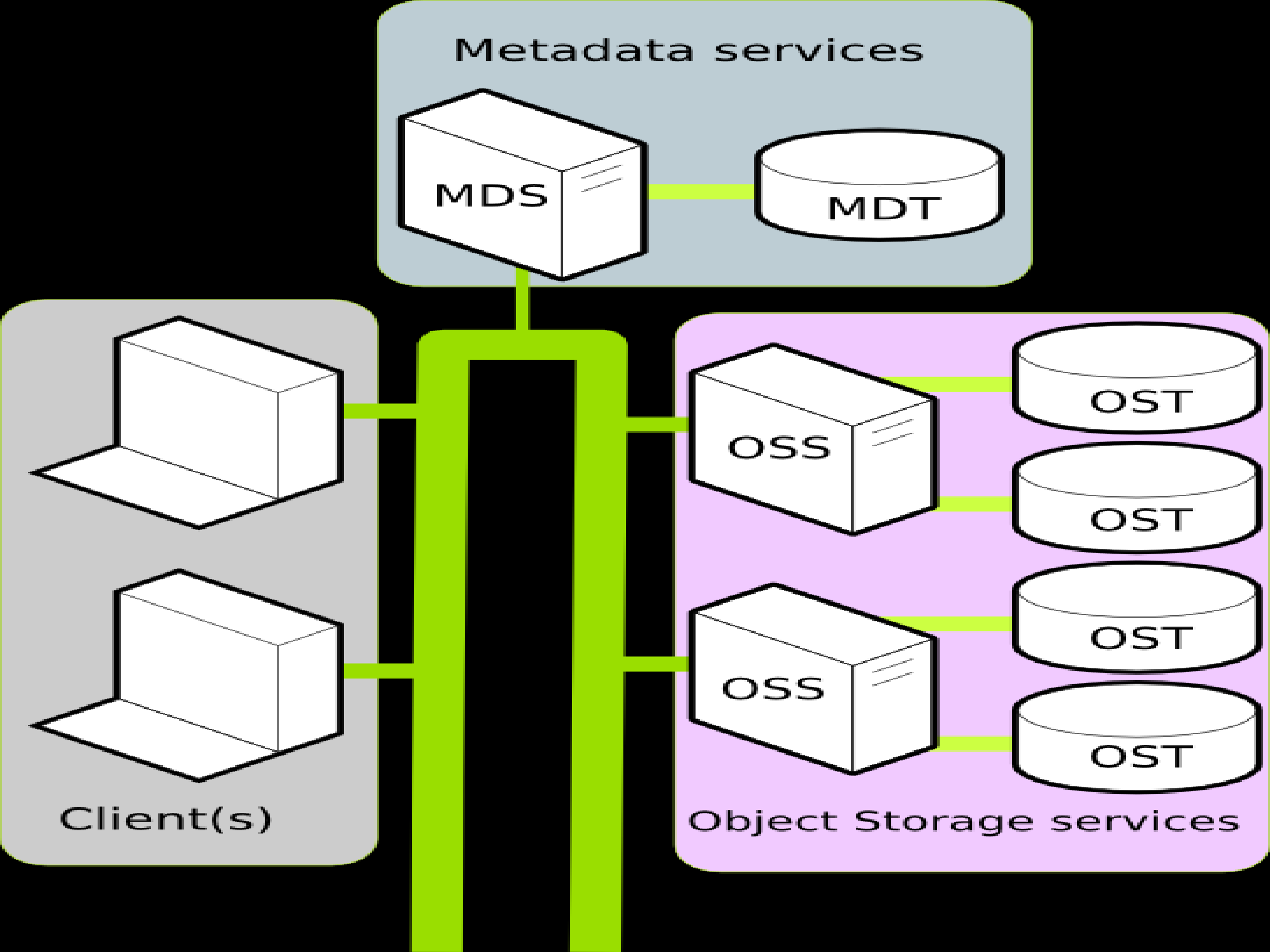
Lustre

Introduction

- Lustre is a POSIX-compliant global, distributed, parallel file system.
  - POSIX Compliant
    - Appears to the client the same way a local disk or NFS volume would
  - Global
    - A single namespace across multiple storage servers/volumes
    - All clients see a single volume regardless of the back-end
  - Distributed
    - Files are distributed across storage servers for load balancing
  - Parallel
    - A single file can be distributed across one or more servers

