OZONE UPGRADES IN AN HDFS CLUSTER

In-place upgrades for Ozone.

Asher Chaudry asher.chaudhry@target.com

Craig Condit craig.condit@target.com

Mohammad J Khan mohammad.j.khan@target.com

Mathew Sharp matthew.sharp@target.com

Marton Elek elek@apache.org Anu Engineer aengineer@apache.org

Xiaoyu Yao xiao@apache.org *

CONTENTS

* Thanks to Jitendra Pandey, Arpit Agarwal, Ajay Kumar, Nandakumar, Mukul Kumar, Shane Kumpf and Tsz-Wo for the discussions and comments.

LIST OF FIGURES

LIST OF TABLES

ABSTRACT

Many HDFS clusters would like to upgrade to Ozone to take advantage of Ozone's scale and reduce metadata pressure on Namenode. This requirement means that after the upgrade to Ozone, HDFS data will be available in Ozone. Since both Ozone and HDFS share the same data nodes, it is possible to do upgrades to Ozone without large data copies and with zero downtime for HDFS. This document proposes an in-place upgrade design for Ozone.

1 INTRODUCTION

Ozone, an object store for big data applications is designed to work concurrently with HDFS. That is, existing HDFS clusters can be upgraded to have both HDFS and Ozone. Working with a mature, successful storage system like HDFS in a production cluster is a huge responsibility. Ozone's guiding principle has always been *first, to do no harm.* This led to many design decisions in Ozone, where we have tried to make sure Ozone never interferes with HDFS; Upgrade is one place where Ozone has a strong and tight coupling with HDFS.

HDFS users are primarily interested in Ozone because it addresses the scale issues of HDFS. Ozone has several magnitudes of scale and can handle

billions of files and block objects. Having Ozone running along with HDFS allows reduction of metadata for the namenode, without the users of the cluster feeling that they were forced to remove data.

With Ozone, an HDFS user can move data into Ozone and remove that data from HDFS. Ozone is a Hadoop compatible file system, hence applications like Apache Spark, Apache YARN, and Apache Hive, can access data by merely replacing *hdfs://<path>* with *o3fs://<path>*. In short, Applications need not be aware of the change in the storage layer.

As discussed already, with an upgrade, most users would like to see existing HDFS data appear magically inside Ozone. Most users do not want to run an application like DistCp to copy data from HDFS to Ozone. In many cases, it is not even possible. This *upgrade to Ozone with HDFS data* is called an **in-place upgrade**.

2 UNDERSTANDING HDFS DATA AND METADATA

Since we are dealing with the HDFS upgrades, a quick primer on how HDFS stores its data and metadata will be useful. HDFS maintains a map between a file name and the set of blocks that hold the data for that file as illustrated in 1.

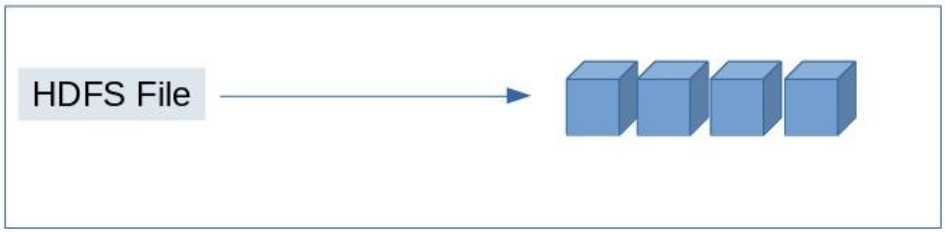

Figure 1: Block Mapping inside Namenode.

FSImage maintains this mapping between the file name and blocks. FSImage is the master store for all metadata including file system objects like files and directories, blocks, and ACLs. Only missing information is the block locations. Namenode learns the location of the blocks from the block reports sent by the data nodes.

To recap, FSImage tells us which files have which blocks, and the block reports from data nodes help Namenode learn the location of these blocks.

3 UNDERSTANDING OZONE DATA AND METADATA

Ozone maintains a similar map like HDFS, A map of Ozone key to the list of blocks is managed by the Ozone Manager(OM). All the information that is part of FSImage maps directly to the OM metadata as illustrated in 2.

The blocking mapping part where Namenode learns from block reports is very different in Ozone. Ozone relies on a block layer, called Hadoop Dis-

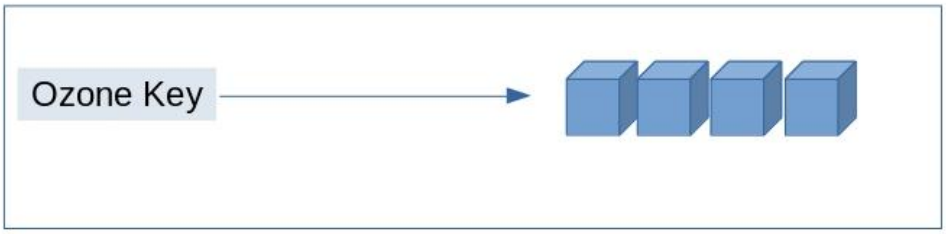

Figure 2: Block Mapping inside Ozone Manager.

tributed Data Store (HDDS). Storage container manager (SCM) offers blocks as a service to namespace managers like Ozone Manager(OM).

Under Ozone, Data nodes talk to SCM and not to Ozone Manager, and instead of block reports, data nodes send container reports. A container is merely a collection of blocks, along with some metadata that describes the blocks. This allows Ozone datanodes to report the presence of blocks at a higher granularity than HDFS, by sending in container reports instead of block reports.

3.1 Ozone Containers

HDDS allows co-existence of different kinds of containers on the data node. The default ozone container is described in figure 3.

