Scaling HDFS Namenode using Multiple Namenodes and Block Pools

HDFS Team
Yahoo!

March 2010

1 Introduction.............................................................................................................................................. 2
1.1 Terminology........................................................................................................................................ 2
1.2 Namenode Scalability ......................................................................................................................... 2
1.3 Proposal: Horizontal Scaling of Namespaces ...................................................................................... 2
   1.3.1 What is a HDFS Cluster? .............................................................................................................. 2
   1.3.2 Benefits and Drawbacks ............................................................................................................. 3
1.4 Background ........................................................................................................................................ 3
2 Multiple Namespaces using Block Pools ............................................................................................... 4
2.1 Block Identification: .......................................................................................................................... 5
2.2 Benefits ............................................................................................................................................. 5
2.3 Quality of Service Features .............................................................................................................. 5
2.4 Two Implementations Approaches: .................................................................................................... 5
3 Use Cases............................................................................................................................................... 7
   3.1.1 Create a New Cluster ................................................................................................................. 7
   3.1.2 Add Datanodes to a Cluster ....................................................................................................... 7
   3.1.3 Add a Namenode to a Cluster .................................................................................................... 7
   3.1.4 Delete a Namespace/Block pool from a Cluster .......................................................................... 7
   3.1.5 Datanode is Accidentally Moved to Different Cluster ............................................................... 8
   3.1.6 Move a Namespace from one Namenode to a Different Namenode ......................................... 8
   3.1.7 Move a Sub-Namespace from one Namenode to Another ......................................................... 8
   3.1.8 Move a Namespace to a Different Cluster ................................................................................. 8
   3.1.9 Upgrade Namenode or Datanode Software .............................................................................. 8
   3.1.10 Decommission a Datanode ...................................................................................................... 8
   3.1.11 Balance Datanode Storage Utilization .................................................................................... 9
3 Implementation Details ............................................................................................................................ 9
1 Introduction
HDFS cluster has a single namenode that manages the file system namespace. The current limitation that a cluster can contain only a single namenode results in the following issues:

1. **Namespace Scalability**: Namenode maintains the entire file system metadata in memory. The size of the metadata is limited by the physical memory available on the node. This results in the following issues:
   a. Scaling storage – while storage can be scaled by adding more datanodes/disks to the datanodes, the total number of blocks reference-able is limited by the memory on the namenode because more blocks results in more metadata. (Note once can add storage without increasing metadata by making block sizes larger.)
   b. Scaling namespace – the number of files and directories that can be created is limited by the memory on namenode.

   To address these issues one encourages larger block sizes, creating a smaller number of larger files and using tools like the hadoop archive (har).

2. **Performance Scalability**: File system operations throughput is limited to the throughput of a single namenode.

3. **Isolation**: No isolation for a multi-tenant environment. An experimental client application that puts high load on the central namenode can impact a production application.

4. **Availability**: While the design does not prevent building a failover mechanism, when a failure occurs the entire namespace and hence the entire cluster is down.

1.1 Terminology

<table>
<thead>
<tr>
<th>Namespace</th>
<th>A hierarchical naming structure of directories and files</th>
</tr>
</thead>
<tbody>
<tr>
<td>Namenode</td>
<td>A server that stores and provides access to a namespace</td>
</tr>
<tr>
<td>Namespace volume</td>
<td>The namespace and the set of blocks it references. This is an independent unit of management.</td>
</tr>
<tr>
<td>Vertical Scaling</td>
<td>Scale by using a larger unit (e.g. a larger server, more memory, more cores, etc)</td>
</tr>
<tr>
<td>Horizontal Scaling</td>
<td>Scale by adding additional units. (e.g. add more servers – the storage and IO bandwidth is scaled by adding more datanodes).</td>
</tr>
</tbody>
</table>

1.2 Namenode Scalability
To support more files and storage capacity, the current approach is to vertically scale the namenode by increasing the java heap for the namenode process. A cluster of 10.4PB and 3700 datanodes is currently configured with java heap at 40GB. As the demand for cluster capacity increases, the cluster will run into namenode physical memory limitation (64GB on some machines). Optimizing the namenode to use memory more efficiently is expensive both in development effort and complexity added to the code. Further, using larger java heap results in GC issues, and longer startup time. It also makes debugging JVM related problems harder, since such a large JVM is not supported well by tools such as jhat.

