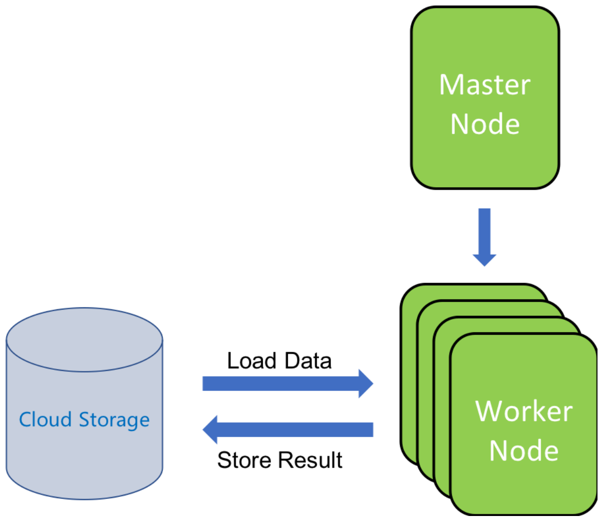
OZone - Rack Awareness Support

Authors: junping, xiaoyu, junjie, jitendra, anu, nicholas

Background and Problem Statement

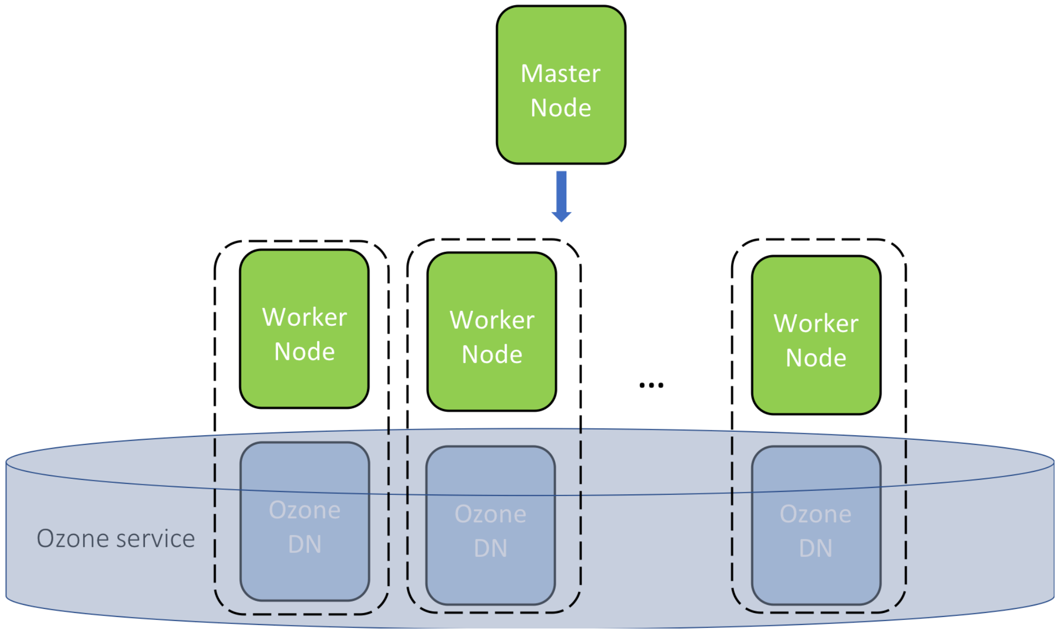
Since HDFS-7240, hadoop community tries to find a way to scale HDFS to support billions of files. Scaling the namespace layer and also the block layer is a proposed way which scales the block layer by grouping blocks into containers, thereby reducing the block-to-location map and also reducing the number of block reports and their processing. These efforts, combining with providing S3 like interface, now also know as Ozone project which target to build a native generic object store in Hadoop ecosystem.

In the era of cloud, we also notice more and more big data workloads are moving towards cloud. A typically cloud big data application workflow is: big data compute engines, such as MapReduce or Spark loads data from object store, then execute the job to process data, and store back result after job finish. Just like below
graph shows:



Through separating compute from storage, the compute cluster can be on demand provision or decommission so that we can have elastic big data services, such as EMR, etc. This practice of big data workloads in public cloud is becoming more and more popular as compute resource is more dynamic to share between tenants. However, the side-effort for this practice is we give up **data locality** which could have big impacts on performance for big data workloads.

Comparing with traditional object store service in cloud, Hadoop native object store - Ozone potentially can achieve the benefit of resource elasticity without trade off high IO performance, like below indicates:



Ozone service is deployed as a standalone object store service and can be accessed by all kinds of applications but not limited to hadoop/big data applications. In this way, it still have benefit of separating compute from storage, but at the same time, Hadoop worker nodes (Nodemanagers) can be deployed with Ozone Data Nodes back to back, so YARN can schedule tasks closed to data so no data locality get lost in this practice.

Thus, data locality, include rack awareness, is very important to ozone service and we could follow the same practice as hdfs implemented before.