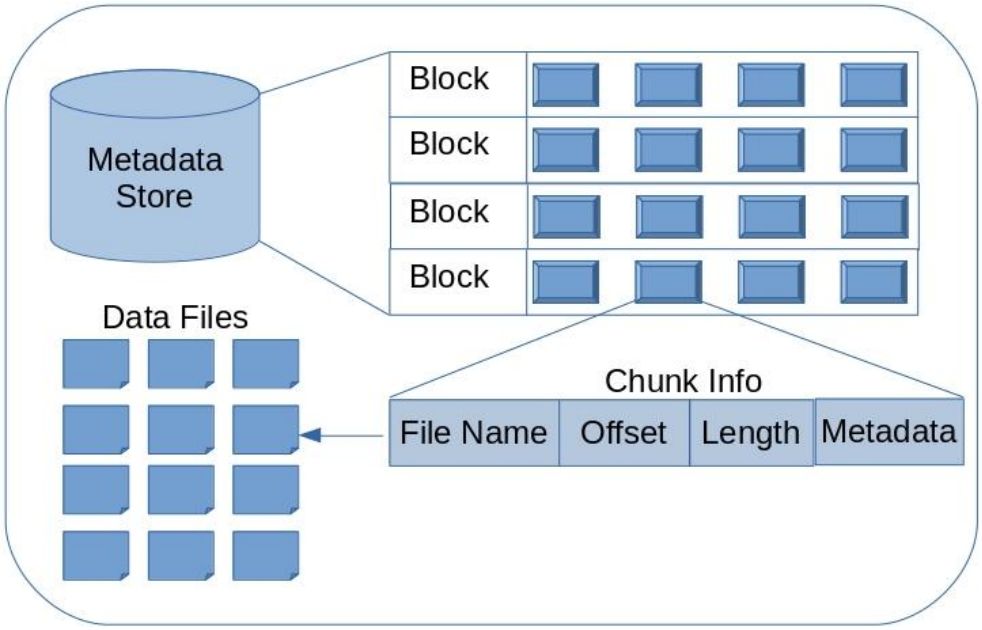

Figure 3: Ozone Container.

The Metadata store is an LSM tree that maintains the list of blocks in the container. Each block, in turn, points to a list of *Chunk Info* structures. The chunk info structure points to the actual data file where this block is stored. This gives Ozone containers the ability to be independent of the physical format for containers since each container is fully self-describing.

This allows Ozone to map a block to multiple physical files if needed and vice-versa; many blocks can be allocated in a single physical file if required.

4 NAMESPACE MAPPING PROBLEM

Given an HDFS path, the users can map it in any number of ways. The ozone upgrade planner(a tool proposed later in this document) will support the following mappings out of the box.

1. **Direct Mapping** - A path like *"/user/hive/data/Jan-2019/data.orc"* can be mapped directly to an Ozone volume, Ozone bucket, and Ozone Key; in this case the first component of the path will become the Ozone volume, that is *"/user"* would be the Ozone volume name. The *"/hive"* component would be the Ozone bucket and the rest of the path will be the key name, namely *"/data/Jan-2019/data.orc"*.

2. **A Cluster to Bucket Mapping** - Another approach is to move an HDFS cluster into an Ozone bucket directly. A bucket mapping like this has an advantage that many operations like OzoneFS are just a one-to-one mapping by setting up the defaultFS. In this model, the HDFS cluster root would map into a bucket and all path names in the HDFS cluster would be proper keys.

3. **A Partial Namespace Mapping** - Some times the users would want to upgrade to Ozone, but only map a part of the HDFS cluster into Ozone. Say an HDFS cluster is struggling with small file problem and all files which six months or older are in a path called "/archive". In this case, it would make sense to map only HDFS files which are under the directory called "/archive".

ENCRYPTION ZONES The current release of Ozone supports encryption zones, which are very similar to HDFS in architecture, we will be able to do a one-to-one mapping and use the same KMS keys for the HDFS files as well as the Ozone keys. The design document for the TDE support for Ozone is available here.

ACLS & APACHE RANGER The current release(0.4.0) of Ozone supports Apache Ranger, so when the migration is underway, we will be able to read the current Ranger policies and apply them to the user-provided path under Ozone. The Policy APIs of Ranger allows us to read and create new policies for Ranger. Apache Ranger supports a notion called Resource paths for an HDFS path. Policies can be applied to any resource path. Under ozone, HDFS resource path will be mapped to Volume, Bucket and a Key. To me more specific, suppose we have a Ranger policy on /hive/data, and we map that path for ozone under /myvolume/mycluster/hive/data. Then Ozone policy will be applied to Ranger objects of *volume=myvolume*, *bucket=mybucket* and *key=/hive/data*.In short, depending on the mapping adopted, we will parse the resource path and map it to the corresponding volume, bucket and key mapping under Apache Ranger.

With Apache Ranger, An HDFS access will look up Ranger and in the absence of Ranger policies will enforce HDFS ACLs. This means that Ozone must have an ACL engine which is capable to supporting HDFS ACLs. This native Ozone ACL engine will be part of next release of Ozone(0.4.1).

ERASURE CODING Ozone does not support erasure coding yet; this means that files which are erasure coded in HDFS will have to be moved to Ozone via a copy operation like DistCp. Fortunately, EC is not deployed widely in the HDFS world, so it is entirely possible to upgrade and ask users to copy EC coded files via DistCp.

HDFS STORAGE POLICIES This will be ignored for the time being. Ozone does not have the notion of Storage policies, hence will not honor them.

QUOTAS HDFS supports name quotas and space quotas. Ozone at this point only support space quotas. The rationale being that namespace is not a scarce resource anymore. The space quotas are very different under ozone since they are applicable only on volumes. Under HDFS quotas are possible at directory level. Ozone does not have the notion of directories; hence quotas will be ignored during upgrade and administrators will have to manually apply quotas as needed.