1.3 Proposal: Horizontal Scaling of Namespaces
The proposal is to allow multiple namespaces (and namenodes) in a HDFS cluster, which share the storage capacity on all the datanodes.

1.3.1 What is a HDFS Cluster?
Currently the HDFS cluster contains:
• A single HDFS namespace implemented inside a single namenode
• A single pool of blocks implemented using multiple datanodes

Instead, with the proposed approach, an HDFS cluster contains:
• Multiple independent HDFS namespaces implemented using multiple namenodes
• Multiple independent pools of blocks, implemented using multiple datanodes; the data nodes are not partitioned – each datanode can provide storage to each block pool and each namenode.

1.3.2 Benefits and Drawbacks

This has the following benefits:
1. Provides a horizontal scaling solution for namespace and namespace operations and hence for the entire HDFS file system: one can scale by simply adding more namespaces/namenodes.
2. Multiple namenodes enable partitioning customers to different namespaces/namenodes for better isolation, availability and manageability.
3. Multiple namenodes share the common datanode pool for storage resulting in better storage utilization compared to partitioning datanodes among namenodes.

However, this approach has the following drawbacks:
1. One needs to manage multiple namespaces and namenodes:
   a. To provide user transparency, one can use symbolic links and/or client-side mount tables (i.e. application-centric or job centric) namespaces (See HDFS-1052).
   b. The operations staff has to partition the current single namespace into multiple namespaces and run multiple namenode servers.

(TBD add more text on how some of the drawbacks can be better addressed.)

1.4 Background

Currently HDFS architecture provides 2 main layers:
• Namespace management - manages namespace consisting of directories, files and blocks. It supports file system operations such as creation/modification/deletion and listing of files and directories.
• Block storage, which has two parts:
  o Block management – manages the membership of datanodes in the cluster for block storage. Supports block related operations such as creation/deletion/modification/getting location of blocks, replica placement and block replication to satisfy replication factor and placement.
  o Physical storage and access to block data across the network.

The current HDFS implementation is structured as follows:
• The namenode implements:
  o Namespace management
  o Block management
• The datanodes provide the physical storage and access to block data. Datanodes register with the block management layer in the namenode to form the storage layer for the HDFS cluster.
  o Note although we think of the datanode as registering and communicating with the namenode, it is really communicating with the block management layer inside the namenode and not with the namespace management layer.
• Hence the block storage subsystem is implemented partly in the datanodes and partly in the namenode.

Within the namenode implementation, the internal java APIs do not provide a very clean separation.
Block Identification:
A file is one or more blocks with replicas stored on datanodes. Each block is identified with a unique cluster-wide block ID (64-bit number). Current system is a system with a single namespace, using a single pool of blocks for storage.

2 Multiple Namespaces using Block Pools
In the proposed architecture, to improve scalability, multiple namenodes/namespaces are used in the cluster. Each namespace uses all the datanodes as common storage. A namespace uses one or more sets of blocks called “block pools”. The following diagram depicts this new model:
Salient features:

- Each **Block Pool** is a disjoint set (i.e. pool) of blocks: a block is a member of **only one** block pool. Block management layer manages the block pools, along with the other block related functionality as described in the previous section.
- Datanodes provide a shared storage layer and store blocks belonging to **all** the block pools in the cluster.
- A namespace uses one or more block pools to store the files in blocks. A block pool belongs to a single namespace and does not cross namespace boundary.
- Namespace volume (the namespace and the set of blocks it references). is a self-contained unit of management.
  - Can be managed independently: unreferenced, missing, under-replicated, and over-replicated blocks can be determined by simply comparing the blocks referenced by a namespace and the block of the volume’s block pools stored on the datanodes.
  - Volumes don’t need to coordinate with each other
- Each datanode communicates with the block management layer:
  - Registers and sends periodic heartbeats
  - Sends block reports for each block pool.
  - Accepts commands for block management (copy blocks, delete blocks, etc.)

