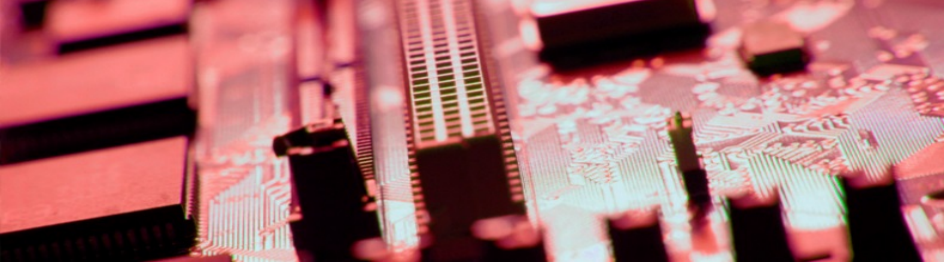


LCI workshop
on high performance clustered computing

Linux Clusters Institute: Ceph & Swift

Georgia Tech, August 15th – 18th 2017

Sam Linston
Utah Center for High Performance Computing
sam.linston@utah.edu

J.D. Maloney | Storage Engineer
National Center for Supercomputing Applications (NCSA)
malone12@illinois.edu

THE UNIVERSITY OF UTAH

NCSA

Introduction

- The rise of object storage is showing itself these past few years
- Driven by the massive growth of datasets and prominence of large web-scale companies that need to store obscene amounts of data
  - Object storage and Amazon S3 storage are closely recognized together for example
- Many options are out there for object storage, way more than just Ceph and Swift
  - These are the two most popular ones in the HPC space
- As research data sharing continues to grow and web portals to datasets are build, we need to build them on something

Ceph

Overview

- Object/Block/Posix file system that is owned by Red Hat
- It along side GlusterFS are in Red Hat Storage Server product
- Open Source, free to use, commercial support available through Red Hat
- Provides Object, Block, and Posix storage
- Very popular in the cloud/virtualization space
  - Can underpin things like Open Stack and Proxmox
- Runs on “commodity” hardware
- Can be deployed over Ethernet or IB
  - Ethernet is very popular with this FS