SNAPSHOTS HDFS has a notion of snapshots, in the first version of Ozone upgrade, we will ignore snapshots. In the future versions, Ozone can be upgraded to a specific snapshot instead of the current state of the cluster.

5 OZONE NAME MAPPING – PROPOSED SOLUTION

To accommodate these different forms of Namespace mapping, the proposal is to create a simple mapping tool(Ozone upgrade Planner). That is for each path in FSImage; we will invoke a function that will provide the Ozone name of that HDFS Key.

By default, we will support three different mapping schemes, the direct mapping, the bucket mapping and the partial mapping , the administrator can chose one of these ready made mapping solutions, or can provide a custom mapper[1].

For files that are not supported, like EC encoded files, the upgrade process will generate a DistCp source file which can be passed to DistCp to copy files from HDFS to Ozone.

6 OZONE BLOCK PACKING PROBLEM

As discussed earlier, Upgrade of an HDFS cluster should leave the system with HDFS and Ozone running concurrently, each managing its data.

1 A custom mapper will be a script or a JAR that given an HDFS file path, returns the Ozone path.

This co-existence of services leads to an interesting problem; of mapping data in HDFS into Ozone without moving or copying blocks. Ozone tries to pack many HDFS blocks into a large "super" block kind of structure called storage containers. This means that we can create an *efficient*[2] ozone storage container only if a large set HDFS blocks share the same set of data nodes.

It might be useful to illustrate this problem with an example, suppose we have 150 million files and 150 million blocks in an HDFS cluster. This size is used as a representative number for the HDFS cluster in this paper.

Under Ozone, namespace is partially loaded into memory, so 150 million files is not an issue at all. The block space, unfortunately, needs to be resident in memory. The trick that Ozone uses to keep all the block metadata in memory is the notion of Storage Containers. A Storage Container is a collection of blocks that are treated as a unit of replication. This container abstraction allows SCM to keep a minimal amount of block mapping in memory and that allows the system to scale.

OZONE BLOCKS With storage containers, the Ozone blocks look slightly different from HDFS Blocks. An Ozone block ID is composed of a container ID and local block ID. The local block ID is called the local ID. Since the local ID is unique only within the container. Each Ozone block is unique, but that uniqueness comes primarily from the container ID. The figure 4 explains how an Ozone block looks like from the SCM's perspective. The container ID is used to locate the container where this block lives. This container to data node mapping is very similar to the HDFS; the Namenode learns the location of blocks from block reports; SCM learns the location of containers from container reports.

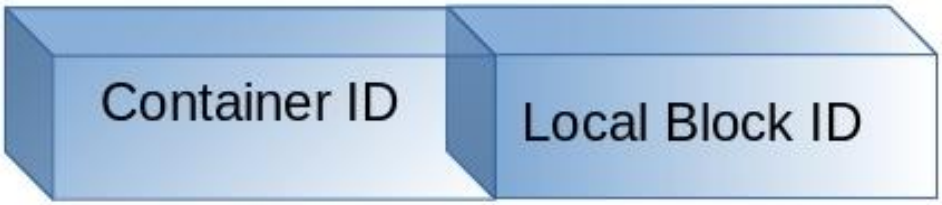

Figure 4: Ozone Block ID.

This container abstraction means that SCM keeps tracks of blocks at the container level, and the default size SCM uses is 5GB. To create a container with a 5GB size, we need 40 blocks of 128 MB each in a container.

When thinking about packing HDFS blocks into Ozone containers, the physical layout of the existing cluster is another constraint. *We need to pick HDFS blocks that share 3 data nodes to pack it into a container*.

HDFS BLOCK PLACEMENT HDFS uses a default block placement strategy called *Random Replication* which is not completely random. The first node is chosen randomly, the second block is replicated onto a different rack, and the third block is replicated onto the same rack as second block.

2 efficient here means that many HDFS blocks are packed into a single container, thus reducing the block metadata required.

If we assume a random distribution (to make the math simpler) and assume we have a nine node cluster, then we need around 84 blocks in the cluster before we start seeing overlaps for sure. That is, there are 84 independent ways to lay out a three-way replica on nine data nodes. Once we reach the 85th block, we are guaranteed that it has to reuse one of the earlier combinations of three data nodes. That is what we mean by *overlap for sure*[3]

$$C_k(n) = \binom{n}{k} = \frac{n!}{k!(n!-k!)}$$
$$C_3(9) = \binom{9}{3} = \frac{9!}{3!(9-3)!} = \frac{9\times8\times7}{3\times2\times1} = 84$$

In the default case[4], Ozone is looking for 40 blocks overlapping in a set of three data nodes. To achieve the metadata density of 40 blocks, we will need around $84 * 40 = 3360$ blocks in the system. For tiny clusters like nine node clusters, it is possible to get to this block-to-data-node ratio. However, as the cluster becomes larger, for example, a 1000 node cluster, the same math for getting overlapping blocks on three data nodes now needs 160 million blocks. It would be impossible for us to get to the 40 overlapping blocks on each set of three data nodes, if the blocks were randomly distributed in the cluster.

$$C_3(1000) = \binom{1000}{3} = \frac{1000!}{3!(1000-3)!} = \frac{1000\times999\times998}{3\times2\times1} = 166167000$$

In other words, with a 1000 node cluster, *it is entirely possible to have Ozone block containers with a single HDFS block after upgrade or the Ozone container to HDFS block ratio to be 1*.

This 1:1 ratio by itself is not as bad as it sounds since SCM is designed to handle millions and millions of containers. However, it does not allow the user to achieve the full-potential of reduced metadata for an Ozone cluster when the user is upgrading from an HDFS cluster.

BALANCER At this point, we can wonder if the random placement model should be the model used for the distribution of blocks. The HDFS admins could be running balancer, and balancer does not obey the first node, different rack, same rack algorithm. So given enough time with balancer running in the background, an HDFS cluster will have extremely good and even distribution of the blocks, which is what we want the balancer to do. However, closest we can model the behavior of balancer is by assuming random replication.