### 2.1 Block Identification:

A Block is identified by the tuple `<Block Pool ID> <Block ID>`. This is called *Extended Block ID*. The Block Pool ID is unique only within a cluster. Initial proposal is make the block ID is unique within a block pool.; however we are evaluating the benefits of making block ids unique across block pools; we need to decide this before the initial version is released.

Benefits of making block IDs unique across block pools: (TBD).

### 2.2 Benefits

Besides those identified in section 1.3.2), the proposal has the following additional benefits:

- This enables a non-HDFS namespace to make use of block storage layer.
- We are investigating special block pools for MapReduce temp storage (details soon).

### 2.3 Quality of Service Features

Current plan is to not provide QoS features such as quotas, performance at the block storage layer but to continue providing some these (such as storage quota) at the namespace layer. This can be revisited later, with the caution that we need to avoid the complexity introduced in SAN storage systems.

### 2.4 Two Implementations Approaches:

1. Keep the block management function in the namenode (as today), with each namenode managing **only its** block pools. (See Figure 3)
   - Each datanode registers with and sends heartbeats to the block management layer in **each** namenode.
   - Each datanode sends block reports to the block management layer in **each** namenode. The block report contains **only the blocks for the specific block pool** owned by that namenode.
2. Move block management out of the namenodes management as a distinct service. This service could be implemented on horizontally scaled set of block management servers.

---

1 Initially a namenode/namespace will use only one block pool. There are some benefits to generalizing this as described later.
• The pattern of registration, heartbeats and block reports will depend on how the block manager service is partitioned across multiple servers.

• Advantages of retaining block management in namenode.
  a. Easily fits with the existing design and implementation
• Advantages of separating block management from the namenode.
  b. Forces a cleaner separation of layers.
  c. Less message exchange with the block management layer.
    • Registration and heartbeat is sent from the datanodes to the block management layer and not to each namenode. (note since the block management layer is likely to be implemented as multi-node service this benefit is somewhat less).
    • A block report can contain data for multiple block pools. (very minor benefit).
  d. Namenode implementation or a non-HDFS namespace in another programming language is easier since the block management is already provided as service.
  e. The separated block management layer can scale differently compared to namespace management.
  f. Currently namenode plays a subtle role as cluster manager. This can be subsumed by the block management layer since there is no clear master namenode in a multi-namenode environment.

The initial implementation will use approach 1, as shown below.

Salient features:
  1. A namenode manages a single namespace and the set of block it refers (i.e. a single namespace volume)
2. A namespace has its own unshared set of block pools (the first implementation is likely to limit a namenode to have a single block pool.
3. Each namenode is completely independent of other namenodes: there is no coordination amongst the namenodes. Each namenode provides the place to manage the namespace volume.
4. Block pool management remains in namenode as today
5. Continues the existing separation of namespace layer from block storage layer. Even though the namenode has the namespace and the block management, the goal is to maintain and further separate the two layers.

3 Use Cases

3.1.1 Create a New Cluster
1. Setup HDFS cluster with namenodes and the corresponding block pool IDs and list of datanodes and other configuration.
2. Namenodes and datanodes are started (order does not matter).
3. Datanodes registers with the block management layer in each of the namenodes, and send heartbeats and block reports periodically.