General High Level View

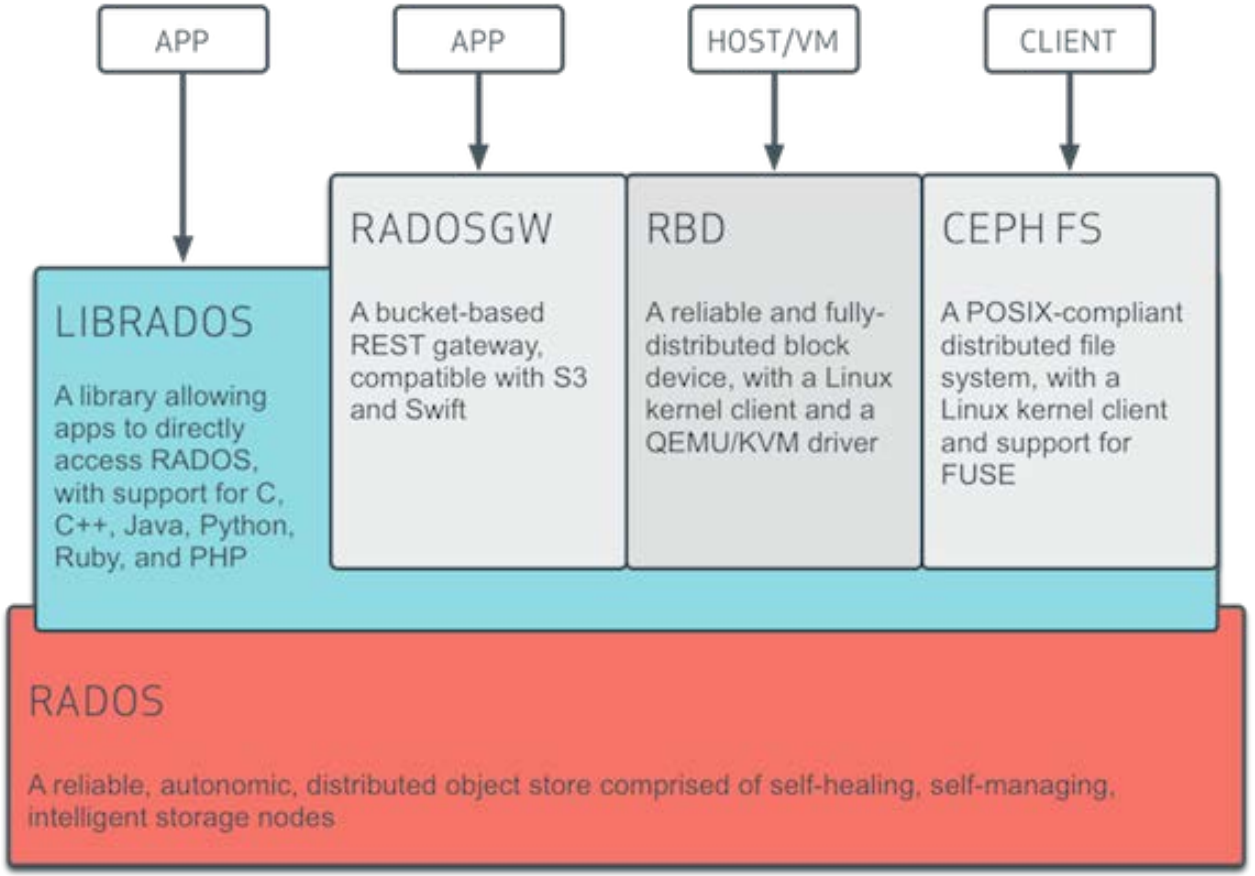


Image Credit: ceph.com

Deeper Architecture Views

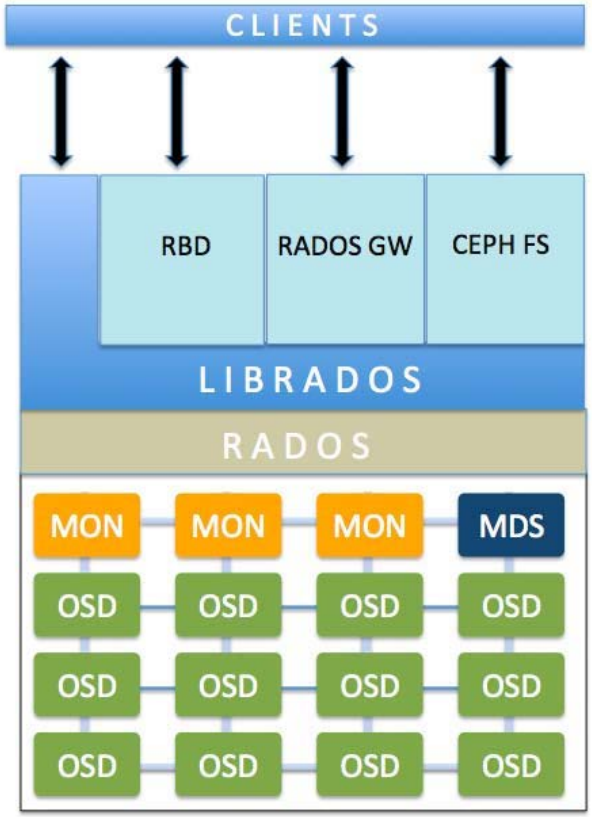


Image Credit: apprize.info

Deeper Architecture Views

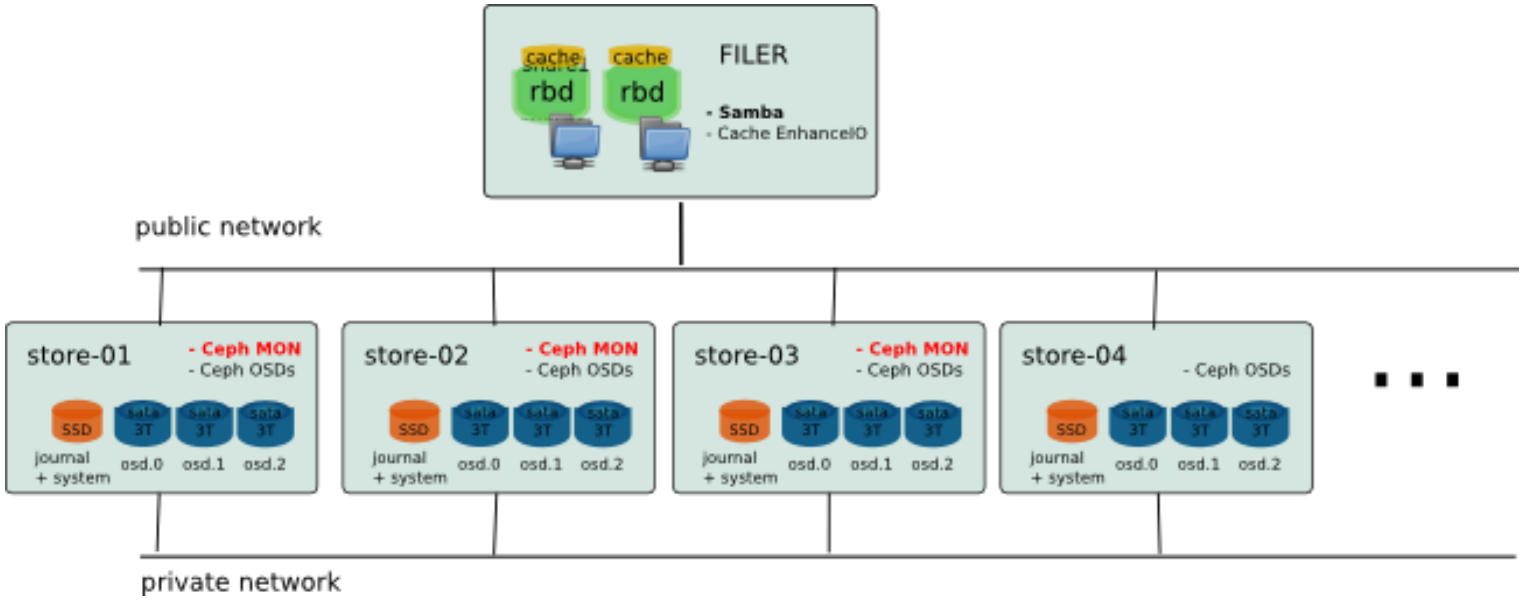


Image Credit: ksperis.com

A ”Jack of All Trades”

- Offers all three kinds of storage
- The LIBRADOS layer brokers all interfaces with the storage system for great flexibility
- Valuable in that one managed system can house everything
- Only mainstream file system besides Spectrum Scale that can support all three types
  - Many only provide POSIX, or block/object

Data Structure

Storage Blob

- The configuration of the MONs and OSDs result in a blob of storage