6.1 Validating the Model

Models are good in theory and helps to think clearly about the problem. It is always a good idea to validate a model with real world data. So we ran a study against a production cluster. The cluster under analysis has more than 150 million blocks, and thousands of jobs are run against this cluster each day.

3 It is certainly possible to see overlaps before reaching 85th block, but at the 85th block we must have an overlap.

4 because we assume that a storage container size is 5 GB and HDFS blocks are 128 MB.

DEFINITIONS To understand this study we need to define some common terms. A set of three data nodes that share the instance of a specific block replica is called a *Replica Set*. That is, if we have block 1 on data nodes A, B, and C, then the set {A, B, C} is the replica set for block 1.

During this study we have ignored all blocks that are not exactly 3 copies, that is blocks which are over-replicated and under-replicated were ignored for the purposes of this study.

First, we counted the number of blocks per replica set. That is we took all the blocks in the cluster and found out on which data nodes they existed. Then we grouped the blocks by replica set and counted how many blocks existed per replica set. Table 1 is the result of that study.

	50 Million Blocks	100 Million Blocks	137 Million Blocks
Average Blocks. in R.Set.	3.839505	5.17598	6.83531
Min Blocks. in R.Set	1	1	1
std. dev. in R.Set	9.585126	13.983161	16.716041

Table 1: Replic Set Data

In figure 5 we can see that data seems to follow the pattern that our model suggested. From the data, we see that at 50 million blocks processed, we have 3.83 HDFS blocks per Ozone container, at 100 million we have 5 and at 137 million blocks processed, we get a block density of 7. The reality is slightly better than our model, since our model was computing the worse case scenario.

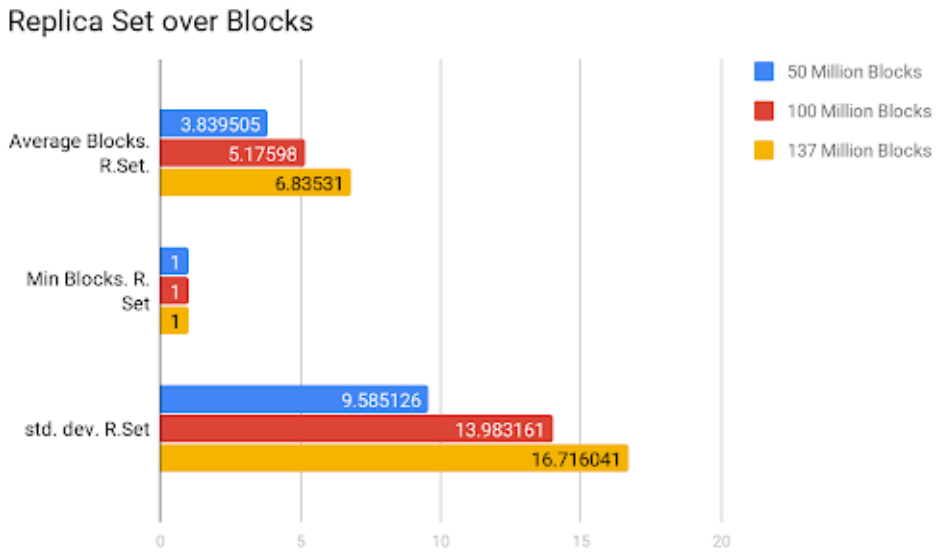

Figure 5: Replica Set Stats

Two interesting facts stand out for us, the Minimum Blocks in Replica Set is **one**, even when all the blocks have been processed [5]. This number means that we will have ozone containers which map to single blocks of HDFS. Second is that the standard deviation is very high, and we investigated this

5 137 million blocks since we ignored blocks that are not exactly 3 replicas in this study.

in depth and found there are replica sets with an incredibly high number of blocks. We saw some replica sets which had more than 1000 blocks. We know what is contributing to the high standard deviation values; we did not investigate why.

The most important conclusion is that in a cluster like this the best case container packing we can achieve is

$$7 \times 128\text{MB} = \textbf{896 MB}$$

We also found that the average block size for this cluster was **90 MB**; hence the size of ozone containers in the average case would be

$$7 \times 90\text{MB} = \textbf{630 MB}$$

.

The 7x reduction in metadata means that SCM would see only 20 million containers instead of the 137 million blocks after the upgrade. So even with no data movement, we will be able to achieve considerable block metadata compression.

Given this data, the next question we asked was how can we improve upon this, since the best case scenario for this cluster from Ozone point of view is around 2.5 million containers[6], which means we do have an 8x improvement left on the table.

7 SINGLE BLOCK MOVEMENT SIMULATOR

To create better packing densities, we tried to simulate what happens if we move one single block? That is, if we have three copies of a block, what happens if we move one single copy of the block, How much does that improve the ozone container density and what are the trade-offs? To answer this question, we ended up building a block movement simulator.

The block movement simulator solved this by creating a list of blocks in the cluster and finding the same prefix. Let us say we have block-1 on data nodes {D1, D2, D3}, and block-2 on {D1, D2, D4}, and this means that we can move block-2 from data node D4 to data node D3 and it will allow us to create an ozone container with two blocks (blocks 1 and 2).

Since the block-1 and block-2 have a common prefix {D1, D2}, we are looking for blocks that share common prefixes in the cluster. This common prefix is called the *leading key* in the simulator.

