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

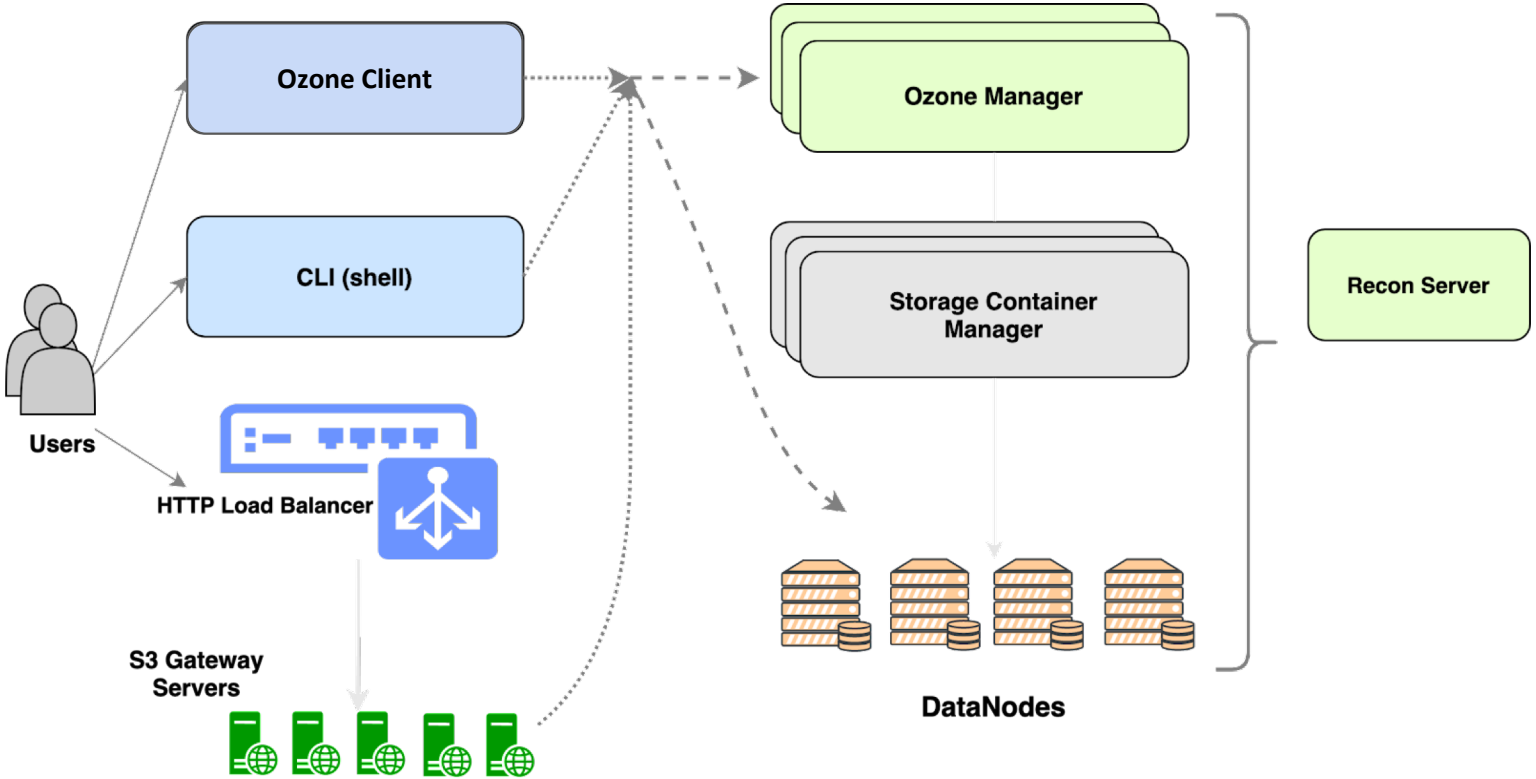
Who Am I?

