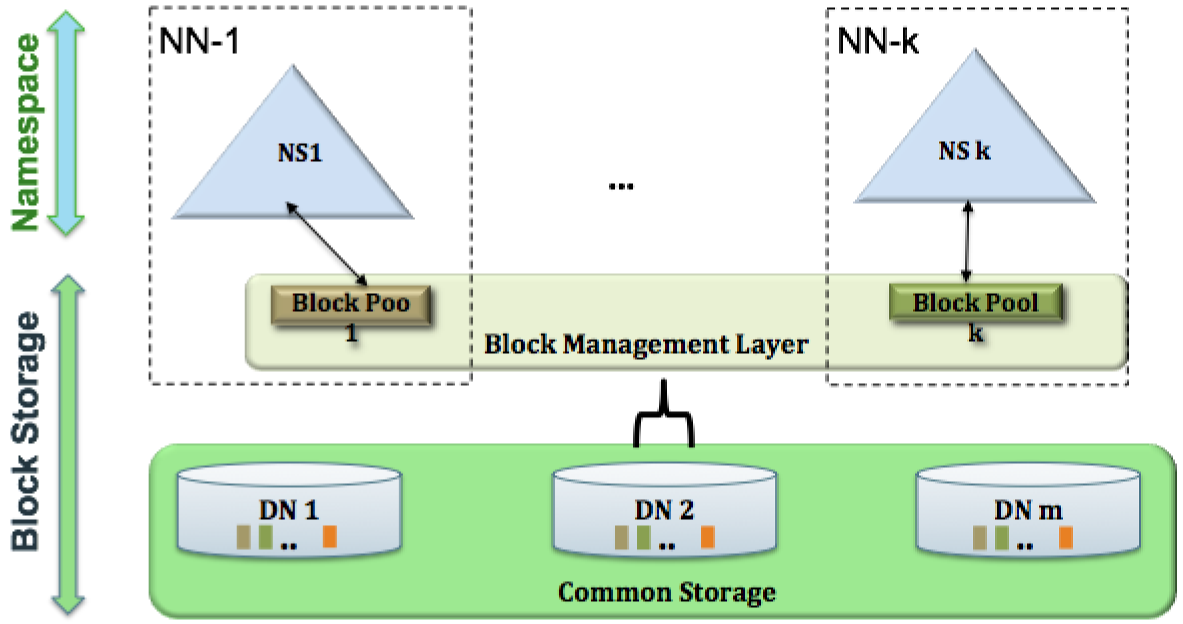
HDFS Scalability and Ozone

Summary

This document explains a design for scaling HDFS and how Ozone paves the way towards the full solution. Ozone's block-container layer scales the block layer of HDFS with the IO characteristics of HDFS's data layer. While Ozone's Key-value namespace is not as rich as HDFS's hierarchical namespace, it serves as an interim step; it is much simpler to implement and can be used via a Hadoop Compatible FS API that is commonly used by Hive and Spark on cloud object stores. In the past we have prototyped a Scalable HDFS namespace by simply caching a working set in memory; this can be completed in the future for a fuller solution as proposed in HDFS-10419*.*

HDFS is layered as follows:

- A namespace layer - implemented in the name node
- A block layer - implemented mostly in the Datanodes with block management module in the Namenode.



HDFS does the following very well:

- Scaling storage, IO, clients
  - Horizontal scaling – IO + PBs
  - Fast IO – scans and writes
  - Number of concurrent clients 60K to 100K++

        - But number of files & blocks is limited. Fault tolerant storage layer
- Locality
- Replicas/Reliability and parallelism
- Layering – Namespace layer and storage layer

HDFS has several key problems wrt. to scalability.

- Scaling namespace
- Scaling block namespace
- Scaling block reports
- Scaling Datanode's block management
- Scaling client/rpc load on NN

Ironically, keeping all the Namespace in memory is a strength that allows tens of thousands of clients to access the namespace in parallel. However, keeping all the Namespace in memory is weakness because it limits the scalability of the namespace.

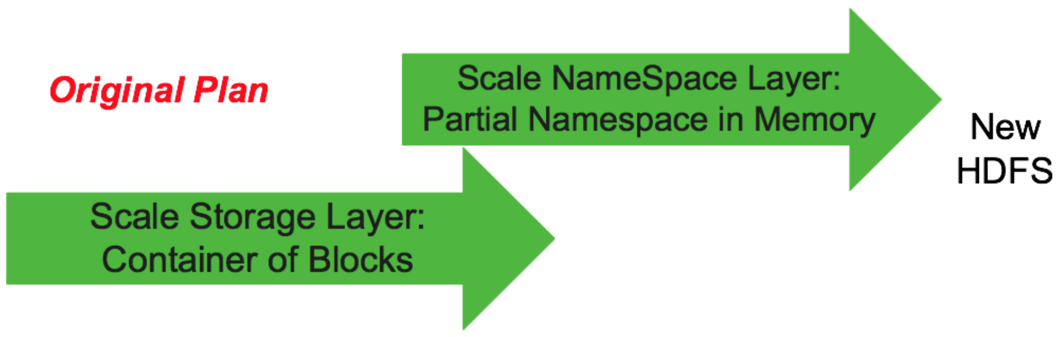
Solving all the limitation is a fair amount work. The Ozone storage system is step to evolving HDFS and solving some of these problems. We explain this below.