The *leading key* is computed as follows: Since the default HDFS algorithm picks the first node randomly and the second node on a different rack, and the third replica on the same rack as the second block, we assume that nodes with the same rack id, is the leading nodes[7]. To make sure that we get consistent leading keys, we group the data nodes with the same rack id and declare them to be the nodes in the leading key. If we find no common rack id, (the balancer effect perhaps), then we try to sort the nodes by data node

6 (137 million * 90 MB) / (5 GB) = Two million four hundred sixty-six thousand

7 They should really be called trailing nodes, but we take some poetic license and view the graph from last two nodes and look at the first node as tail.

id and treat the first two nodes as the leading key [8]. Only **invariant** needed by us is to get the same leading key for a given HDFS block it really does not matter how we calculate the leading key.

The figure 6 is a graphical view of what the leading key looks like, for each data node pair we have a set of data nodes and the corresponding block counts along with the blocks in that ReplicaSet. In this figure 6, the data nodes {D1, D2 } are the leading keys, and the third data node(s) has the corresponding ReplicaSet count. This allows us to move a single block between the leaf nodes to create better block layout for Ozone.

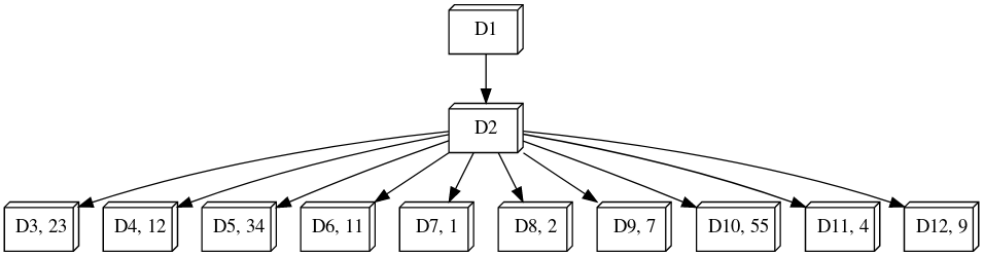

Figure 6: Replica Sets with Leading Key

8 SINGLE BLOCK MOVEMENT RESULTS

The simulator with a single block move can achieve rapid convergence to the ozone's required levels of block packing. To quantify the gains we ran the simulator and computed the inflection points in the system. The first metric we looked at was how many moves are needed to eliminate single HDFS block containers.

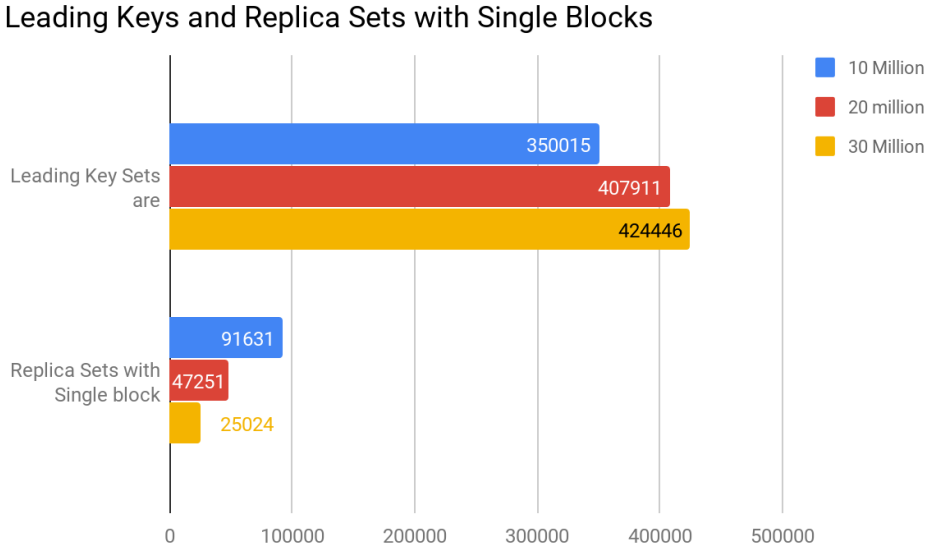

Figure 7: Leading Keys with Single block Replica Sets

8 This does have the issue of skewing the data movement in favor of lower numbered data nodes.

From the figure 7 we can see that when we process 10 million blocks, we can find 350 thousand leading keys (or common data prefixes) and 91 thousand HDFS blocks are still in unique replica sets.

The leading key count remains stable as we process more data; this is because large number of prefixes are already discovered. We also see that Replica sets with single blocks are steadily reducing in the count. With 30 million blocks processed, if we move 25K blocks, it will eliminate all single block containers.

We also looked at when we would be able to achieve the threshold of 40 blocks under a leading key. That number was met with very few blocks being processed from the set.

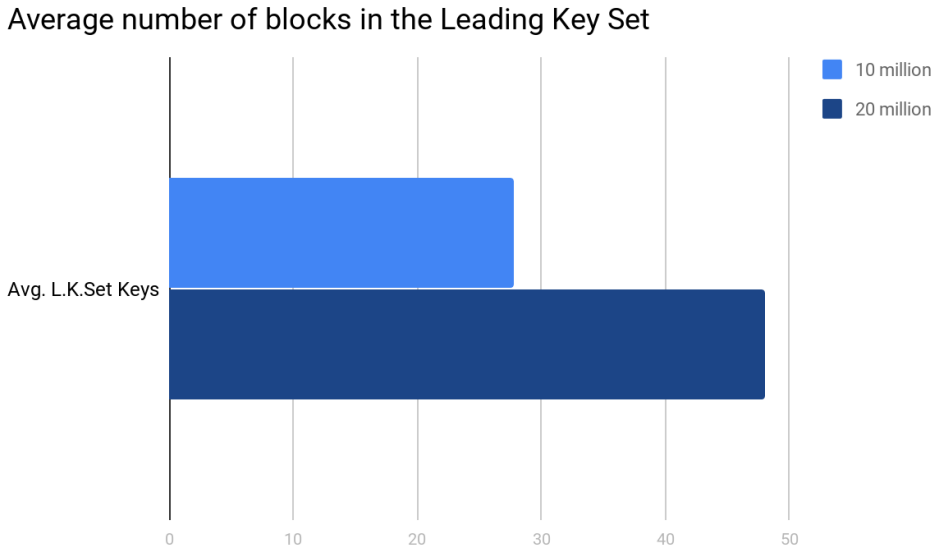

Figure 8: Average number of blocks under a leading set

The figure 8 counts how many blocks are under the leading set or the prefix key. This is the maximum number of blocks we can have in a container if we compact all blocks under that leading key. As soon as we processed 20 million blocks, we were able to achieve the average 40 blocks under a leading key.