- ❏ 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

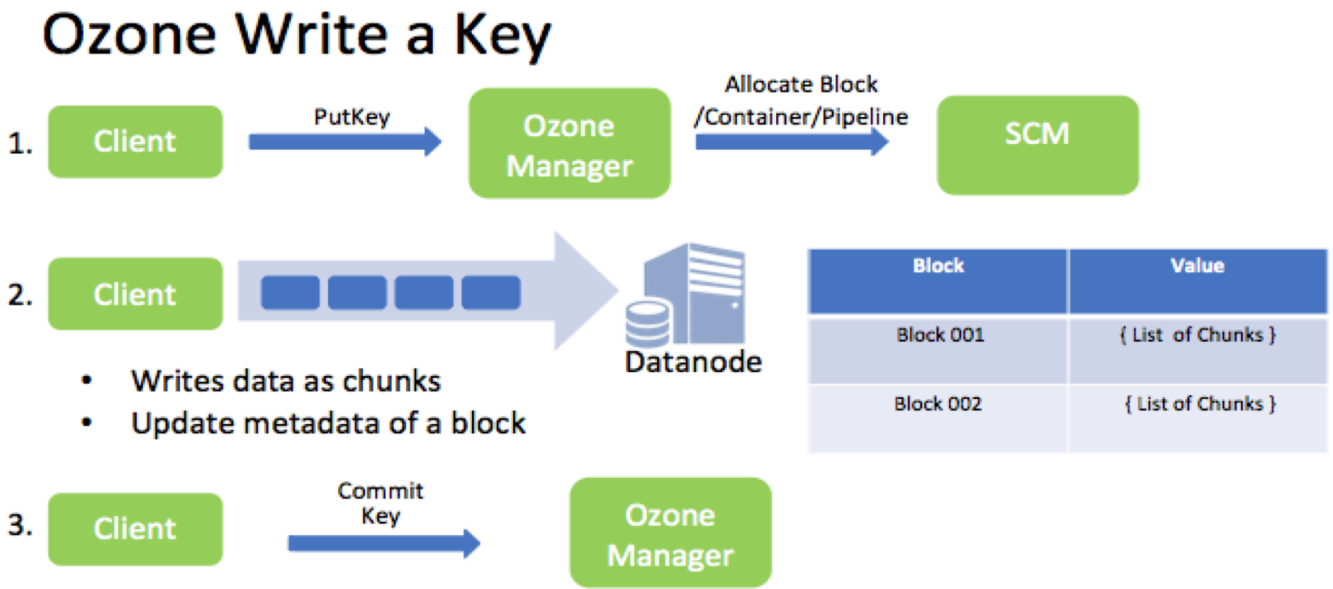
What is Ozone?