  - Organized mass of storage
- No structure
- This very loose structure is what allows Ceph to be the flexible and powerful solution it is

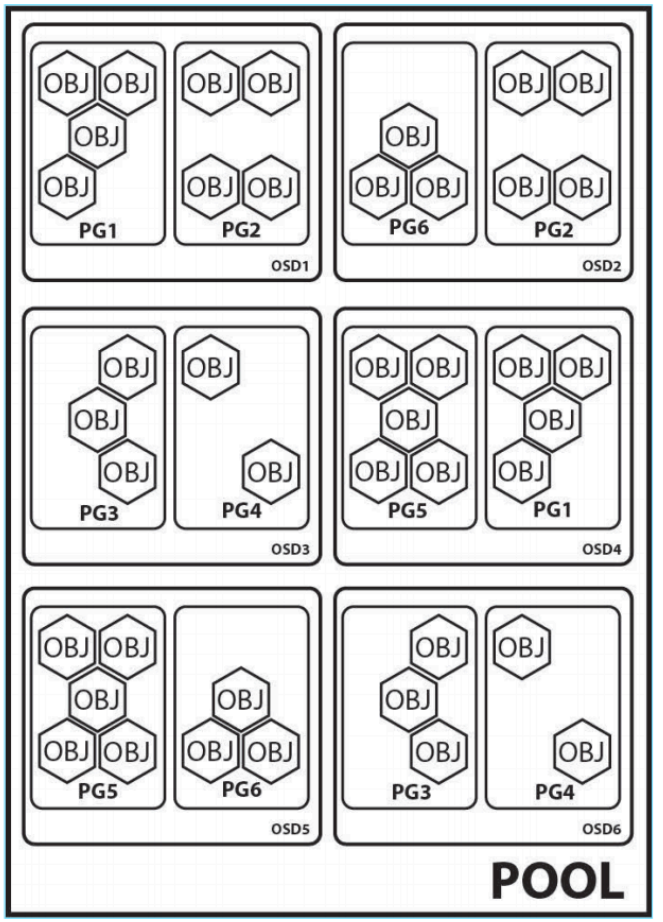


Image Credit: Sam Linston

Data Structure

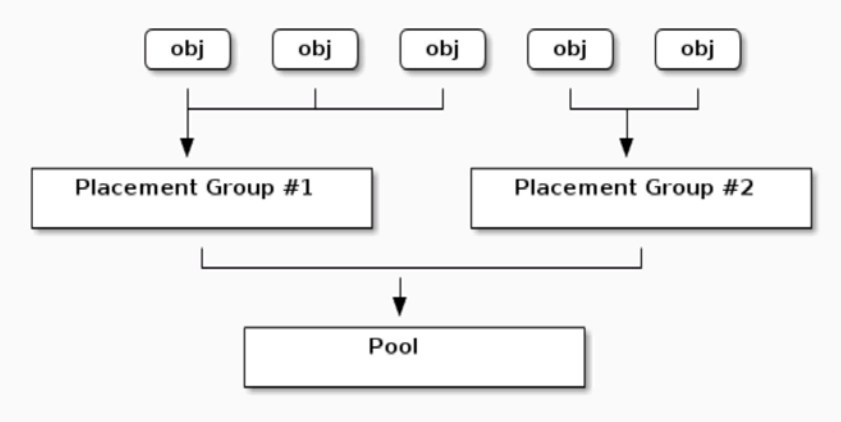
Pools

- A set of pools are created during system setup
- Several pools for user information and several other utility pools
- Pool for data payload
  - Pool is configured with desired level of resiliency
- Two resiliency methods
  - Can be N way replication (2x or 3x data copies)
  - Can be K+M fragment erasure coding (newer, but stable feature)
  - Desired endurance of data and other environment characteristics determine which type of data protection is best