Lustre Architecture: Components

- File system clients, which can be used to access the file system
- Object storage servers (OSS), which provide file I/O service
- Metadata servers (MDS), which manage the names and directories in the file system


Metadata services
MDS
MDT
Client(s)
OSS
OST
OST
OSS
OST
OST
Object Storage services

OSS (Object storage servers)

- Analogous to the head node for each storage server,1 or more OSS per cluster Performs the disk I/O when prompted by client
- Each OSS can be responsible for multiple object storage targets (OSTs), one for each volume
- OSS typically serves between 2 and 25 targets,each target up to 8 terabytes(TB) in size. The capacity of a Lustre file system is the sum of the capacities provided by the targets
- Here we use RAID 5 or RAID 6 striping patterns
- OSS provides I/O access pattern, which typically involves large data transfers

MSD (Metadata servers)

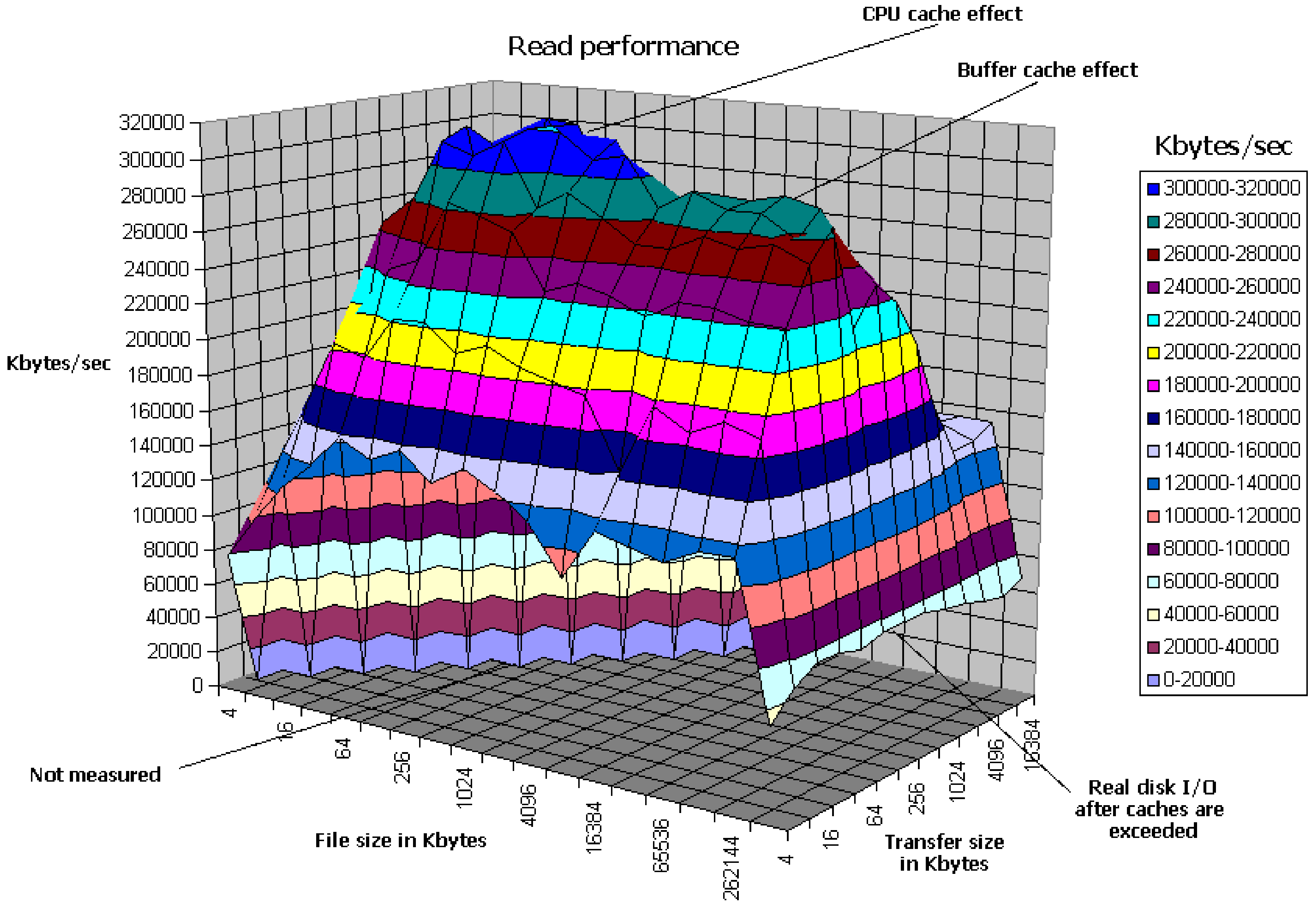
- Manages filesystem metadata, but stores no actual data
- Only 1 MDS is supported, 2 for Active-Passive
- Clustered Metadata is in the works
- Ideally, enough RAM to fit all of metadata in memory
- MSD is metadata access pattern with many seeks and read-and-write operations of small amounts of data
- Metadata Target (MDT)
  - Disk back-end to the MDS
  - Ideally SAS or SSD RAID10 for lots of small files