- 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



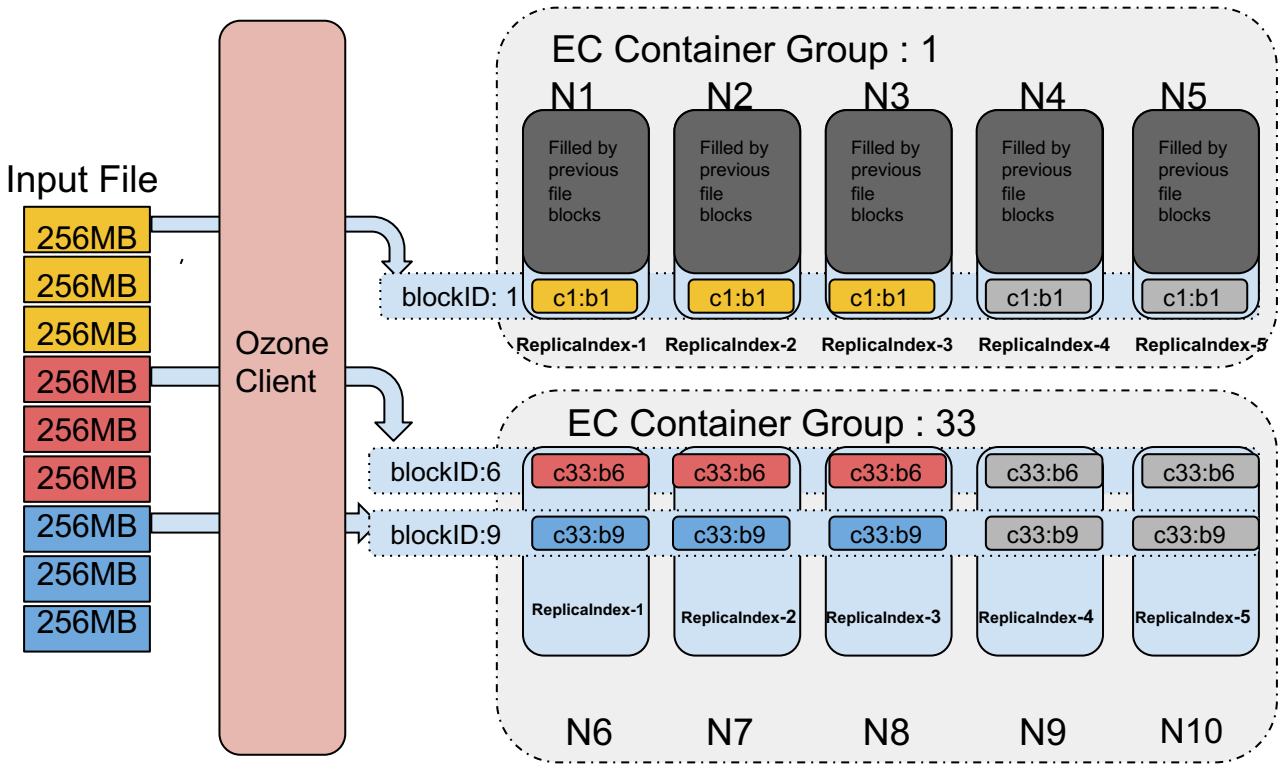
Quick Overview of Non EC Flow



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

CLOUDERA
APACHECON North America

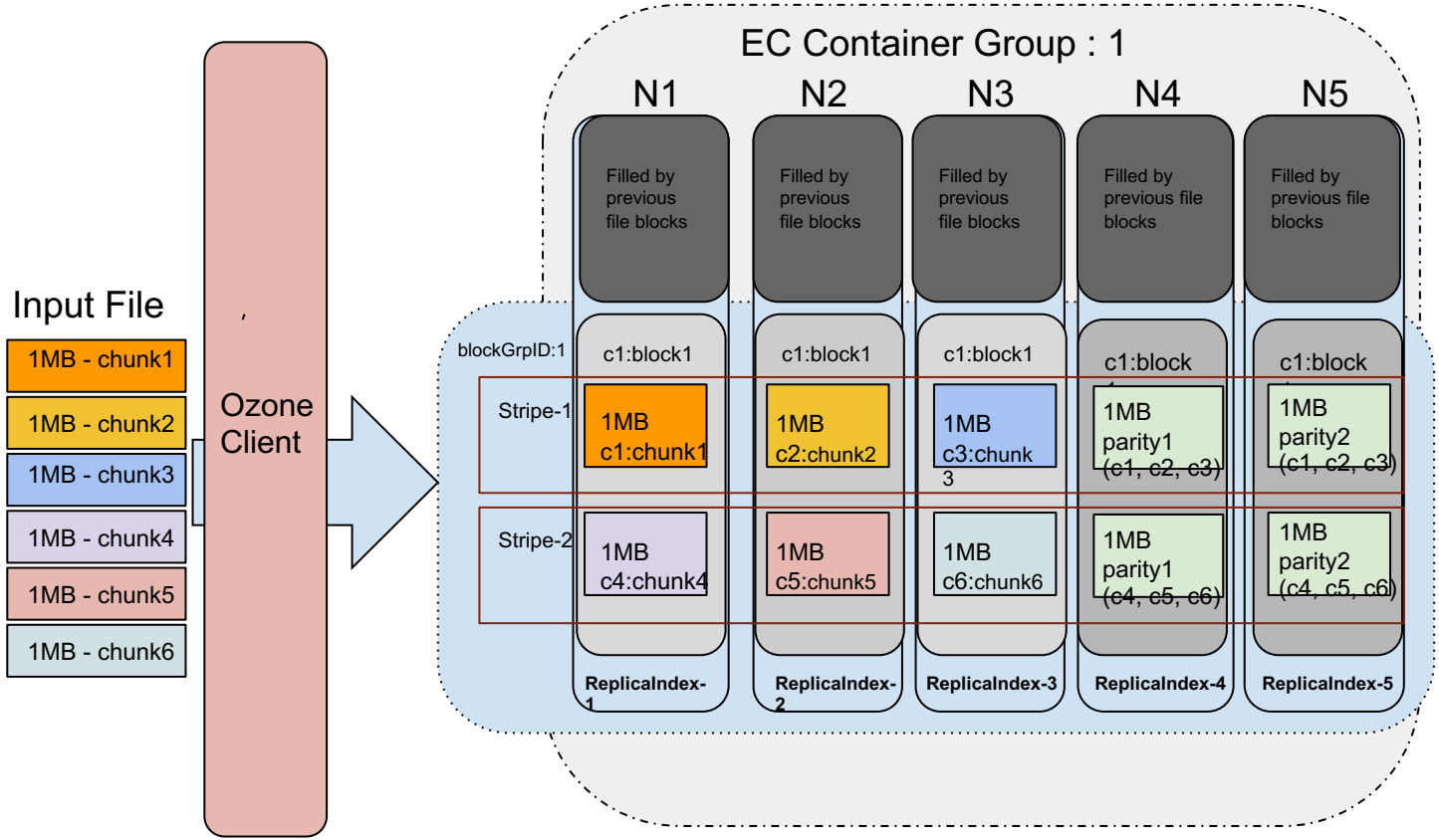
EC Architecture - Write

- 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.

