HDFS METADATA DIRECTORIES EXPLAINED

HDFS metadata represents the structure of HDFS directories and files in a tree. It also includes the various attributes of directories and files, such as ownership, permissions, quotas, and replication factor. In this blog post, I'll describe how HDFS persists its metadata in Hadoop 2 by exploring the underlying local storage directories and files. All examples shown are from testing a build of the soon-to-be-released Apache Hadoop 2.6.0.

WARNING*: Do not attempt to modify metadata directories or files. Unexpected modifications can cause HDFS downtime, or even permanent data loss. This information is provided for educational purposes only.*

Persistence of HDFS metadata broadly breaks down into 2 categories of files:

- **fsimage** – An fsimage file contains the complete state of the file system at a point in time. Every file system modification is assigned a unique, monotonically increasing transaction ID. An fsimage file represents the file system state after all modifications up to a specific transaction ID.

- **edits** – An edits file is a log that lists each file system change (file creation, deletion or modification) that was made after the most recent fsimage.

Checkpointing is the process of merging the content of the most recent fsimage with all edits applied after that fsimage is merged in order to create a new fsimage. Checkpointing is triggered automatically by configuration policies or manually by HDFS administration commands.

NAMENODE

Here is an example of an HDFS metadata directory taken from a NameNode. This shows the output of running the tree command on the metadata directory, which is configured by setting d*fs.namenode.name.dir* in *hdfs-site.xml*.

```
data/dfs/name
├── current
│ ├── VERSION
│ ├── edits_0000000000000000001-0000000000000000007
│ ├── edits_0000000000000000008-0000000000000000015
```

```
| ├── edits_0000000000000000016-0000000000000000022
| ├── edits_0000000000000000023-0000000000000000029
| ├── edits_0000000000000000030-0000000000000000030
| ├── edits_0000000000000000031-0000000000000000031
| ├── edits_inprogress_0000000000000000032
| ├── fsimage_0000000000000000030
| ├── fsimage_0000000000000000030.md5
| ├── fsimage_0000000000000000031
| ├── fsimage_0000000000000000031.md5
| └── seen_txid
└── in_use.lock
```

In this example, the same directory has been used for both fsimage and edits. Alternatively, configuration options are available that allow separating fsimage and edits into different directories. Each file within this directory serves a specific purpose in the overall scheme of metadata persistence:

- **VERSION** – Text file that contains:
  - **layoutVersion** – The version of the HDFS metadata format. When we add new features that require changing the metadata format, we change this number. An HDFS upgrade is required when the current HDFS software uses a layout version newer than what is currently tracked here.
  - **namespaceID/clusterID/blockpoolID** – These are unique identifiers of an HDFS cluster. The identifiers are used to prevent DataNodes from registering accidentally with an incorrect NameNode that is part of a different cluster. These identifiers also are particularly important in a federated deployment. Within a federated deployment, there are multiple NameNodes working independently. Each NameNode serves a unique portion of the namespace (namespaceID) and manages a unique set of blocks (blockpoolID). The clusterID ties the whole cluster together as a single logical unit. It's the same across all nodes in the cluster.
  - **storageType** – This is either NAME_NODE or JOURNAL_NODE. Metadata on a JournalNode in an HA deployment is discussed later.
  - **cTime** – Creation time of file system state. This field is updated during HDFS upgrades.
- **edits_start transaction ID-end transaction ID** – These are finalized (unmodifiable) edit log segments. Each of these files contains all of the edit log transactions in the range defined by the file name's through. In an

HA deployment, the standby can only read up through the finalized log segments. It will not be up-to-date with the current edit log in progress (described next). However, when an HA failover happens, the failover finalizes the current log segment so that it's completely caught up before switching to active.

- **edits_inprogress__start transaction ID** – This is the current edit log in progress. All transactions starting from are in this file, and all new incoming transactions will get appended to this file. HDFS pre-allocates space in this file in 1 MB chunks for efficiency, and then fills it with incoming transactions. You'll probably see this file's size as a multiple of 1 MB. When HDFS finalizes the log segment, it truncates the unused portion of the space that doesn't contain any transactions, so the finalized file's space will shrink down.

- **fsimage_end transaction ID** – This contains the complete metadata image up through. Each fsimage file also has a corresponding .md5 file containing a MD5 checksum, which HDFS uses to guard against disk corruption.