SCATTERWIDTH TRADE-OFF This consolidation of blocks on a replica set cannot be uncontrolled. The problem is that when a machine in a replica set fails, we need to have enough data nodes to reconstruct data from. The copyset paper [1] and the work done in facebook indicate that controlling the scatter width and moving away from purely random placement yields better resiliency for the HDFS cluster. However, each move from the leaf nodes to other leaf nodes does reduce the number of source data nodes that we can use for data node reconstruction in case of failure. For small clusters, the probability of ozone upgrade needing any data moves it very small[9]. After all, if we have small numbers of data nodes, and a small number of blocks will give us the required density. When the number of data nodes is large,

9 Remember the 9 node with 3360 blocks.

the copy set paper argues that placing the blocks in a controlled pool of data nodes is better for systems overall resiliency.

9 OZONE UPGRADE – REQUIREMENTS

Now that we have discussed the problems to be solved, we can look at the core part of this paper, the ozone upgrade problem. We will define the requirements, an algorithm that does the HDFS to Ozone upgrade in the rest of the document. The core set of requirements are:

1. **Understand the current cluster layout and what it will look like post upgrade** - The ability to understand the current namespace and block layout and what it will look like post upgrade.
2. **Minimum downtime for the HDFS cluster** - Ozone upgrade should not cause significant downtime for the HDFS operations. We want to reduce the impact of Ozone upgrades on existing clusters and make introduction of Ozone to an existing HDFS cluster transparent for users.
3. **Optimum data layout** -Provide visibility and control over the HDFS/Ozone cluster, so that users can decide if they would like Ozone upgrade to move data before the upgrade.
4. **Ability to control source path to destination paths** - As discussed already, the user should be able to decide how the HDFS paths should appear in the Ozone namespace.
5. **Generate a DistCp file** - For files, where a direct linking or no copy approach does not work (say EC encoded files in HDFS) then a distCp file should be generated which can be used to move data from HDFS. Optionally, Ozone upgrade experience should be able to execute this data move via DistCp.

10 OZONE UPGRADE PLANNER — BEFORE YOU UPGRADE

The Ozone upgrade is a three-phase operation, which involves visualizing the current cluster, planning how the upgraded cluster should look like and then actually executing the upgrade. The figure 9 shows the different phases of the Ozone upgrade planner.

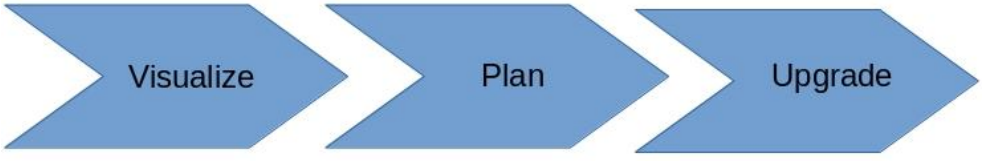

Figure 9: Ozone Upgrade Planner Flow

The Ozone upgrade planner will read the FSImage, connect to the namenode and create a block-to-data-node map. This block-to-data-node map

will be used by the ozone upgrade planner to compute the statistics of the HDFS cluster and will offer the user the ability to control the ozone block placement and namespace mapping.

The user will be presented with three knobs that user can use to plan the cluster layout; the amount of data to move (both in terms of GBs and in number of blocks) to achieve better block metadata densities, the minimum scatter width to be maintained and the namespace mapping algorithm to be used.

For example, if the user has a tiny cluster (in terms of data nodes), then ozone upgrade planner will inform the user about the container packing density, that after upgrade average size of Ozone containers is 2 GB; in that case the user may choose to continue without any data movement.

If the user has a large cluster, or if the container to HDFS block ratio remains at something like three blocks per container, the user might choose to schedule some block moves before the upgrade. In that case, the Ozone upgrade planner will create the HDFS move operations like the balancer and get them executed before the real upgrade begins [10].

11 OZONE UPGRADE ALGORITHM

This section discusses the actual upgrade from HDFS to Ozone. This section presumes that Ozone upgrade planner has been run and HDFS cluster is primed for an upgrade.

11.1 Information needed - FSImage and Block Map

The table 2 captures the information that we need from the HDFS/Namenode.

HDFS Information	Comment
FsImage	Ozone upgrade planner will connect to the Namenode and get a copy for FSImage.
Block-To-Node Mapping	Once we have the FSImage, Planner will connect to namenode and read the block locations and create a block-to-data-node mapping.

Table 2: HDFS Information needed for Ozone upgrade

With FSImage and Block Mapping from HDFS, Ozone upgrade has all the information needed for the upgrade [11].

10 The ozone upgrade planner is very close to the simulator we have now, a separate design document will be created for it.

11 Why not fsck and save the output to a file?. For all practical purposes, that is the all this information is. Fsck captured to a file is all we need. FSImage provides some more info, like ACLs, Erasure coded status of the block, Encryption zones, etc. which is useful during migration.

11.2 Static Mapping Algorithm

DEFINITIONS To recap, these are the definitions that we are using in the algorithm.

- **ReplicaSet** – A set of data nodes where the copy of a block exists. For example, if Block ID 1 exists on data node 1, data node 3 and data node 5, then

$$ReplicaSet(BlockID_1) \rightarrow \{DN_1, DN_3, DN_5\}$$

  In this discussion, we illustrate the ReplicaSet of a block as a set comprised of 3 data nodes, but this assumption is NOT a requirement for this algorithm. It is used to make the illustrations simple.