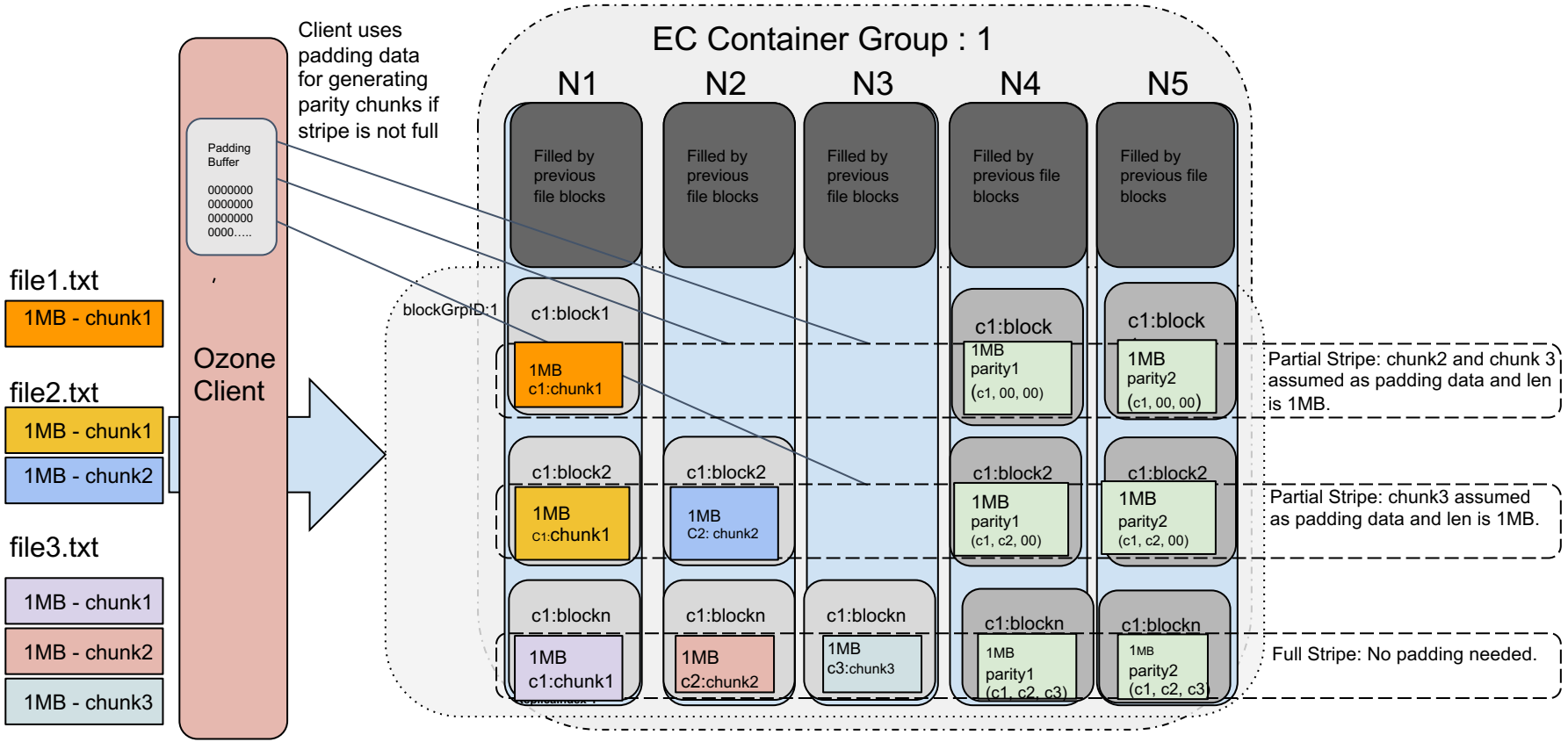
EC Write: Striping

- **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

EC Write: Striping



EC Write: Partial Stripe with Padding



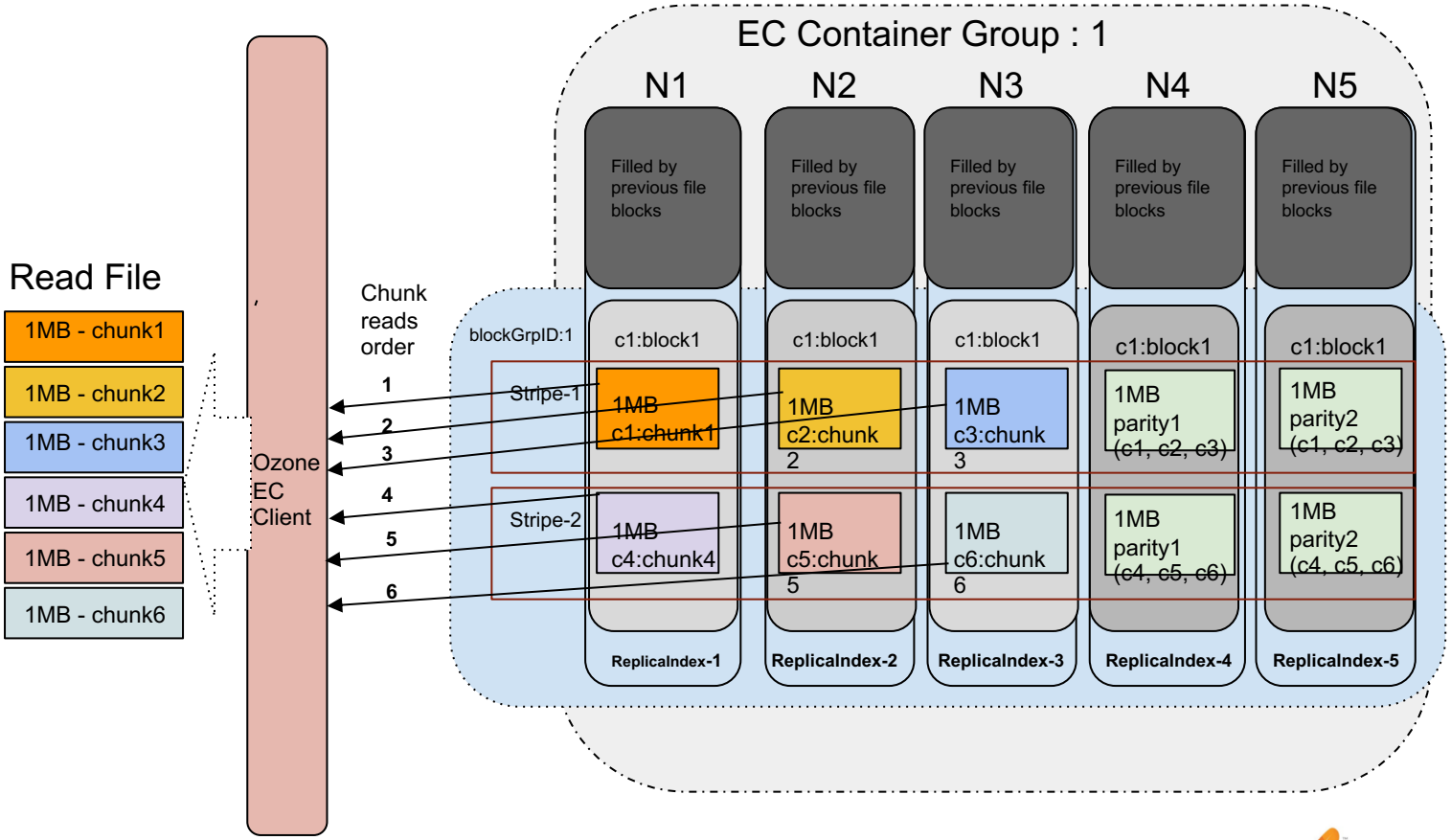
