By Suresh Srinivas
on
August 23rd, 2011

HDFS Federation

HDFS Federation improves the existing HDFS architecture through a clear separation of namespace and storage, enabling generic block storage layer. It enables support for multiple namespaces in the cluster to improve scalability and isolation. Federation also opens up the architecture, expanding the applicability of HDFS cluster to new implementations and use cases.

Overview of Current HDFS

HDFS has two main layers:

- **Namespace** manages directories, files and blocks. It supports file system operations such as creation, modification, deletion and listing of files and directories.
- **Block Storage**has two parts:
    - **Block Management** maintains the membership of datanodes in the cluster. It supports block-related operations such as creation, deletion, modification and getting location of the blocks. It also takes care of replica placement and replication.
    - **Physical Storage** stores the blocks and provides read/write access to it.

The current HDFS architecture allows only a single namespace for the entire cluster. This namespace is managed by a single namenode. This architectural decision made HDFS simpler to implement. However, the architectural layering described above in practice got blurred in the implementation, resulting in some limitations described below. Only the largest deployments like Yahoo! and Facebook face ***some*** of these limitations. These are addressed by HDFS Federation.

Tight coupling of Block Storage and Namespace

Currently the collocation of namespace and block management in the namenode has resulted in tight coupling of these two layers. This makes alternate implementations of namenodes challenging and limits other services from using the block storage ***directly***.

Namespace scalability

While HDFS cluster storage scales horizontally with the addition of datanodes, the namespace does not. Currently the namespace can only be vertically scaled on a single namenode. The namenode stores the entire file system metadata in memory. This limits the number of blocks, files, and directories supported on the file system to what can be accommodated in the memory of a single namenode. A typical large deployment at Yahoo! includes an HDFS cluster with 2700-4200 datanodes with 180 million files and blocks, and address ~25 PB of storage. At Facebook, HDFS has around 2600 nodes, 300 million files and blocks, addressing up to 60PB of storage. While these are very large systems and good enough for majority of Hadoop users, a few deployments that might want to grow even larger could find the namespace scalability limiting.

Performance

File system operations are limited to the throughput of a single namenode, which currently supports 60K tasks. The Next Generation of Apache MapReduce will support more than 100K concurrent tasks, which will require multiple namenodes.

Isolation

At Yahoo! and many other deployments, the cluster is used in a multi-tenant environment where many organizations share the cluster. A single namenode offers no isolation in this setup. A separate namespace for a tenant is not possible. An experimental application that overloads the namenode can slow down the other production applications. A single namenode also does not allow segregating different categories of applications (such as HBase) to separate namenodes.

HDFS Federation

In order to scale the name service horizontally, federation uses multiple independent namenodes/namespaces. The namenodes are federated, that is, the namenodes are independent and don't require coordination with each other. The datanodes are used as common storage for blocks by all the namenodes. Each datanode registers with all the namenodes in the cluster. Datanodes send periodic heartbeats and block reports and handles commands from the namenodes.

A **Block Pool** is a set of blocks that belong to a single namespace. Datanodes store blocks for all the block pools in the cluster.

It is managed independently of other block pools. This allows a namespace to generate Block IDs for new blocks without the need for coordination with the other namespaces. The failure of a namenode does not prevent the datanode from serving other namenodes in the cluster.

A Namespace and its block pool together are called **Namespace Volume**. It is a self-contained unit of management. When a namenode/namespace is deleted, the corresponding block pool at the datanodes is deleted. Each namespace volume is upgraded as a unit, during cluster upgrade.

Key Benefits

Scalability and isolation

Support for multiple namenodes horizontally scales the file system namespace. It separates namespace volumes for users and categories of applications and improves isolation.

Generic storage service

Block pool abstraction opens up the architecture for future innovation. New file systems can be built on top of block storage. New applications can be directly built on the block storage layer without the need to use a file system interface. New block pool categories are also possible, different from the default block pool. Examples include a block pool for MapReduce tmp storage with different garbage collection scheme or a block pool that caches data to make distributed cache more efficient.

Design simplicity

We considered distributed namenodes and chose to go with federation because it is significantly simpler to design and implement. Namenodes and namespaces are independent of each other and require very little change to the existing namenodes. The robustness of the namenode is not affected. Federation also preserves backward compatibility of configuration. The existing single namenode deployments work without any configuration changes.

It took only 4 months to implement the features and stabilize the software. The namenode has very few changes. Most of the changes are in the datanode, to introduce block pool as a new hierarchy in storage, replica map and other internal data structures. Other changes include fixing the tests to work with new hierarchy of data structures and tools to simplify management of federated clusters.

Acknowledgements & Conclusion

HDFS Federation was developed in HDFS-1052 branch. Please see HDFS-1052, the umbrella JIRA for the detailed design and the sub tasks that introduced the federation feature. The feature has been merged into trunk and will be available in 0.23 release.

The design of this feature was driven by Sanjay Radia and Suresh Srinivas. The core of the implementation was done by Boris Shkolnik, Jitendra Pandey, Matt Foley, Tsz Wo (Nicholas) Sze, Suresh Srinivas and Tanping Wang. The client-side mount table feature was implemented by Sanjay Radia. The feature development was completed while the team was at Yahoo!. We are continuing to test and enhance this feature further at Hortonworks.

Stay tuned for an upcoming blog by Sanjay Radia on the client-side mount table, which will discuss how to manage multiple namespaces in the federated HDFS cluster.