Apache Ozone Erasure Coding (EC)

The Modern Big Data Object Store with More Than 50% Storage Space Savings

Uma Maheswara Rao Gangumalla
Sr. Engineering Manager, Cloudera Inc

Stephen O’Donnell
Sr. Staff Software Engineer, Cloudera Inc

@UmaMaheswaraG
Who Am I?

- Sr. Engineering Manager at Cloudera
- Apache Software Foundation Member
- More than 10 years with Apache Projects.
- Apache Hadoop Project Management Committee (PMC) Member
- Apache Ozone PMC Member
- Apache Incubator PMC
- Mentored several projects at Incubator
- ApacheCon Big Data track chair - 2021, 2022
What is Ozone?

- Apache Ozone is a distributed, scalable, and high performance object store.
- Ozone is designed and optimized for Big Data workloads.
- Ozone can scale up to billions of objects and work effectively in containerized environments like Yarn or Kubernetes.
- Ozone is strongly consistent and provides the benefits of traditional HDFS and S3 Object Store.
- Scale to 1000’s of nodes with dense storage configurations.
- Apache Spark, Hive and YARN work without any code modifications by using OFS protocol.
Apache Ozone Architecture
Quick Overview of Non EC Flow

Ozone Write a Key

1. Client → Ozone Manager → SCM
   - Client
     - PutKey

2. Client
   - Writes data as chunks
   - Update metadata of a block
   - Datanode

3. Client → Ozone Manager
   - Commit Key

<table>
<thead>
<tr>
<th>Block</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Block 001</td>
<td>{ List of Chunks }</td>
</tr>
<tr>
<td>Block 002</td>
<td>{ List of Chunks }</td>
</tr>
</tbody>
</table>
Erasure Coding Requirements

- **Phase - I**
  - Enable EC at Cluster/Bucket Level
  - Should be able to Write files in EC format
  - Should be able to Read the files which were written in EC format.
  - Should support 3:2, 6:3, 10:4 EC Schemes
  - Should be able to recover the files automatically on failures
    - Online recovery
- **Phase - II**
  - Offline recovery
- **Phase - III**
  - Should provide options to enable EC via Recon / CM
  - Should be able to convert the files from EC to RATIS (and vice versa)
EC Architecture - Write

- **Container Group**: A container created in data + parity with separated instances.
- **Block Group**: A block presents in a container group.
- Each data+parity chunks written to block group.
- Parity generated at

---

**Protobuf:**

```protobuf
descriptor file
```

**Input File:**

```
256MB
```

**Ozone Client:**

```
Container Group:
A container created in data + parity with separated instances.
```

**Block Group:**

```
A block presents in a container group.
```

**Each data+parity chunks written to block group.**

**Parity generated at**
EC Architecture - Write

➢ When node fails, block group will be closed and new block group requested from OM
➢ SCM uses EC Pipeline Provider for creating EC pipeline.
➢ No Ratis in the EC Path. Pipeline is just a logical group id for set of nodes.
EC Write: Striping

- **Stripe**: One round of data + parity chunks called as full stripe.
- Chunks would be written in round robin fashion to data nodes.
- Parity Generation: After every data number of chunks written, parity will be generated and send to remaining nodes in group.
- ReplicaIndex: It will represent the position of chunk with respective to ec input buffers order. In other words, EC Chunk position in full stripe, in the order of 1 to (data + parity)
- If stripe write fails, the current block group will be closed and rewrite the failed stripe to new block group.
EC Write: Striping

**Input File**
- 1MB - chunk1
- 1MB - chunk2
- 1MB - chunk3
- 1MB - chunk4
- 1MB - chunk5
- 1MB - chunk6

**Ozone Client**

**EC Container Group: 1**

<table>
<thead>
<tr>
<th>N1</th>
<th>N2</th>
<th>N3</th>
<th>N4</th>
<th>N5</th>
</tr>
</thead>
<tbody>
<tr>
<td>Filled by previous file blocks</td>
<td>Filled by previous file blocks</td>
<td>Filled by previous file blocks</td>
<td>Filled by previous file blocks</td>
<td>Filled by previous file blocks</td>
</tr>
</tbody>
</table>

**Stripe-1**
- c1:block1
- c2:block2
- c3:block3
- parity1 (c4, c5, c6)

**Stripe-2**
- c4:block4
- c5:block5
- c6:block6
- parity2 (c1, c2, c3)

**Replica Index**
- ReplicaIndex-1
- ReplicaIndex-2
- ReplicaIndex-3
- ReplicaIndex-4
- ReplicaIndex-5
EC Write: Partial Stripe with Padding

Client uses padding data for generating parity chunks if stripe is not full.