EC Write: Striping

- 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



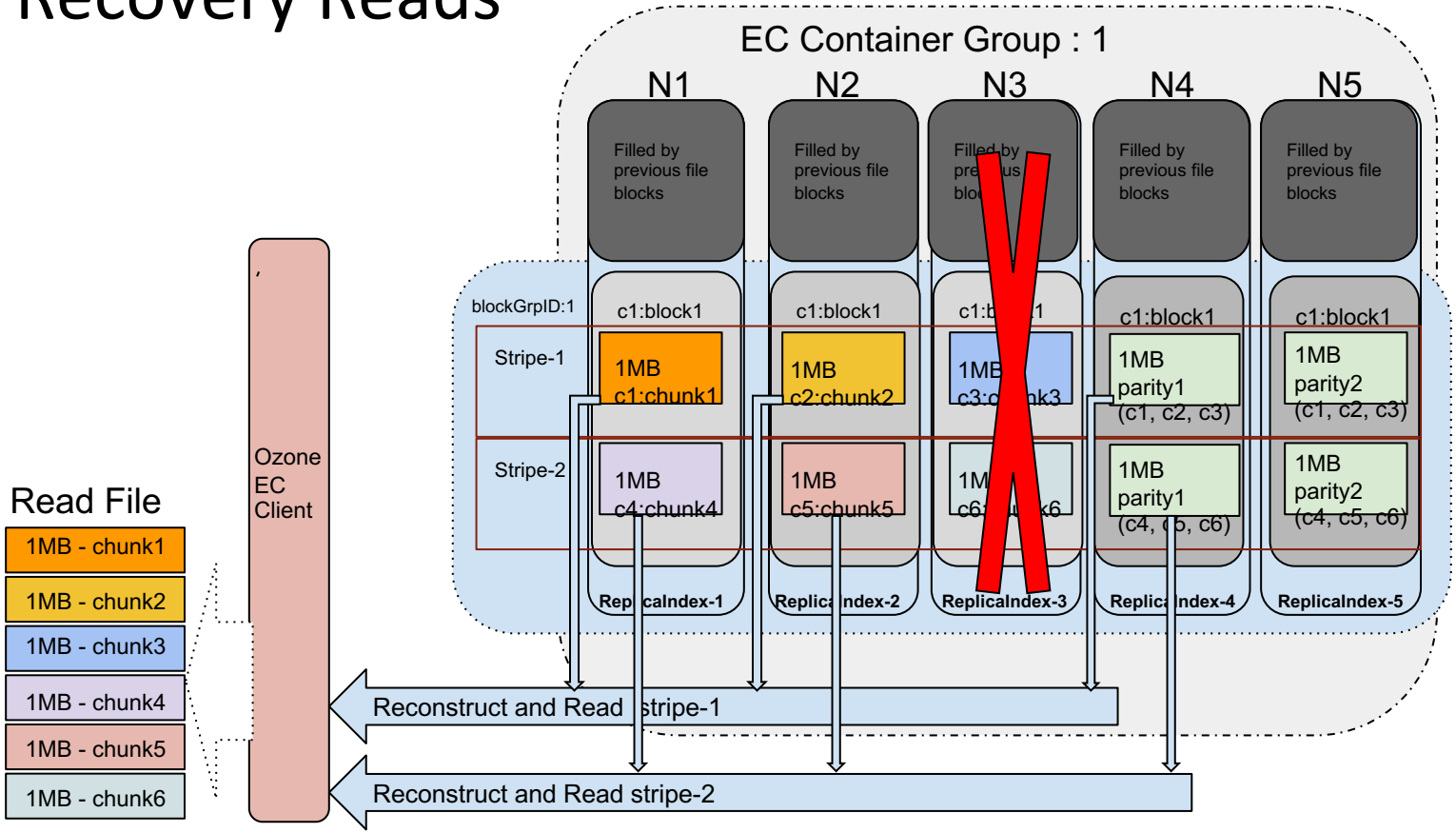
CLOUDERA

APACHECON North America

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