Containers of blocks Instead of keeping track of individual blocks we group blocks in to containers. A container is somewhere between 2GB to say 16GB. Block reports now become container reports. This approach solves the scaling of block namespace, scaling of block reports and scaling Datanodes' block management.

Keep only working-set of NameSpace in memory. Keeping only working set namespace in memory (as explained in Jira HDFS-5389, HUG - Removing Namenode's Limitation, Scalable Metadata Service in HDFS,) solves the scalability of Namespace . Independent studies have also analysed LRU caching of HDFS Metadata [Metadata Traces and Workload Models for Evaluating Big Storage Systems] This approach works because in spite of having large amounts of data (say data for the last five years) most of the data that is access is recent (say last 3-9 months); hence the working set can fit in memory.

Scaling with the client load can be addressed by having the NN better multi-threaded and or sharding the namespace.

Implementing both "**Containers of blocks"** and the "**Working set in memory NN**" is a fairly big effort. Ozone implements containers of blocks. Our original goal was to finish that and then do a Namenode with only the "Working set in memory"; Lin Xiao originally prototyped this as part of HDFS-5389 in HDFS 0.23.


Original Plan
Scale NameSpace Layer: Partial Namespace in Memory
Scale Storage Layer: Container of Blocks
New HDFS

Key-Value Namespace as an interim Solution,
Recent efforts to run Hadoop first class in the cloud have offered another opportunity. Spark has from early days focused on running on a key-value object stores using a Hadoop compatible FS (HCFS) API (ie either Hadoop's FileSystem or FileContext). The Hive community is taking a similar approach: run first class on an object store using the HCFS API. The lack of rename atomicity is being addressed in a number of ways:

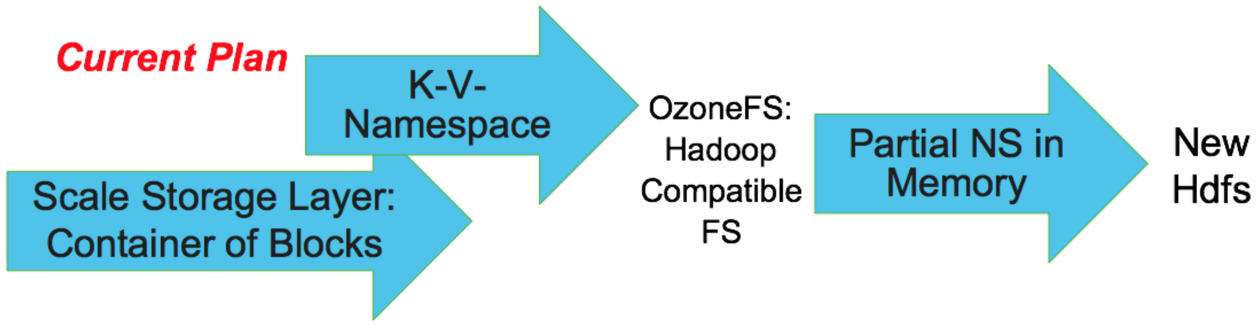
- Netflix's output committer
- Use a metastore - both Databricks's Spark and Hive is doing the same.

To summarize, due to the growth of cloud, Hadoop applications frameworks developers and application developers are realizing that it is critical that the applications frameworks and applications run first class on both HDFS and a Key-value object stores using HCFS API

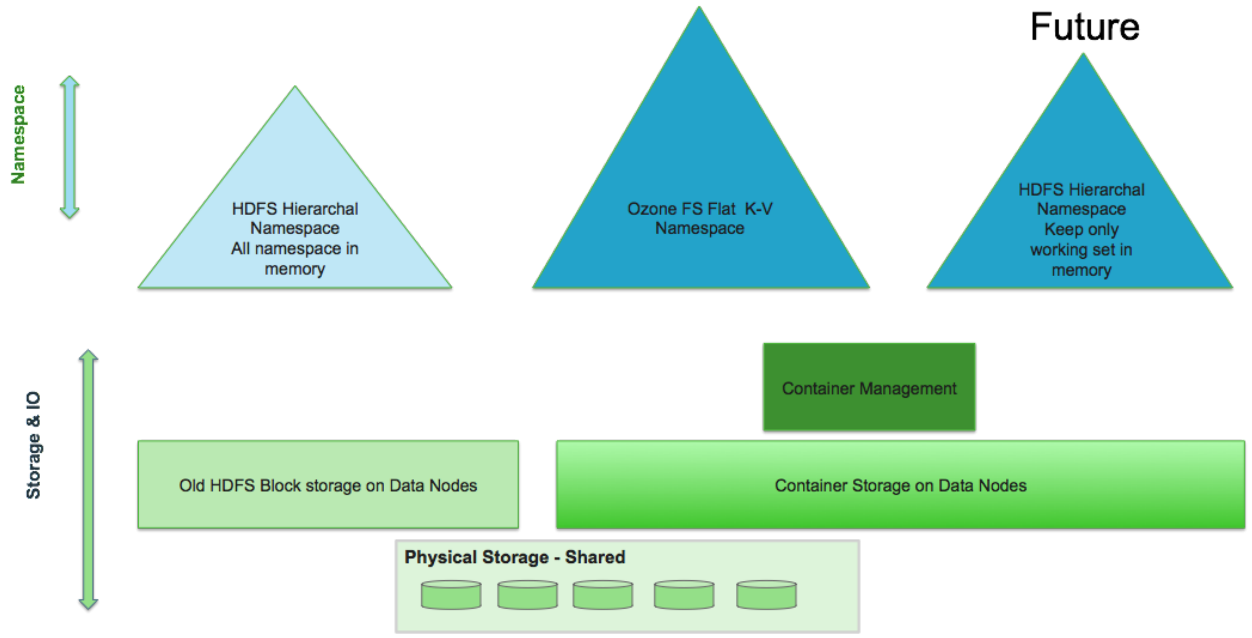
So we came to the following conclusion: lets implement, **as a first step**, the meta namespace as a flat key-value namespace. It much much simpler to implement than a scalable hierarchical namespace. **Later** we will implement a scalable hierarchical namespace as tie it to block-container layer as proposed in [HDFS-10419]. .