- **seen_txid** – This contains the last transaction ID of the last checkpoint (merge of edits into a fsimage) or edit log roll (finalization of current edits_inprogress and creation of a new one). Note that this is not the last transaction ID accepted by the NameNode. The file is not updated on every transaction, only on a checkpoint or an edit log roll. The purpose of this file is to try to identify if edits are missing during startup. It's possible to configure the NameNode to use separate directories for fsimage and edits files. If the edits directory accidentally gets deleted, then all transactions since the last checkpoint would go away, and the NameNode would start up using just fsimage at an old state. To guard against this, NameNode startup also checks seen_txid to verify that it can load transactions at least up through that number. It aborts startup if it can't.

- **in_use.lock** – This is a lock file held by the NameNode process, used to prevent multiple NameNode processes from starting up and concurrently modifying the directory.

JOURNALNODE

In an HA deployment, edits are logged to a separate set of daemons called JournalNodes. A JournalNode's metadata directory is configured by setting *dfs.journalnode.edits.dir*. The JournalNode will contain a VERSION file, multiple edits__ files and an edits_inprogress_, just like the NameNode. The JournalNode will not have fsimage files or seen_txid. In addition, it contains several other files relevant to the HA implementation. These files help prevent a split-brain scenario, in which multiple NameNodes could think they are active and all try to write edits.

- **committed-txid** – Tracks last transaction ID committed by a NameNode.

- **last-promised-epoch** – This file contains the “epoch,” which is a monotonically increasing number. When a new writer (a new NameNode) starts as active, it increments the epoch and presents it in calls to the JournalNode. This scheme is the NameNode’s way of claiming that it is active and requests from another NameNode, presenting a lower epoch, must be ignored.

- **last-writer-epoch** – Similar to the above, but this contains the epoch number associated with the writer who last actually wrote a transaction. (This was a bug fix for an edge case not handled by last-promised-epoch alone.)

- **paxos** – Directory containing temporary files used in implementation of the Paxos distributed consensus protocol. This directory often will appear as empty.

DATANODE

Although DataNodes do not contain metadata about the directories and files stored in an HDFS cluster, they do contain a small amount of metadata about the DataNode itself and its relationship to a cluster. This shows the output of running the tree command on the DataNode’s directory, configured by setting *dfs.datanode.data.dir* in *hdfs-site.xml*.

```
data/dfs/data/
├── current
| ├── BP-1079595417-192.168.2.45-1412613236271
| | ├── current
| | | ├── VERSION
| | | ├── finalized
| | | | └── subdir0
| | | | └── subdir1
| | | | ├── blk_1073741825
| | | | └── blk_1073741825_1001.meta
| | | |── lazyPersist
| | | └── rbw
| | ├── dncp_block_verification.log.curr
| | ├── dncp_block_verification.log.prev
| | └── tmp
| └── VERSION
└── in_use.lock
```

The purpose of these files is:

- **BP-random integer-NameNode-IP address-creation time** – The naming convention on this directory is significant and constitutes a form of cluster metadata. The name is a block pool ID. “BP” stands for “block pool,” the abstraction that collects a set of blocks belonging to a single namespace. In the case of a federated deployment, there will be multiple “BP” sub-directories, one for each block pool. The remaining components form a unique ID: a random integer, followed by the IP address of the NameNode that created the block pool, followed by creation time.

- **VERSION** – Much like the NameNode and JournalNode, this is a text file containing multiple properties, such as layoutVersion, clusterId and cTime, all discussed earlier. There is a VERSION file tracked for the entire DataNode as well as a separate VERSION file in each block pool sub-directory. In addition to the properties already discussed earlier, the DataNode’s VERSION files also contain:

  - **storageType** – In this case, the storageType field is set to DATA_NODE.

  - **blockpoolID** – This repeats the block pool ID information encoded into the sub-directory name.

- **finalized/rbw** – Both finalized and rbw contain a directory structure for block storage. This holds numerous block files, which contain HDFS file data and the corresponding .meta files, which contain checksum information. “Rbw” stands for “replica being written”. This area contains blocks that are still being written to by an HDFS client. The finalized sub-directory contains blocks that are not being written to by a client and have been completed.

- **lazyPersist** – HDFS is incorporating a new feature to support writing transient data to memory, followed by lazy persistence to disk in the background. If this feature is in use, then a lazyPersist sub-directory is present and used for lazy persistence of in-memory blocks to disk. We’ll cover this exciting new feature in greater detail in a future blog post.

- **dncp_block_verification.log** – This file tracks the last time each block was verified by checking its contents against its checksum. The last verification time is significant for deciding how to prioritize subsequent verification work. The DataNode orders its background block verification work in ascending order of last verification time. This file is rolled periodically, so it’s typical to see a .curr file (current) and a .prev file (previous).

- **in_use.lock** – This is a lock file held by the DataNode process, used to prevent multiple DataNode processes from starting up and concurrently modifying the directory.