Data Structure

Placement Groups

- Ensures proper, balanced object disbursement and configured redundancy
- Number of PGs in a pool determined by
  - Number of OSDs in cluster
  - Number of replicas or level of resiliency



Architecture Considerations

Failure Domains

- Set of rules defining segregation of replicated instances or distributed object fragments
  - Can be as coarse or granular as needed
  - Machine, rack, row, room, datacenter, etc
- Configure failure domains to minimize the chance of down time of the service
  - How this is done is mostly dependent on environment and resources you have available
- Something to consider and plan for before going into production

Architecture Considerations

I/O Storms

- What will the rebalance impact be if one server and its disks fails
- Chose smaller number of disks/server to reduce the intensity of the I/O storm
  - As small as 12 to 1
  - As large as 60 to 1
  - Depends on tier, server power and bandwidth
- Take rebuild situations into consideration when deciding on backend network infrastructure so that recoveries don't have a very noticeable impact on the quality of service

Architecture Considerations

Set a High Watermark

- Never good to push file systems to high levels of filled capacity, a high watermark should be determined
- Rough target of ~75%-80% full
- Must be room to rebalance to
  - Too full and the ability to be resilient decreases
- Once your system starts to approach your high watermark start working on adding capacity and rebalance the data across systems
- Consider this in your design, how are you going to add on in the future when growth occurs

Architecture Considerations

Performance

- Dedicated Journal devices (usually SSDs)
  - Ratio of Journal devices to non-journal devices
  - The more that share the same journal device, the more contention there is going to be for quick writes
- Network bandwidth on the backend
  - 10G/40G/100G Ethernet
  - Backbone between switches
- Different tiers of performance
  - Consider a fast tier with SSD journals, or even an all SSD tier
  - Lower tier with fewer/lesser performing journal devices or no journal devices at all

Ceph Deployment

Ceph-Deploy Tool

- Consolidates the deployment process under one command
- Originally Ceph was installed using a suite of tools
- Now the same tool is used for:
  - Installation of Ceph software
  - Commissioning the various Ceph components (ADM, MON, RGW, CephFS, RBD)
  - Prepare raw storage
  - Create OSDs
- Does not destroy OSDs
  - Convoluted process

Ceph Deployment

Other Tools

- Ceph Tool
  - Older tool
  - Still used to provision pools
    - Query System
- RADOSGW-ADMIN Tool
  - Used for user creation and manipulation
  - Set quota
  - Query usage
- Custom Scripts
  - Many deployment and management scripts available
  - See what the community has provided already, don't re-invent the wheel if you don't have to

Features Worth Highlighting

- Supports Thin Provisioning
  - Great for hosting virtual machine images
- Both the S3 Restful API and the Openstack Swift API are supported
- Erasure Encoding across disks
  - Much more efficient use of space than replication
- Snapshots and Clones
- Large and growing community that is there to support others getting into Ceph for the first time

Usage Patterns

Archive/Backup

- Very popular consumption pattern
- Can be User or Admin managed
- Space usage managed by quotas
  - One drawback in management — No groups
- Access through many tools
  - rclone, Globus, s3cmd, posix mount (if enabled)
- Great for dumping large tar.gz bundled datasets into

Usage Patterns

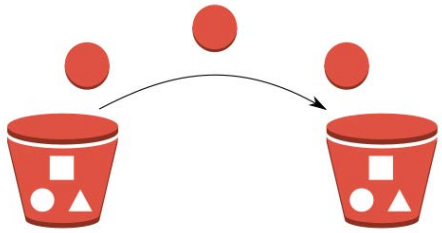
WEB

- Objects can be accessed via a URL
  - Similar to AWS
  - Resolved as https://<bucket_name>.hostname.domain.name
    - Or https://hostname.domain.name/<bucket_name>
- Not easily browsed
  - Space is inherently flat
  - All objects are peers
  - Expresses the page as raw XML
- Objects must be made “public” to be URL accessible
  - Access can be set to expire

