Unified Cluster Federation: A Hyper Scaled Architecture for Heterogeneous Clusters

Romit Misra
Varada Kari
Agenda

- Flipkart Cloud Platform and CEPH deployment
- Challenges in managing a highly available Object Storage Service
- Programmable routing layer
- Use cases in Action at Flipkart
About Flipkart

• A leading Indian e-commerce organization
• Developed a highly scalable and reliable storage infrastructure
• Has been running Ceph with RGW for more than 4 years in production
• Owns data centers having more than 50,000 hosts
• Completely virtualized infrastructure with over 200,000 VMs
Data and Access Patterns

- Lot of images, videos, pdfs, VM images, debs, xls, DB backups
- Multiple users with different use cases and patterns, a good fit for multi-tenant system
- Need a highly available, durable backend and easily addressable/accessible.
- Need distributed access across multiple workflows
Flipkart Cloud Platform Object Storage Service

- Scale-out object storage service built on CEPH for Flipkart internal consumption
- Simple HTTP API - A subset of AWS S3 API
- Accessible through ELB virtual IP
- Single endpoint for multiple sub clusters
- Key features
  - Focus on durability
  - Elastic for performance and capacity
  - Multi-tenant with user SLAs
  - Five 9's of reliability
CEPH deployment at Flipkart Cloud Platform

- Boto
- AWS SDK for Java
- Clients(CDN)
- Clients(Repository)
- Clients(Digital Signature Service)
- Clients(Catalogue)
- Clients(Backup)

VIP

Elastic Load Balancer

Nginx

ELB (Read)

RGW (Read)

RGW (Write)

VIP

ELB (Write)

RGW (Read)

RGW (Write)

Debian VMs

Mon

RADOS Cluster

Cache/Index

OSD (SSD)

Base Tier

OSD Hybrid

Bare Metal H/W (Debian)

Time Series Based Health Monitoring

- Grafana
- Open TSDB
- Alerts
Challenges

- Zero downtime and 24X7 availability of the service
- Unpredictable object growth - 3x growth, in 9 months, current cluster is holding up close to 3 Billion objects
- Transparent addition of capacity without any service impact
- Enterprise Grade Multi Tenancy Support missing in Ceph
- Practical Limitations for Scaling a Single Cluster (Pre Luminous)
- Missing QOS Support
Path to L-release

- Cluster Upgrade (Ceph Releases Hammer → Luminous +)
  - Need a jump to Jewel
  - Chown to ceph user
  - Huge number of small files ~1.5 Billion

- Store upgrade (FileStore → BlueStore)
  - No direct upgrade path from FileStore to BlueStore
  - Use recovery to migrate to BlueStore

- Cluster Upgrade (Ceph Releases Hammer → Luminous +)
  - Create a new cluster with BlueStore backed osds with Luminous
  - Lift and Shift the objects to new cluster

- Store upgrade (FileStore → BlueStore)
  - No need to upgrade any osds, new ones are BlueStore based
Cluster Templates

- A qualified cluster configuration
  - Specified cluster capacity
  - Tested for IOPS/Bandwidth
  - Cluster tuned for object sizes and latencies
  - Reserved recovery and replication bandwidth
  - Partial QOS support for all the ops
  - Failover support for machine failures without degradation
How to?

- Keep the endpoint same, Federate the clusters (Horizontal Scaling)
- Introduce a configurable policy based data placement layer
- Routing of request based upon USER and BUCKET Policies
Standard Nginx Deployment

- Nginx connected under a Elastic Load Balancer
- Nginx Instances are horizontally scalable
- Request processing via a config file. Nginx config allows to define upstreams
- Peeking into the request and writing complex logic is not dynamic and not straightforward
- Works well when the topology changes are infrequent
Need a Next Generation Scalable Routing Layer...
Policy driven programmable Nginx

- Extend the standard Nginx Config File, via Lua Interface
- Introduce a Policy Driven Framework, powered by a Distributed Cache
- On the Fly Config Change
- Policy driven config based on user and Bucket
- Seamless addition of Clusters
- Cluster, User, Bucket --> Entities to Model Policy
Nginx Openresty (Luajit + Nginx)

- Enables to write complex logic for request processing, A programmable Nginx

- Access to Request States for Processing

- Availability of libraries for integrating different authentication, encryption, http processing and much more ....

- Availability of modules for integrating solutions like redis, memcached..
Federation, R6

- Route :- Multi Cluster Traffic Routing
- Redirect :- Traffic Rerouting under Outages
- Replicate:- Load Testing, Migration
- Rebalance*:- User / Bucket Movement across clusters.
- Rate Limit**:- Rate Throttling per User
- Request Analytics:- Deep insights into latency, access patterns, error codes etc..
Modified Architecture

[Diagram showing a modified architecture with connections between various clusters and repositories.]
Deepdive to Capabilities
S3 Request (Simplified)

headers={'Authorization': 'AWS ABCDEFFHYMM2XGJKLMN:NUzM8BQ62bu2ht5Fbt07F+CDFdf='}

Bucket (URI → http://bucket)

Object (URI → http://bucket/object)
Possible Entities for Policy Engine
## Cluster Model (CM)

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>cluster name (Unique Key)</td>
<td>zone_1_cluster</td>
</tr>
<tr>
<td>cluster_write_endpoint</td>
<td>20.0.7.14</td>
</tr>
<tr>
<td>cluster_read_endpoint</td>
<td>20.0.7.15</td>
</tr>
<tr>
<td>cluster_state</td>
<td>600</td>
</tr>
<tr>
<td>fallback_cluster</td>
<td>zone_1_fallback_cluster</td>
</tr>
<tr>
<td>epoch</td>
<td>12</td>
</tr>
<tr>
<td>version</td>
<td>4</td>
</tr>
</tbody>
</table>