- **HDFS File** – An entry in the name node, we assume that Namenode metadata that we read from the FSImage is a consistent copy of the file system, as we proceed with the upgrade there might be changes to HDFS file system, we will ignore those changes as if Ozone upgrade works against a snapshot of HDFS.

- **Ozone Key** – A mapping of the HDFS name, whether it is a directory or a file to Ozone namespace. We will support an option to ignore empty directories. If that option is not specified, A blank key will be created to represent an empty directory in HDFS namespace. We can ignore empty directories without much loss to fidelity. If the user chooses not to ignore such directories, they will appear as keys inside Ozone bucket. An Ozone Key points to a set of HDDS Blocks, which is composed of ContainerID and Block ID.

- **HDDS Container ID** – A 64 bit ID that uniquely identifies a Container.

- **HDDS Block ID** – A 64 bit ID that is unique within a Container.

At the high level the Static Mapping Algorithm works like this:

- **Step 1** – For all HDFS Blocks, group them by data node prefixes and create ReplicaSets. This step allows us to consolidate the HDFS blocks into Ozone Containers. Group them into Ozone Containers, create HDFS Block ID to Ozone Block ID mapping.

- **Step 2** – Convert the FSImage into an Ozone Path, and assign Ozone File the corresponding Ozone block IDs.

- **Step 3** – Start SCM and Ozone Manager, instruct data nodes to create the containers.

11.3 Step 1: HDFS Block to Container Mapping

Depending on the output of the Ozone Planner and choices made by the user, this step can create a dense ozone container which has many HDFS blocks or a very sparse ozone container, perhaps containing only one HDFS block. In other words, any mapping at this stage is valid; all we care about is having a mapping function, including an identity or a one-to-one mapping.

Replica Set	Blocks in ReplicaSet
$RID_1(DN_1, DN_2, DN_3)$	$Blk_1, Blk_{23}, Blk_{24}, Blk_{32} \ldots$
$RID_2(DN_1, DN_2, DN_4)$	$Blk_2, Blk_4, Blk_9 \ldots$
$RID_3(DN_4, DN_5, DN_6)$	$Blk_3, Blk_7, Bkl_{10} \ldots$

Table 3: ReplicaSetTable

INPUT HDFS Namenode address, this process will connect to Namenode and read the Block Information.

1. *For Each block in the BlockToNodeMapping*
    a) *Compute the ReplicaSet for the HDFS block.*
    b) *Insert the Block and ReplicaSet to the ReplicaSetTable.*

At the end of this loop, we have a map of all HDFS blocks grouped by the Replica Set they belong to, this table is called a Replica Set Table. The table 3 shows the state at the end of this loop.

Now we can iterate over the Replica sets and find all the blocks and try to group them into containers. We keep track of the container size and consolidate HDFS blocks into different containers. The result is all the HDFS blocks which share the same data nodes land into one or more containers.

1. *For Each ReplicaSet(R) in the ReplicaSetTable*
    a) *For Each Block in R.getBlockList*
        i. *Assign a Container ID for ReplicaSet entry*
        ii. *Group the blocks into the same container using the container ID*
2. *Write the grouped information to a table.*

This table 4 will now look like this, where each HDFS block has a mapping to Ozone block ID, and they grouped via the data nodes where they exist. That is for each ReplicaSet; we have one or more unique containers.

HDFS Block ID	ReplicaSetID	Ozone/HDDS block ID
$Block_{200}$	RID : DN_1, DN_2, DN_3	CID:blockID (CID:1, LID:1)

Table 4: HDDS container Map

TESTING At this point, we can talk to Namenode to verify that this information is correct. `This is a crucial step in our process.` The fact that we can ascertain our upgrade in small self-contained steps will make it easy for us to develop, debug and deploy this feature quickly.

OUTPUT This step of the algorithm produces the HDDS Container Map, which is a map from HDFS block ID to Ozone Block ID(Container ID + Local Block ID).

11.4 Step 2: HDFS Files to Ozone Keys mapping

Now that we have created a logical mapping from HDFS blocks to Ozone blocks, we are ready to create the Namespace information, that is we can solve the issue of mapping the HDFS file name to the Ozone keys.

INPUT HDFS FSImage, HDDS Container Map (Generated by the previous step)

In this step, We will read the FSImage and create a corresponding mapping in Ozone Manager DB. To do this, we need two data structures, the FSImage from Name node and HDDS Container Map generated by the first step of this algorithm.

1. *For each File and Directory in FSImage*
    a) *Get the first path component and Create OzoneVolume if needed*
    b) *Get the second path component and Create OzoneBucket if needed*
    c) *Get the rest of the path, create an empty key if it is a directory*
    d) *if it is an HDFS File*
        i. *For each HDFS BlockID in HDFS File:*
        ii. *Lookup HDDSContainerMap for the ozone block ID.*
        iii. *Add the Ozone block ID to a temporary block list.*
        iv. *Write the ozone key with the temporary block List (which will have ozone block IDs)*

When this iteration is complete, we will have all the files and directories in the HDFS file system appear as valid Ozone volume, bucket, and keys. Each of the ozone keys will point to proper HDDS Blocks. For error handling purposes a reverse map of the block to HDFS file can also be maintained in the RocksDB. This will allow us to print out which files need correction/DistCp kind of approach.

TESTING Since this is also an offline operation, we can now check the output to verify that each and every key from HDFS is mapped correctly and there exists and valid mapping in the Ozone Manager. We can also test the HDDS block ID does not violate any constraints of our system.

OUTPUT This step creates the Ozone Manager DB or the namespace information in Ozone. This should be a mirror image of HDFS, based on the constraints passed to the Ozone.