Usage Patterns

S3 API

- Popular in the web application space, the leading standard that most object storage systems support (not just Ceph or Swift)
- Program directly to the API
  - Very scriptable, and is Restful
- Use other programmatic applications like Python Boto



User Experience

Buckets and Objects

- All object space is flat
  - No tree structure or hierarchy
- To help users navigate a tree structure is represented through tools
- Structure is represented as buckets (analogous to directories) and objects
- One user cannot by default see another users buckets/objects
  - Must be shared

User Experience

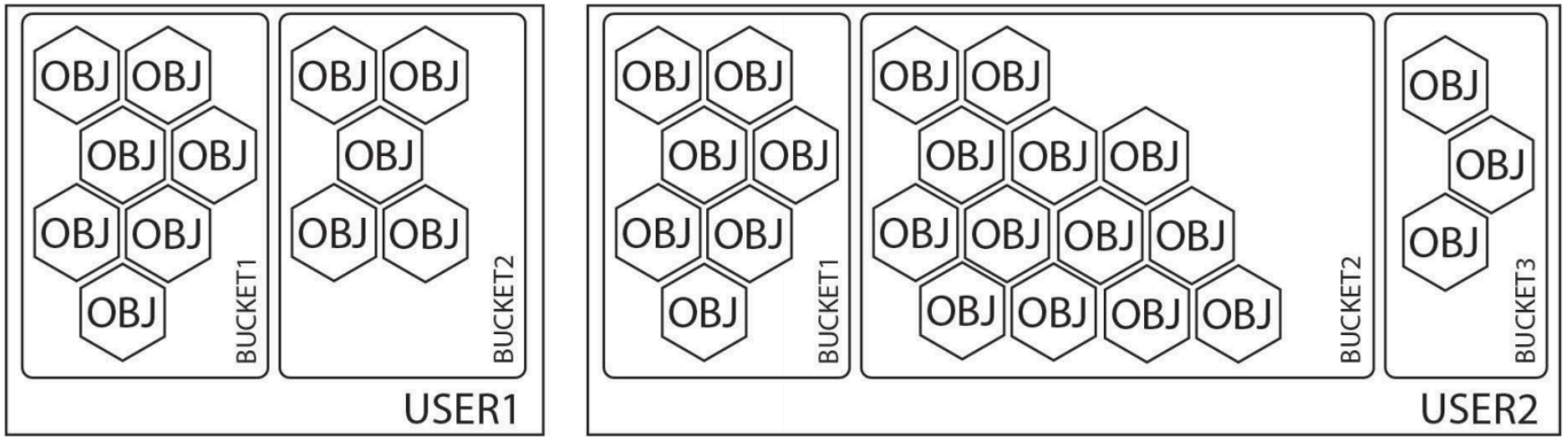


Image Credit: Sam Linston

Resources

http://docs.ceph.com/docs/luminous/

http://ceph.com/resources/

https://www.redhat.com/en/technologies/storage/ceph/resources

http://ceph.com/irc/

Ceph Questions Before Moving Along

Swift

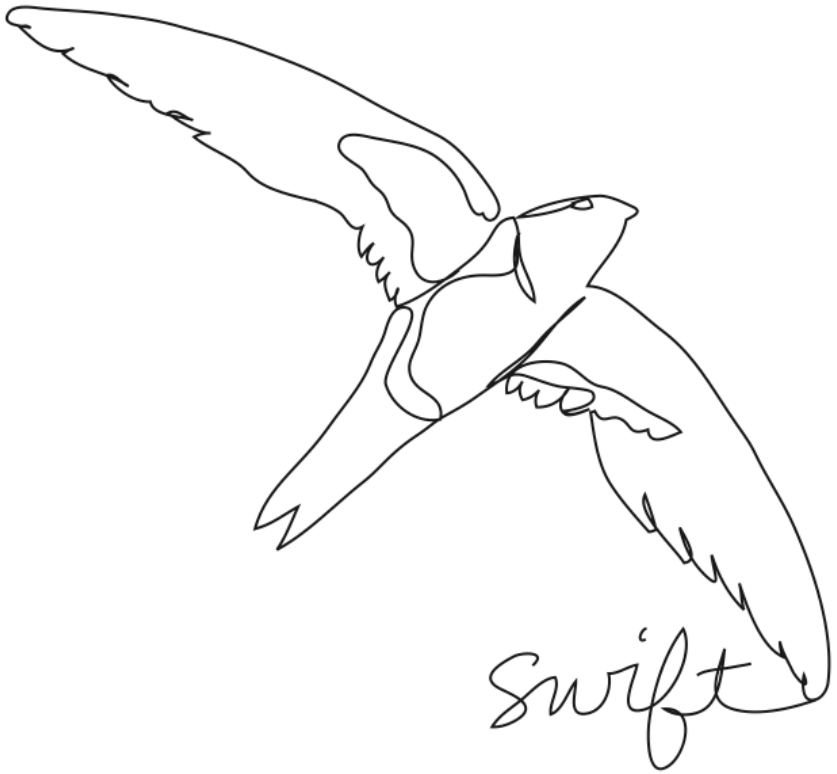


Image Credit: swiftstack.com

Overview

- Object file system that was developed around Open Stack
  - Open stack use for object is the largest use case for this object storage solution
- Large user base in the cloud space with lots of companies backing the project
- Leverages a ring based approach, with hashes determining file placement across the system
- Uses two types of servers
  - Proxy Servers – Field API requests
  - Storage Servers – Dish out and take the data

How Is it Different

- Unlike Ceph, only provides Object based storage
  - Has a very different architecture
  - Narrower use focus
  - Scales in a different, but neat way
- Leverages proxy servers for data access instead of clients directly being in contact with the storage servers
- Has better multi-region support
  - Doesn't rely on the Master-Slave concept that Ceph uses
- Assumes eventual consistency vs absolute consistency
  - Can be an issue in certain instances