Offline Recovery

What is the Offline Recovery?

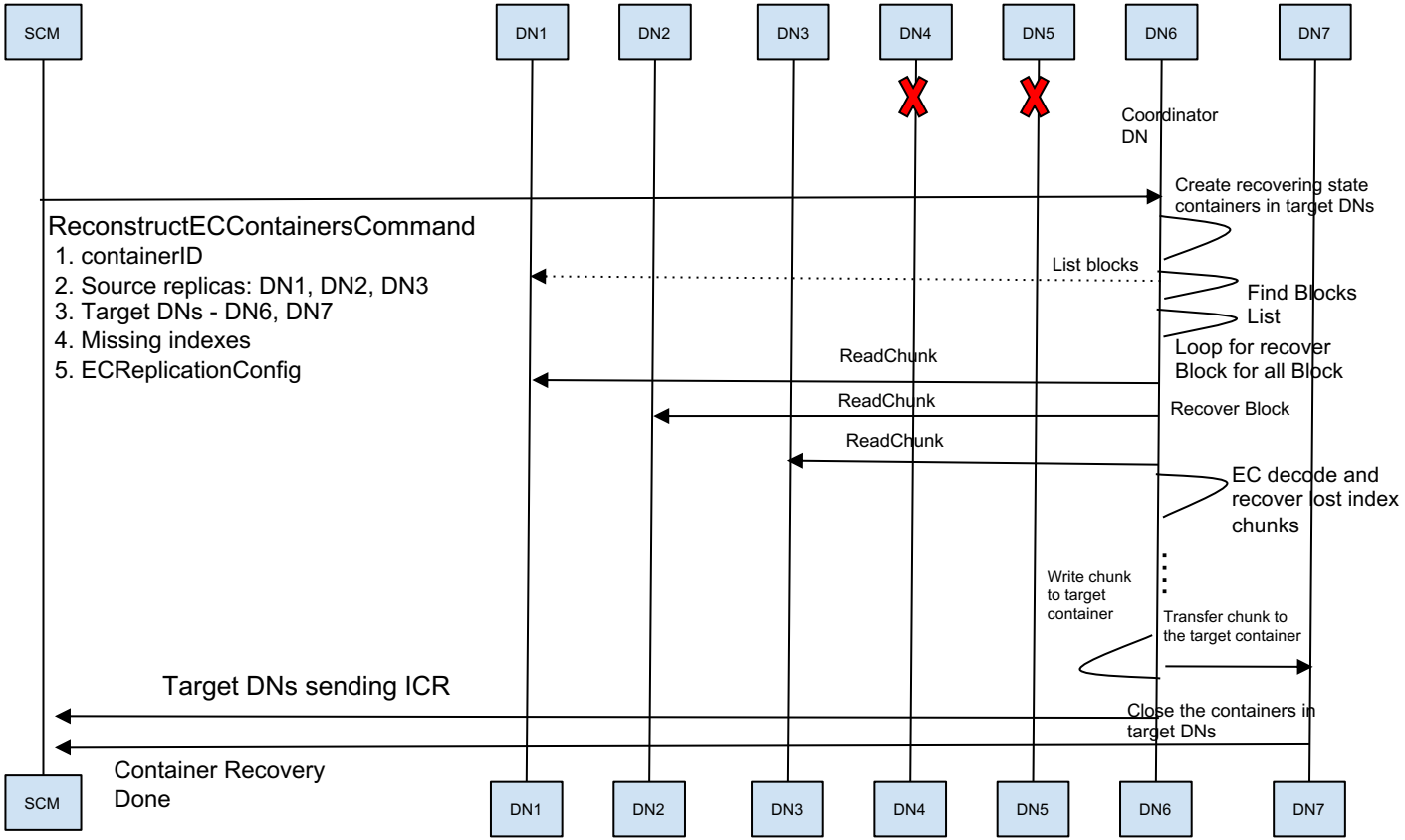
- When a node/Disk lost, we will lose the containers which are residing in that node/disk. We need a mechanism to recover that lost containers in the background. We call this process of background recovery as “Offline Recovery”.
- This is very critical background task similar re-replication on node/disk 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