11.5 Step 3: Bringing the System Online

In order to bring the system online, the Ozone planner would hand over the Ozone Manager DB (the namespace) and SCM DB (the block space) information to Ozone Manager and SCM.

Please note that SCM DB is not the SCM container DB. It is the HDDS Container Map DB (table 5 that we produced in step 2.

Repeating this here, so that you can see it is different from SCM DB.

HDFS Block ID	ReplicaSetID	Ozone/HDDS block ID
$Block_{200}$	RID : DN_1, DN_2, DN_3	CID:blockID (CID:1, LID:1)

Table 5: HDDS container Map

SCM in the upgrade mode will read this table and instruct data nodes to create the required containers. This step can take a while, hence SCM will report the progress separately for each data node. It will also report the failures that a tool like a Planner can read and report.

This approach makes it easy for us to work similarly in a secure or non-secure cluster.

Once the container construction is complete, Planner would verify that OM DB is consistent, via random sampling and once a statistically significant number of keys are found to be correct, it will request both OM and SCM to restart in Normal mode.

12 CHANGES NEEDED IN SCM, OZONE MANAGER AND DATA NODES

There is a set of change needed in Ozone for making this in-place upgrade possible. First and foremost, is the ability to communicate to data nodes and request the ability to construct "HDFS containers". The HDFS containers in principle look and feel the same as Ozone containers. The only difference is that instead of data being streamed into the data node, we make the container out of the data blocks already present in the data nodes.

DATA NODE To achieve this, we will need to augment the current data node to SCM protocol, we will need to be able to specify the ozone block ID and which HDFS block maps to that ozone ID. We will also need to create hard links in the ozone container namespace since we do not want an HDFS delete operation causing a loss of data in the ozone namespace. This is a mini-feature by itself, we have to handle the issue of errors while the containers are being created.

UPGRADE MODE Both OM and SCM need to understand they are being booted up in upgrade mode so that they will not trigger panic actions like massive replications until the upgrade is complete. Till the upgrade is complete, these components will see partial information in the cluster, which in the normal course of action, they would be tempted to correct.

SCM SCM needs code which can read the HDDS container DB data and send commands to data node via heartbeats. It also needs code to keep track of failure at a specific HDFS block level so that a tool like a Planner can attempt recovery or reconciliatory actions. The responses from the Data node, that is container reports will be used to create the actual container DB inside SCM.

13 HANDLING MULTIPLE BLOCK POOLS

Nothing prevents us from passing multiple name nodes which represent different namespaces but those that share the same data nodes. We will be able to process the multiple block pools and map them to the same Ozone server. It is just that both FSImage and Block processing needs to span both the namespaces. We will support this feature, but not may be part of the first release. There are no technical blockers for achieving this objective since the algorithm and tools are very similar.

14 POST UPGRADE BEHAVIOR

It is important to understand the post-upgrade behavior of both HDFS and Ozone. HDFS will operate exactly as before, and the user is free to add or delete files. However, the changes that the user makes to HDFS will have no impact on Ozone. Let us say a cluster was upgraded to use Ozone, after the upgrade the user is free to go to "/archive/data/" and delete all files. This delete action will remove files in HDFS, but the similar path under Ozone will have no change.

One interesting effect of this change is that if you are trying to remove the file to free up some space in the cluster, you will need to make sure that block object is freed in both HDFS and Ozone namespace. Ozone will provide a way of reporting which files in the Ozone namespace has a corresponding HDFS reference. This information will be available in the Recon Server (the management console for Ozone).

In short, after the upgrade Ozone and HDFS will appear as two different file systems. The I/O operations under one file system will be visible in the other file system.

14.1 Curious case of Appends

The append is a complex operation inside HDFS, including dealing with Generation Stamps and choosing the right version of the block. From the Ozone point of view, this does not impact us since we keep a hard link to the file and process that file. Even if a change is made to the file by appending, the Ozone block data maintains the offset and length of each data block. Therefore, a change to the end of a block file should not matter.

14.2 Moving HDFS blocks via Balancer

From the ozone's perspective, the move operation of a block is very similar to the delete. The physical file system will notice that this file has two hard links and will remove the HDFS path to the file; the actual block file will not be deleted. So running balancer against an upgraded Ozone system has the potential downside of creating data copies.

14.3 Guaranteeing the data is the same in Ozone and HDFS

Right after the upgrade, and also in a long time after that a general requirement is to make sure that data files in a given HDFS path and Ozone path are same and if not, what is the difference. Ozone upgrade will create a tool that can take two path HDFS and Ozone and compute this difference. This feature is useful for keeping ozone upto date with changes in HDFS.

14.4 Automatically listen to changes in HDFS

While it is effortless to listen to changes in HDFS via Journal nodes, that feature is not part of the Ozone upgrade. We might build that feature in the future.

14.5 Merging Containers after upgrade

Another way to achieve high-density ozone containers is to merge the Ozone containers after the upgrade. While this makes the upgrade almost effortless, the ozone will end up creating smaller containers and then later merge them. Both of these are extra I/O in the cluster. With the Ozone planner, these trade-offs can be understood before the upgrade and a lot of unnecessary I/O in the cluster can be avoided. Also, the merge operation might cause these blocks to move from the HDFS data nodes to other data nodes, which in -effect increases the data storage. So if the user does not care about the divergence in the block space, an upgrade of Ozone can be performed without looking at the HDFS state.

REFERENCES

[1] Asaf Cidon, Stephen Rumble, Ryan Stutsman, Sachin Katti, John Ousterhout, and Mendel Rosenblum. Copysets: Reducing the frequency of data loss in cloud storage. In *Presented as part of the 2013 USENIX Annual Technical Conference (USENIX ATC 13)*, pages 37–48, San Jose, CA, 2013. USENIX.