Swift High Level Architecture

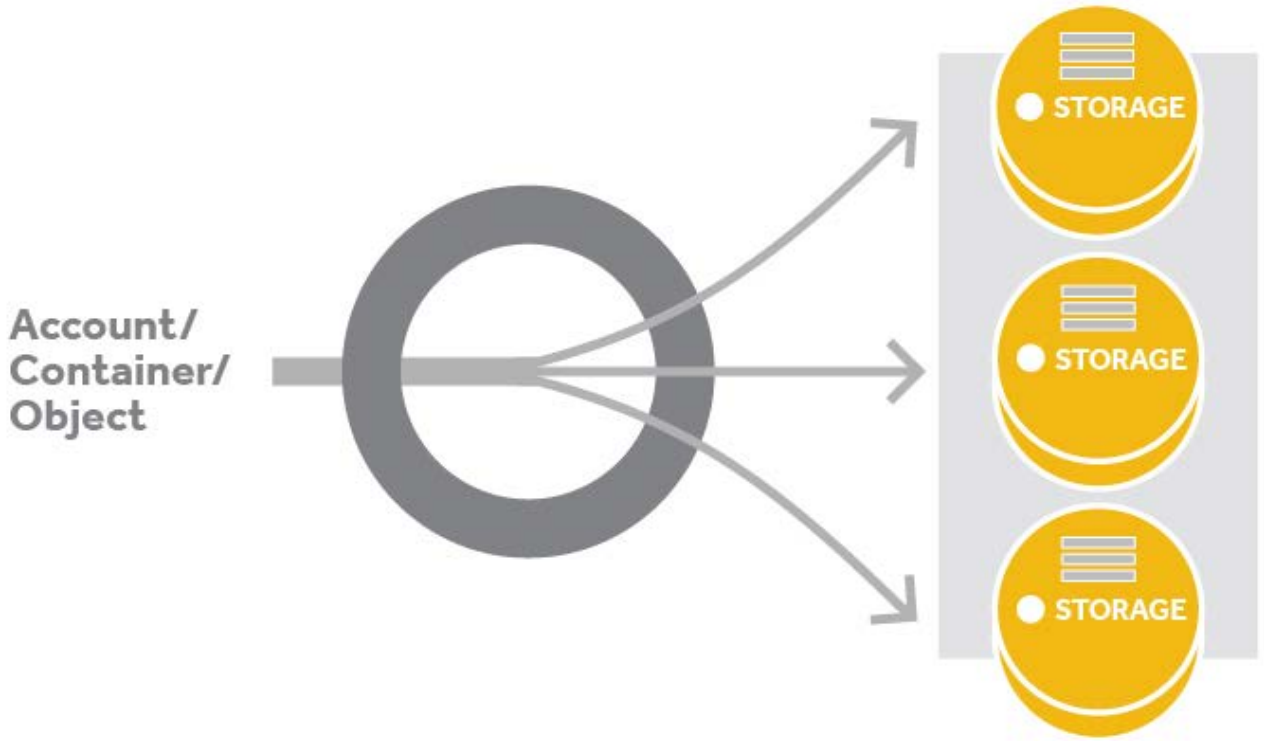


Image Credit: linkedin.com

Swift Lower Level Architecture

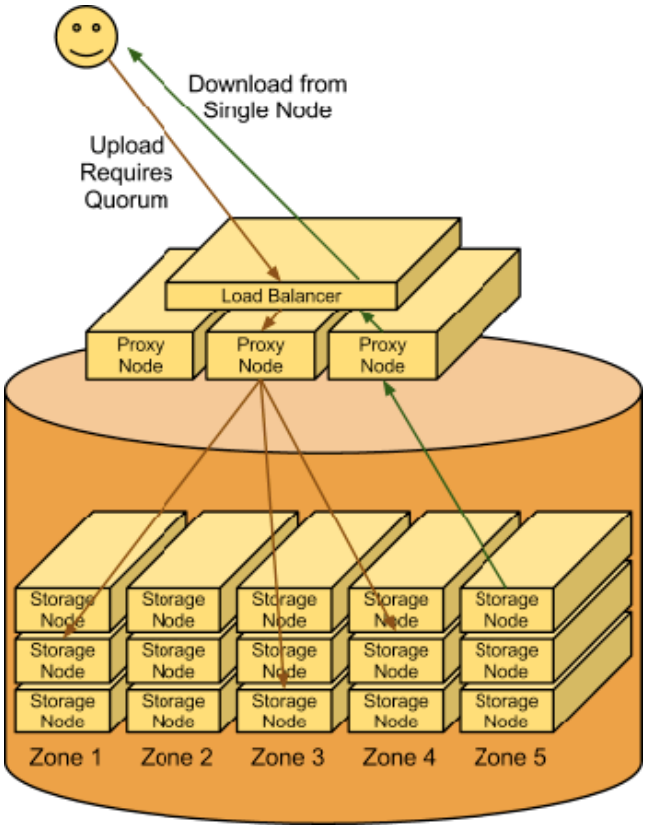
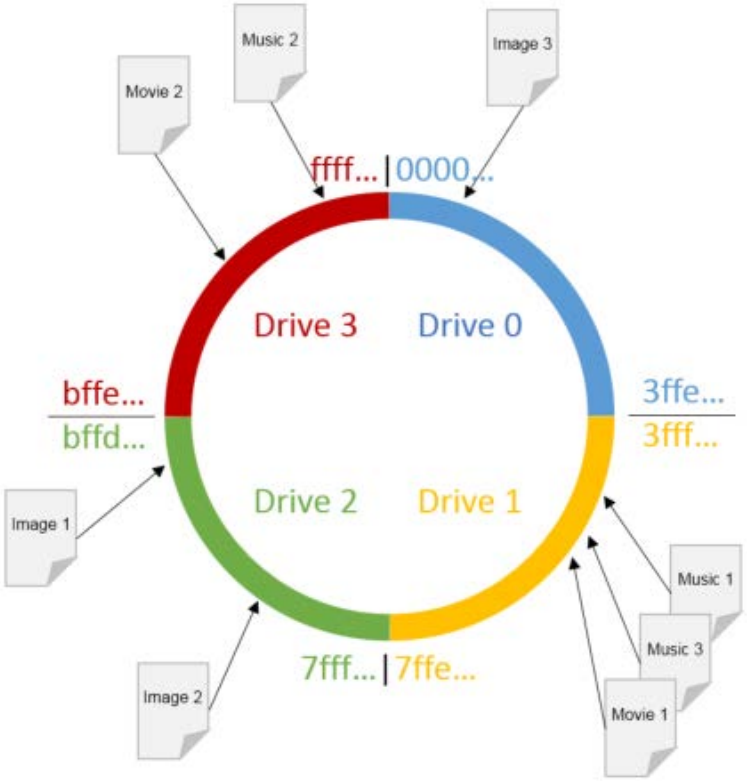


Image Credit: redhat.com

Ring Hashing



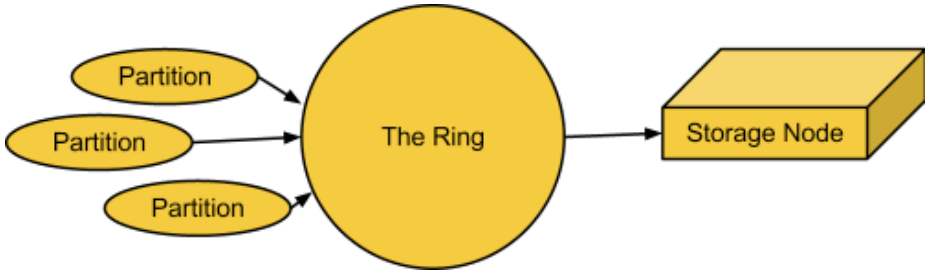


Image Credit: ihong5.wordpress.com, redhat.com

Benefits to the Ring Method

- Allows the number of proxy servers to scale extremely well
- All proxy servers are equal as the ring ensures they each know where all the data lives
- No need for coordinating between the proxy servers
  - Ring hash ensures that there will be no collisions of objects
  - Ring is static (unless capacity is being added or removed)
- Ease of updating and rolling proxy servers as there are just less to serve data requests
  - If proxy server update failure occurs, doesn't impact the system in any other way as long as quorum is still available