Partial Stripe: chunk2 and chunk 3 assumed as padding data and len is 1MB.

Partial Stripe: chunk3 assumed as padding data and len is 1MB.

Full Stripe: No padding needed.
EC Write: Striping

➢ If stripe write fails, the current block group will be closed and rewrite the failed stripe to new block group.

➢ Client keep track of bytes written and check for failures.

➢ After all data writes finishes, then parity writes. Once full stripe write done, client calls putBlock on all streams.

➢ Writes will update the current block group length on every put block which will be stored at DNs.
EC Read

➢ Reads in the same order in which order writes done. Order will be based on replica Indexes.
➢ Client stitches the data back to original order and serves to user.
➢ Client does not read from parity replicas unless there are failures.
EC Read

Read File

- 1MB - chunk1
- 1MB - chunk2
- 1MB - chunk3
- 1MB - chunk4
- 1MB - chunk5
- 1MB - chunk6

Chunk reads order

1 2 3 4 5 6

Ozone EC Client

EC Container Group : 1

- N1
  - Filled by previous file blocks
  - c1:block1

- N2
  - Filled by previous file blocks
  - c1:block1
  - c2:chunk

- N3
  - Filled by previous file blocks
  - c1:block1
  - c3:chunk

- N4
  - Filled by previous file blocks
  - c1:block1
  - parity1 (c4, c5, c6)

- N5
  - Filled by previous file blocks
  - c1:block1
  - parity2 (c4, c5, c6)

- Replication Index-1
- Replication Index-2
- Replication Index-3
- Replication Index-4
- Replication Index-5

Stripe - 1

- 1MB c1:chunk1
- 1MB c2:chunk
- 1MB c3:chunk

Stripe - 2

- 1MB c4:chunk4
- 1MB c5:chunk
- 1MB c6:chunk

- parity1 (c1, c2, c3)
- parity2 (c1, c2, c3)

- parity1 (c4, c5, c6)
- parity2 (c4, c5, c6)
EC Reconstructional Reads

➢ First read will attempt to read data blocks (non parity blocks).
➢ When node failed while reading, client will switch to reconstructional read and uses parity for reconstructing the lost data transparently.
➢ Degraded Reads: Reconstruction read will be slow due to ec decode operation.
➢ To avoid the degraded reads, we need to recover the lost replicas offline.
EC Recovery Reads

EC Container Group: 1

N1
Filled by previous file blocks

N2
Filled by previous file blocks

N3
Filled by previous file blocks

N4
Filled by previous file blocks

N5
Filled by previous file blocks

blockGrpID: 1

1MB - c1:block1

1MB - c1:block1

1MB - c1:block1

1MB - c1:block1

1MB - parity1
(c4, c5, c6)

1MB - parity2
(c4, c5, c6)

Reconstruct and Read stripe-1

Reconstruct and Read stripe-2

Read File

Ozone EC Client

1MB - c1:chunk1
1MB - c1:chunk1
1MB - c1:chunk1
1MB - c1:chunk1
1MB - c1:chunk1
1MB - parity1
(c1, c2, c3)
1MB - parity2
(c1, c2, c3)

1MB - c3:chunk3
1MB - c3:chunk3
1MB - c3:chunk3
1MB - c3:chunk3
1MB - c3:chunk3

1MB - c2:chunk2
1MB - c2:chunk2
1MB - c2:chunk2
1MB - c2:chunk2
1MB - c2:chunk2

1MB - c4:chunk4
1MB - c4:chunk4
1MB - c4:chunk4
1MB - c4:chunk4
1MB - c4:chunk4

1MB - c5:chunk5
1MB - c5:chunk5
1MB - c5:chunk5
1MB - c5:chunk5
1MB - c5:chunk5

1MB - parity1
(c1, c2, c3)

1MB - c6:chunk6
1MB - c6:chunk6
1MB - c6:chunk6
1MB - c6:chunk6
1MB - c6:chunk6

1MB - parity2
(c4, c5, c6)
What is the Offline Recovery?

➢ When a node/Disk lost, we will lose the containers which are residing in that node/disk. We need a mechanism to recover that lost containers in the background. We call this process of background recovery as “Offline Recovery”.

➢ This is very critical background task similar re-replication on node/disk failures.
Offline Recovery

How the missing containers are detected in EC?