Lustre Disadvantages

- Complexity
- Reliability is improving but not quite there
- No High Availability at the Lustre level
- Doesn’t work well with Ubuntu

IOzone

- Benchmark tool that generates a variety of file operations and measures performance of the file system
- Operations include the following: *read, write, re-read, re-write, read backwards, read strided, fread, fwrite, random read, pread, mmap, aio_read, aio_write*


Read performance
CPU cache effect
Buffer cache effect
Kbytes/sec
320000
300000
280000
260000
240000
220000
200000
180000
160000
140000
120000
100000
80000
60000
40000
20000
0
Kbytes/sec
300000-320000
280000-300000
260000-280000
240000-260000
220000-240000
200000-220000
180000-200000
160000-180000
140000-160000
120000-140000
100000-120000
80000-100000
60000-80000
40000-60000
20000-40000
0-20000
4
16
64
256
1024
4096
16384
65536
262144
File size in Kbytes
Not measured
4
16
64
256
1024
4096
16384
Transfer size in Kbytes
Real disk I/O after caches are exceeded

Features of IOzone

- ANSII C source
- POSIX async I/O
- Mmap() file I/O
- Normal file I/O
- Single stream measurement
- Multiple stream measurement
- Distributed fileserver measurements (Cluster)
- POSIX pthreads
- Multi-process measurement
- Excel importable output for graph generation
- Latency plots
- 64bit compatible source
- Large file compatible
- Stonewalling in throughput tests to eliminate straggler effects
- Processor cache size configurable
- Selectable measurements with fsync, O_SYNC

REST

- Defines a set of architectural principles used to design Web services that focus on a how a system’s resource states are addressed and transferred over HTTP/HTTPS by a wide range of clients written in different languages
- Widely used
- Explicit use of HTTP methods in a way that follows the protocol as defined by RFC 2616
- Mostly displaced SOAP- and WSDL-based interface design because it's a simpler style to use
- Which is better ? REST or SOAP

- Creates one-to-one mapping between CRUD operations and HTTP
- To create a resource on the server, use POST.
- To retrieve a resource, use GET.
- To change the state of a resource or to update it, use PUT.
- To remove or delete a resource, use DELETE.
- RESTful API is a flexible way to provide different kinds of applications with data formatted in a standard way
- helps to meet integration requirements

NETMIST

- Protocol independent (NFS,CIFS,Lustre, …)
- Portable across operating systems and platforms. Unix, and Windows
- POSIX compliant API usage
- Scalable from small to very large systems.
- Supports complex data and meta data workloads
- Supports customizable workloads (Commercial version)
- Provides results of Ops/sec and latency
- Multi-client testing
- Support IPv4 and IPv6 environments

NETMIST

- Support op rate controlled runs
- Supports physical & virtualized environments
- Support for simultaneous mixed workloads
- Ability to control client and server caching
- Support user defined op mixes as well as transfer size distributions. (Commercial version)
- Provide monitoring callouts for external vendor tools
- Unencumbered licensing for everything

set of operations that are measured.

- read()
- read_file()
- mmap_read()
- read_random()
- write()
- write_file()
- mmap_write()
- write_random()
- rmw()
- mkdir()
- unlink()
- append()

- lock()
- unlock()
- access()
- stat()
- chmod()
- readdir()
- statfs()
- copyfile()
- rename()
- pathconf()

Source

The User’s guide: – http://dl.dropbox.com/u/20400396/Netmist_Users_Guide.doc

The run rules guide: – http://dl.dropbox.com/u/20400396/Netmist_run_rules.doc

This power point kit: – http://dl.dropbox.com/u/20400396/Pub_Netmist_presentation.ppt