Downsides to the Ring Method

- Increases the latency of data access to the clients
  - No matter the speed of the server, it will add latency to the request as it has to broker the data exchange
  - Especially when working with smaller objects, latency increases are more noticeable
- Streaming performance also effected
  - Same issue as the increased latency, proxy servers decrease the ability of streaming performance of large files
- Eventual consistency can be problematic
  - Details on next slide
- Performance is now dependent on scaling two independent server groups

Consistency Overview

- What has to occur before a write to the system has been acknowledged back to the client machine
  - Multiple operations occur when a write is done on the file system
  - When is it ok to tell the client the write is done?
  - The sooner the write is acknowledged, the sooner the next one can be sent
- Absolute vs Eventual consistency each come with upsides and downsides
- Ceph and Swift handle this differently even though both are object stores

Absolute Consistency

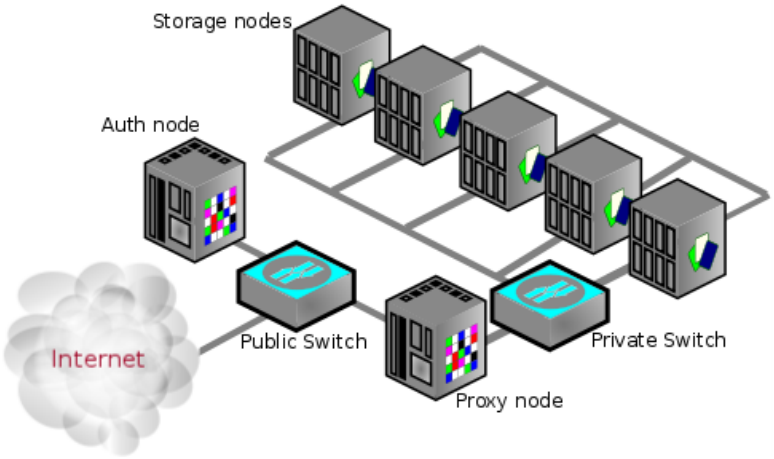
- All replicas of the object being written need to be in place and completed before the write is acknowledged back to client
- More strict, but ensures the write completed and the object is in the correct state
- Prevents conflict writes to an object from happening at the same time on different replicas, and a collision occurring
- Becomes a performance bottleneck when replication across physical regions is involved
  - Latency between sites will delay the write completion which thus increases the latency that the client sees

Eventual Consistency

- All replicas of the object being written do not need to be in place and completed before the write is acknowledged back to client
- Less strict, more flexible for architectures where more latency is involved
- Opens door to conflicting writes to an object from happening at the same time on different replicas
- Beneficial in multi-region deployments as usually multiple regions aren't dealing with the same objects
  - No increased latency from having a site over a WAN connection that could hurt performance

Load Balancing

- Load balancer is needed to distribute the API requests across the different proxy servers
- Clients aren't explicitly aware of the specific architecture, count, or names of the servers backing the service
  - Load balancer puts them in contact with random proxy server, client is handed data from the storage server holding data



Architecture Considerations

Proxy Servers

- Design with high network throughput capacity, 10Gb or greater speeds
- Moderate RAM needs, 32GB/64GB sufficient
- Good candidate for dense blade systems (4 node- 2U chassis available from many vendors)
- Doesn't need much local disk
- If access comes over public networks, use SSL to encrypt traffic

Architecture Considerations

Storage Servers

- Where the data actually lives, so high drive capacity is desired
- Consider your drive choice
  - Request is single thread to single drive, so single object stream throughput is at most the performance of one drive
  - Beware of SMR drives or consumer drives to keep performance up
- Network performance important for feeding data to clients, use 10Gb or faster
  - Beneficial to have two networks one for data access and another for cluster activities such as replication, especially if you use lower speed networks

Rate Limiting

- Prevents a single container access from dominating the system and hurting performance of other users
- Can be down with bandwidth or IOPS
- Rate limits can be applied not just to containers, but also to certain users/accounts
  - Great for users who consume high I/O needs
  - Ensures fair access to the system for all
- Architecture Note:
  - Time synchronization across the proxy servers is important here, skew can result in some rate limiting issues as the proxy servers track how much data is being access over given times

Resources

- https://docs.openstack.org/swift/latest/admin_guide.html
- https://www.swiftstack.com/product/openstack-swift
- https://wiki.openstack.org/wiki/Swift

Acknowledgements

- Thanks to Sam Linston from University of Utah for help with slide development content
- Members of the SET group at NCSA for slide review
- Members of the steering committee for slide review

Questions