- Node failures detection happens at the SCM. When a node failed, all the container replicas in that node would be considered as missing. So, all SCM replica copies of that node will be removed.
- RM scans the containers periodically and find if any missing replicas.
- RM will also detect if any container is over replicated.
- RM creates the reconstruction command if it finds the container is in under replication.
- The first DN from the target will be chosen as coordinator to reconstruct all the lost containers.
Offline Recovery

ReconstructECContainersCommand
1. containerID
2. Source replicas: DN1, DN2, DN3
3. Target DNs - DN6, DN7
4. Missing indexes
5. ECReplicationConfig

Container Recovery Done
Target DNs sending ICR

Create recovering state containers in target DNs
List blocks
Find Blocks
Loop for recover Block for all Block
Recover Block
EC decode and recover lost index chunks
Write chunk to target container
Transfer chunk to the target container
Close the containers in target DNs
EC Replication Config

➢ Format: CODEC-DATA-PARITY-CHUNKSIZE
  ◦ RS-3-2-1024K
  ◦ RS-6-3-1024K
  ◦ RS-10-4-1024K
  ◦ XOR-3-2-1024K
  ◦ XOR-6-3-1024K
  ◦ XOR-10-4-1024K
Enabling at Cluster Level EC

Use the following configurations for enabling EC at cluster level. They should present at OM service.

```
<property>
  <name>ozone.server.default.replication</name>
  <value>RS-X-Y-1024k</value>
</property>

<property>
  <name>ozone.server.default.replication.type</name>
  <value>EC</value>
</property>
```
Enabling at Bucket Level EC

➢ Users organizes data with buckets.
➢ Depending on use case, they choose bucket types (pure objects, FSO objects).
➢ Enable EC at bucket creation time:

```
ozone sh bucket create <bucket path> --type EC --replication rs-6-3-1024k
```

➢ Changing EC config on existing bucket:

```
ozone sh bucket set-replication-config <bucket path> --type EC --replication rs-6-3-1024k
```
Enabling at Key Level EC

- Key Creation time:

```
ozone sh key put <Ozone Key Object Path> <Local File> --type EC --replication rs-6-3-1024k
```
EC Configuration Preferences

➢ For Ozone/Java Client:
  *Client Specified Value > Bucket Config > Cluster Config*

➢ For OFS/O3FS/S3 Clients:
  *EC Bucket Config > Client Specified > Cluster Config*
OFS, O3FS and S3 Clients EC Options

➢ FS and S3 client can use only bucket level EC.

➢ There is no direct way, they can specify EC options per file from clients.
  ○ FS interface does not have appropriate API to specify EC options. We could only pass short value as replication factor.

```java
/**
   * Create an FSDataOutputStream at the indicated Path.
   * @param f the file name to open
   * @param overwrite if a file with this name already exists, then if true, the file will be overwritten, and if false an error will be thrown.
   * @param bufferSize the size of the buffer to be used.
   * @param replication required block replication for the file.
   * @throws IOException IO failure
   */
  public FSDataOutputStream create(Path f,
          boolean overwrite,
          int bufferSize,
          int replication,
          long blockSize) throws IOException {
      return create(f, overwrite, bufferSize, replication, blockSize, progress: null);
  }
```

○ S3 storage classes are not covering directly EC options to specify.

```java
package com.amazonaws.services.glacier.model;

public enum StorageClass {
    STANDARD, REDUCED_REDUNDANCY, STANDARD_IA;
}
Where We Are?

Phase - I
1. Enable EC at Cluster/Bucket Level
2. Should be able to WRITE files in EC format
3. Should be able to READ the files from EC buckets.
4. Should support 3:2, 6:3, 10:4 EC Schemes
5. Should be able to recover the files automatically on failures
   a. Online recovery

Phase - II
a. Offline recovery

Phase - III
1. Should provide options to enable EC via Recon / CM
2. Should be able to convert the files from EC to RATIS (and vice versa)
Ozone EC Development Stats And Acknowledgements

➢ Developed ALL Jiras under HDDS-7285 and HDDS-6462
➢ 200+ Apache JIRAs Resolved

Acknowledgements: (Names are in alphabetical order)
Aswin, Attila, Jackson, Kaijie, Mark, Marton, Nilotpal, Pifta, Stephen, Swami, Uma

Many thanks to design reviewers:
Arpit, Bharat, Karthik, Marton, Nanda, Sid, Stephen, Yiqun Lin
Please come and join in Ozone Development

- Github repo: https://github.com/apache/ozone
- Looking to contribute to the Apache Ozone project?
  - Start with https://github.com/apache/ozone/blob/master/CONTRIBUTING.md
- Bug reporting is at: https://issues.apache.org/jira/projects/HDDS
Thanks

Q&A

umamahesh@apache.org  |  umagangumalla@cloudera.com

Twitter: @UmamaheswaraG