Hadoop Topology

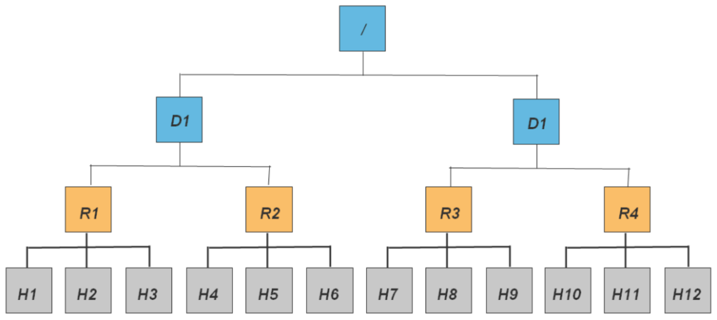
A Hadoop cluster architecture is based on a hierarchical topology with three-layers. The topology was introduced and documented in the issue HADOOP-692. The root node represents the cluster and the first level represents the data center. The second level represents racks/switches and the third level represents nodes where data and compute processing occurs. The data center level is not taken into account in the current code base.

To optimize the network bandwidth used when running Hadoop, the locality of the data is taken into account when scheduling a task to be run. Ideally the data and the compute task are co-located, but failing that placing data and compute within the same rack is preferred. <**Xiaoyu**: can you clarify the previous sentence?> A balance between reliability and bandwidth is taken into account with a replica placement model that is rack aware so that rack failure does not result in data loss. These concerns are modeled by calculating the network distance between nodes in the tree and knowing if two nodes are located on the same rack.

Proposed Solution

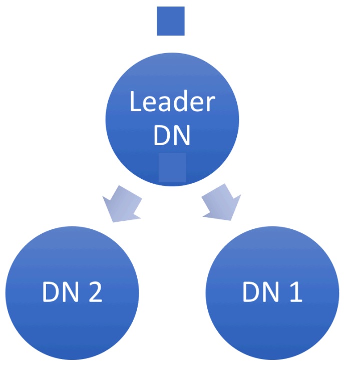
1. The OZone Manager, SCM should have way to understand network topology, just like hdfs taking a script to get rack location with given hostname.

Ozone manager should get rack id of the cluster slaves by invoking either an external script or java class as specified by configuration files. Following hdfs practice, output must adhere to the java *org.apache.hadoop.net.DNSToSwitchMapping* interface. To use the java class for topology mapping, the class name is specified by the net.topology.node.switch.mapping.impl parameter in the configuration file. With external script or java class to specify each node location, Ozone daemons should build a network topology (please refer *NetworkTopology.java*) like below:



1. Write process:

Ozone key write process can have two stages: 1. write key into a new open containers; 2. write key into existing containers.



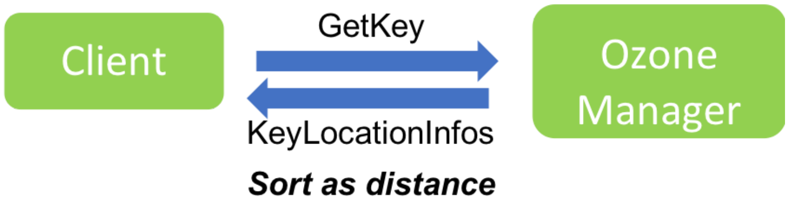
Make sure Leader DN, DN1 and DN2 are not on the same rack

To choose proper locations for replications, the selection of DNs in 1st stage - write key into new open containers , need to be carefully across different racks. so SCM (StorageContainerManager.java) 's ReplicationManager should have a new ContainerPlacementPolicy - previously it only randomly choose two node and pick up the less utilization one, but here it need to consider more to make replica robust enough to survive from rack failure.

pipeline - open container?
close container - full or failed.

<Xiaoyu: For the open containers, we will use RackAwarePlacementPolicy to create Ratis pipeline using 3 datanodes that are not on the same rack to avoid data loss in the event of rack failures. When container is closed because it is close to full or failed, the SCM ReplicationManager use the same RackAwarePlacementPolicy to choose 3 datanodes to replicate the closed containers.

1. Client read process



The flow of client read key also have two stages: 1. client get key locations from OM; 2. Client talk with DN to fetch block as chunks. As lacking of cluster information, client doesn't know which location are close to itself, so OM should return sorted (nearest first) location info so client can read block with lowest cost.

4. Other changes?

<Jitendra> 1) Need to understand how network topology script can work for containers. Is there a way to figure if two containers are on the same host?
An alternative approach could be to base the topology on storage volumes, i.e. schedule the replicas on storage volumes and then lookup which container is managing that volume.
However, I am ok with the HDFS like approach because that may the fastest to build.
2) An additional change needed would be an under-replicated queue based on priorities:
a) containers with only one replica get highest priority.
b) containers with two replica get a different priority
c) containers with 3 replica but don't satisfy rack policy get a different priority.
HDFS has these policies baked in.

<Junping> Discussed with Xiaoyu last time, we think a better solution could be a abstract & hierarchical topology support, such as layer 0, 1, 2 … etc. instead of fixed layers like: rack, node, etc. It can fit into more scenarios, like: cross data centers, etc. The policy can be customized to fit each case. The proposal above need some significant changes. @all, please comment here before we go ahead for more explorations.

JIRA to be open:
Abstract topology
M/R, YARN, Spark integration

Links:
HDDS-698. Support Topology Awareness for Ozone