CLOUDERA

APACHECON North America

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>
```

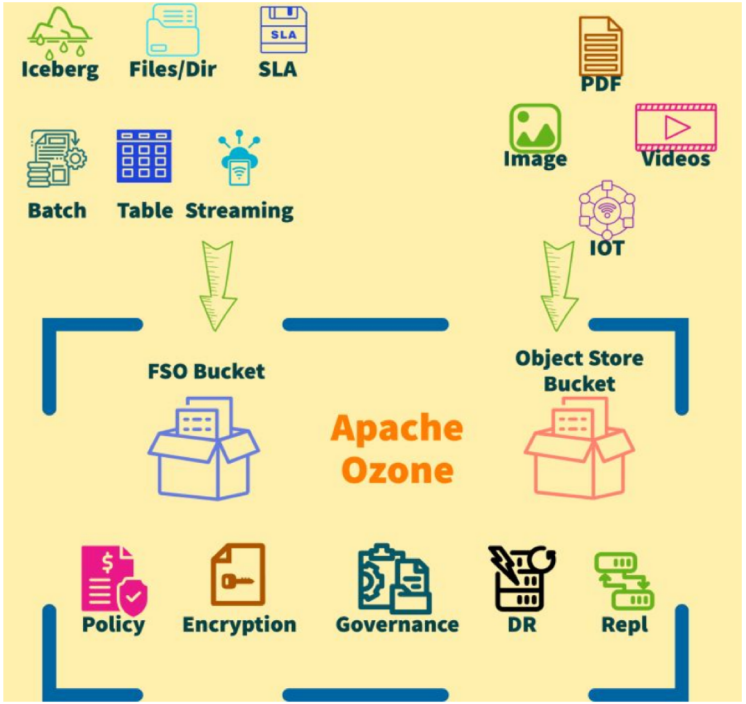
Enabling at Bucket Level EC

- 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.

```
/**
 * 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,
    short 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.

```
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)

Project Status

Phase -I

HDDS-3816 - ~ **140 JIRAs Resolved**
HDDS-5351 - **12 JIRAs Resolved**

Phase - II

HDDS-6462
~ 75 JIRAS Resolved

MVP

Phase - III

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