3.1.2 Add Datanodes to a Cluster
1. Cluster configuration is updated to include new datanodes.
2. New datanodes are started.
3. The new datanodes register with the block management layer in the namenodes and become part of the cluster.
   Variations:
   • Some namenodes are down when datanodes were added
     o Datanodes periodically register with failed namenodes.

3.1.3 Add a Namenode to a Cluster
1. Allocate a new block pool ID (unique within the cluster) and add this new block pool ID and the new namenode to the cluster configuration.
2. Next steps (order unimportant as today)
   a. New namenode is started.
   b. All the data nodes in the cluster are refreshed with the new configuration.
3. Datanodes discover the new block pool ID. They create the required directories for storing blocks from the new block pool. The datanodes register with the block management layer in the newly added namenode and send heartbeats and block reports periodically.
   Variations:
   • Datanode was down when a namenode was added to the cluster
     • When datanode is restarted, it gets the latest configuration, which includes new block pool and its manager the new namenode. Continue as in use case 3.1.2

3.1.4 Delete a Namespace/Block pool from a Cluster
1. Delete the namespace from the namenode and shutdown namenode ensuring that datanodes have sync’ed with namenode.
2. Next step:
   a. If there is a central Block manager (approach 2) then mark the block pool as deleted; it communicates with datanodes to delete the block pool.
If there is no central block manager (approach 1) then mark the block pool as deleted in the config file and ssh into each datanode and ask it to remove the block pool.

### 3.1.5 Datanode is Accidentally Moved to Different Cluster

1. Datanode configuration error points it to one or more namenodes on a different cluster.
2. Datanode registration is rejected in the foreign cluster by each namenode.
   - A birthmark can be used to deal with such issues. In the current implementation the namespaceld is a birthmark to prevents a data node from joining a wrong cluster.

### 3.1.6 Move a Namespace from one Namenode to a Different Namenode

1. Namenode is stopped.
2. Move the required configuration and data directories with metadata from one namenode to another namenode.
3. The DNS name and/or address of the namenode are updated in the config file and or the DNS server as appropriate.
4. The new namenode is started.

### 3.1.7 Move a Sub-Namespace from one Namenode to Another

1. Copy the sub-namespace to the namespace of the second namenode, using tools such as distcp.
2. Delete the sub namespace from the first namespace.

In the future we may support creating a copy-on-write snapshot with a different block pool ID and then moving the snapshot so as to avoid copying the data blocks.

### 3.1.8 Move a Namespace to a Different Cluster

1. Copy the namespace to the other cluster, using tools such as distcp.
2. Delete the namespace from the first cluster.

### 3.1.9 Upgrade Namenode or Datanode Software

**Background: Current upgrade procedure:**

One of the nice things about current HDFS is the system allows one to take a snapshot as part of an upgrade. A rollback can be done if one is not happy with the upgrade.

If a layout change has occurred in either namenode or datanode, a snapshot is mandatory: both namenode and all datanodes take a snapshot. If a software change has occurred, a snapshot is optional but a recommended operating procedure.

Upgrade use cases TBD

### 3.1.10 Decommission a Datanode

1. With implementation approach 1 (block manager in each namenode)
   - Two approaches:
     a. Ask each namenode to decommission the datanode for its block pools
     b. Ask the datanode to be decommission to perform the work by communicating with each of the namenode
2. With implementation approach 2 (separate block management layer)
   - Ask block management layer to decommission the datanode
3.1.11 Balance Datanode Storage Utilization

1. Start balancer
2. Balancer obtains utilization and block info for all blocks:
   • With implementation approach 1 (block manager in each namenode): Each namenode’s block management layer reports each datanode’s total utilization and the block maps for its block pools to the balancer. Balancer collects this info from each running namenode, aggregates the info.
   • With implementation approach 1 (separate block manager layer): Balancer contacts the block management service to each datanode’s total utilization and get block info.
3. Balancer balances the utilization.

3. Implementation Details
Details to follow in a separate document.