Ozone's Metadata server that manages the K-V namespace can scale much better than HDFS Namenode for the following reasons:

- It keeps only the working set of metadata in memory - this scales the Key-Value namespace (note this solution match the NN working set caching solution that was described above).
- Ozone does not need to keep the equivalent of the block-map; instead the corresponding container-to-location mapping resides in in the SCM - this leaves additional free space for caching the metadata.
- It does not have a single global lock like the Namenode - this will scale the number of concurrent clients/rpcs. Note this is historical limitation of current Namenode's ***implementation***.
- It can be partitioned/sharded much more easily because it is a flat namespace. Indeed a natural partitioning level is the bucket. Partitioning would further scale the number of clients/rpcs and also the Key-value namespace.


Current Plan
K-V-Namespace
Scale Storage Layer: Container of Blocks
OzoneFS: Hadoop Compatible FS
Partial NS in Memory
New Hdfs

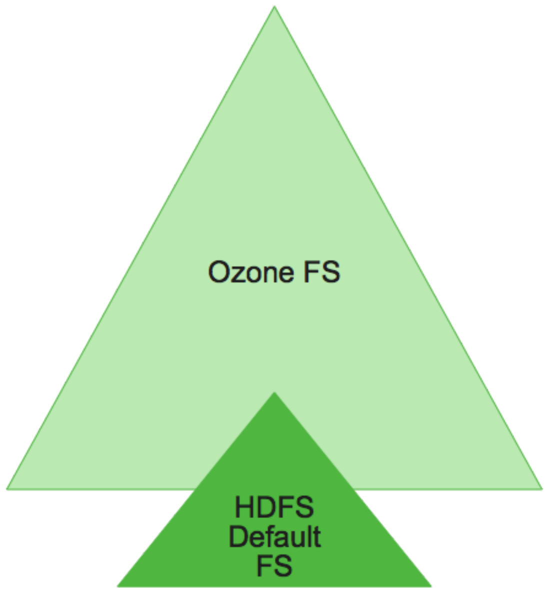
HDFS and Ozone components fit together as follows


Future
Namespace
HDFS Hierarchal Namespace All namespace in memory
Ozone FS Flat K-V Namespace
HDFS Hierarchal Namespace Keep only working set in memory
Storage & IO
Container Management
Old HDFS Block storage on Data Nodes
Container Storage on Data Nodes
Physical Storage - Shared

So how will Hadoop customer use Ozone and what will be the migration over time

- Hadoop customers will use both size by side with HDFS as the default file system (they already do this in the cloud):
  - HDFS for Apps that absolutely need the HDFS API
  - Ozone if they have scalability issues for applications like Hive and Spark that run well on a key-value store via a HCFS API.

    - This will allow the Ozone's block-container- layer to stabilize and become rock solid.



- Over time we can add a truly scalable hierarchical namespace that uses Ozone's block-container layer.

So why put the Ozone in HDFS and not keep it a separate project:

1) Hadoop customers want to share the physical storage and disk IO scheduling - they don't want a separate cluster or nodes for Ozone storage. Indeed Ozone's block storage layer uses the same block-pool namespace as HDFS so that both can live side-by-side.
2) Ozone's block-container storage layer is designed to have HDFS file IO characteristics (such as fast IO scans) rather than the small object IO characteristics of object stores. This is because our goal was solve the HDFS scalability and Hadoop Big Data applications not Key-Value storage for documents etc.
3) We want a ***future*** truly scalable hierarchical namespace to use Ozone's scalable block-container layer storage as proposed in [HDFS-10419]. This was the original goal and doing a key-value namespace is an expedient way to give users who are suffering from HDFS namespace scaling some relief and to start the process of stabilizing the block-container layer.