- **Cluster Name** → Unique Name Identifier
- **Write Endpoint** → Write ELB IP/ RGW
- **Read Endpoint** → Write ELB IP/ RGW
- **Cluster state** → Health State *(BAU / DEGRADED / UNAVAILABLE)*
- **Fallback Cluster** → Redirection Cluster *
- **EPOCH** → Key change Epoch**
- **Version** → Map Version **
# User Model (UM)

<table>
<thead>
<tr>
<th>User Access Key (Unique Key)</th>
<th>UHBVC9ZXJKLSAQ</th>
</tr>
</thead>
<tbody>
<tr>
<td>Bootstrap Cluster</td>
<td>zone_1_cluster</td>
</tr>
<tr>
<td>Placement Policy</td>
<td>Default</td>
</tr>
<tr>
<td>user_name</td>
<td>logan</td>
</tr>
<tr>
<td>target_write_cluster</td>
<td>zone_1_cluster</td>
</tr>
<tr>
<td>epoch</td>
<td>4</td>
</tr>
<tr>
<td>version</td>
<td>6</td>
</tr>
</tbody>
</table>

- **User Access Key → Unique Key**
- **Bootstrap Cluster → Name Identifier (Refers CM)**
- **Placement Policy → Default**
- **user_name → User Name (uid)**
- **target_write_cluster → Cluster Target “Bucket Create” (Refers CM)**
- **EPOCH → Key change Epoch**
- **Version → Map Version**
## Bucket Model

<table>
<thead>
<tr>
<th>Bucket name (Key)</th>
<th>wolverine</th>
</tr>
</thead>
<tbody>
<tr>
<td>access_key</td>
<td>UHBVC9ZXJKLSAQ</td>
</tr>
<tr>
<td>Active Sync</td>
<td>zone_1_cluster</td>
</tr>
<tr>
<td>Active</td>
<td>zone_1_cluster</td>
</tr>
<tr>
<td>State</td>
<td>200</td>
</tr>
<tr>
<td>opmode</td>
<td>600</td>
</tr>
<tr>
<td>replication_enabled</td>
<td>0</td>
</tr>
<tr>
<td>epoch</td>
<td>4</td>
</tr>
<tr>
<td>version</td>
<td>1</td>
</tr>
</tbody>
</table>

- **Bucket Name** → Bucket Name Identifier
- **access_key** → bucket to user key mapping (Refers UM)
- **Active sync** → Current Active Sync** (Refers CM)
- **active** → Current Active Cluster (Refers CM)
- **opmode** → Bucket Operation Mode
- **replication_enabled** → replication flag for the bucket
- **EPOCH** → Key change Epoch**
- **Version** → Map Version**
Route Request Flow
Replication Request Flow

Original Request

Bucket Lookup
Highly Available Redis Cache

Elastic Openvzooy Instances

Response Dropped
Replicated Response

Original Request

Original Response

Replicated Request

Cluster A

Cluster B

Cluster C
Redirect Request Flow
Rebalance Request Flow
Rate Limiting Based on Redis GCRA Plugin
Request Analytics
Overall Picture
Nginx Lua Interactions

- content_by_lua
  - Major URI Rewrite and request Processing
- log_by_lua
  - Request logging and Statistics Processing
Under the Hood

- INIT
- Request Builder
- Response Dispatch
- Routing Engine
- Cache Service
- Statistics Manager
- Config Manager
- Async Timer/s
- Application Load Balancer
Redis Deployment Architecture

- Sentinels: Port 16380
- Master: Port 6380 (Initial)
- Slaves: Port 6380, 6381, 6382
Important Observations

- Read Heavy Pattern for Metadata (Redis Keys)
- Keys Seldom Changes, once written under BAU conditions.
- Key Space Is a function of User and Buckets (Can be predicted with better accuracy, without the need to worry for Key Space Explosion)
- Current Projections allow entire Space to be “IN MEMORY” without the need for Flushing
Optimizations

- Lua Local Cache Per Worker Process
- Local Cache Updates Controlled by Epoch Values in Key
- Saves on N/W bandwidth and overhead to and from Redis Cluster
- Master Only Accepts Writes
- Reads are send in Round Robin Fashion to Connected Slaves
- No Sharing of Performance Critical Constructs/Variables Between Workers
- Timer Task (Separate Thread) Drives Async Events Such as Statistics Push, Local Config Reads
- No Blocking Event on The Main Worker
### Performance Latencies and Scale Numbers

<table>
<thead>
<tr>
<th>Category</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Average Increase in Latency (Baseline is Standard Nginx)</td>
<td>2 ms</td>
</tr>
<tr>
<td>Peak Latency Observed (Bucket Create)</td>
<td>10 ms</td>
</tr>
<tr>
<td>Key Space Size (Current)</td>
<td>.2 million</td>
</tr>
<tr>
<td>Key Space Growth (Supported for Current Deployment)*</td>
<td>20 million</td>
</tr>
<tr>
<td>Peak RPS</td>
<td>2000</td>
</tr>
<tr>
<td>Average RPS</td>
<td>800</td>
</tr>
<tr>
<td>Tested and Projected RPS (With Current Deployment)*</td>
<td>15000</td>
</tr>
</tbody>
</table>
Challenges

- Schema Upgrades for Keys (Zero Traffic Disruption)
- Handling Failures and Inconsistencies (Redis can be Inconsistent)
- User / Admin Errors and Offline Reconciliation
References

- [www.ceph.com](http://www.ceph.com)
- [www.scality.com](http://www.scality.com)
- [https://github.com/openresty](https://github.com/openresty)
- [https://github.com/openresty/lua-resty-redis](https://github.com/openresty/lua-resty-redis)
- [https://github.com/ceph](https://github.com/ceph)