HDFS User Guide

https://hadoop.apache.org/docs/r1.0.4/hdfs_user_guide.html

Purpose

This document is a starting point for users working with Hadoop Distributed File System (HDFS) either as a part of a Hadoop cluster or as a stand-alone general purpose distributed file system. While HDFS is designed to "just work" in many environments, a working knowledge of HDFS helps greatly with configuration improvements and diagnostics on a specific cluster.

Overview

HDFS is the primary distributed storage used by Hadoop applications. A HDFS cluster primarily consists of a NameNode that manages the file system metadata and DataNodes that store the actual data. The HDFS Architecture Guide describes HDFS in detail. This user guide primarily deals with the interaction of users and administrators with HDFS clusters. The HDFS architecture diagram depicts basic interactions among NameNode, the DataNodes, and the clients. Clients contact NameNode for file metadata or file modifications and perform actual file I/O directly with the DataNodes.

The following are some of the salient features that could be of interest to many users.

- Hadoop, including HDFS, is well suited for distributed storage and distributed processing using commodity hardware. It is fault tolerant, scalable, and extremely simple to expand. MapReduce, well known for its simplicity and applicability for large set of distributed applications, is an integral part of Hadoop.
- HDFS is highly configurable with a default configuration well suited for many installations. Most of the time, configuration needs to be tuned only for very large clusters.
- Hadoop is written in Java and is supported on all major platforms.
- Hadoop supports shell-like commands to interact with HDFS directly.
- The NameNode and Datanodes have built in web servers that makes it easy to check current status of the cluster.
- New features and improvements are regularly implemented in HDFS. The following is a subset of useful features in HDFS:

- File permissions and authentication.
- *Rack awareness*: to take a node's physical location into account while scheduling tasks and allocating storage.
- Safemode: an administrative mode for maintenance.
- fsck: a utility to diagnose health of the file system, to find missing files or blocks.
- fetchdt: a utility to fetch DelegationToken and store it in a file on the local system.

- Rebalancer: tool to balance the cluster when the data is unevenly distributed among DataNodes.
- Upgrade and rollback: after a software upgrade, it is possible to rollback to HDFS' state before the upgrade in case of unexpected problems.
- Secondary NameNode (deprecated): performs periodic checkpoints of the namespace and helps keep the size of file containing log of HDFS modifications within certain limits at the NameNode. Replaced by Checkpoint node.
- Checkpoint node: performs periodic checkpoints of the namespace and helps minimize the size of the log stored at the NameNode containing changes to the HDFS. Replaces the role previously filled by the Secondary NameNode. NameNode allows multiple Checkpoint nodes simultaneously, as long as there are no Backup nodes registered with the system.
- Backup node: An extension to the Checkpoint node. In addition to checkpointing it also receives a stream of edits from the NameNode and maintains its own in-memory copy of the namespace, which is always in sync with the active NameNode namespace state. Only one Backup node may be registered with the NameNode at once.

Prerequisites

The following documents describe how to install and set up a Hadoop cluster:

- Single Node Setup for first-time users.
- Cluster Setup for large, distributed clusters.

The rest of this document assumes the user is able to set up and run a HDFS with at least one DataNode. For the purpose of this document, both the NameNode and DataNode could be running on the same physical machine.

Web Interface

NameNode and DataNode each run an internal web server in order to display basic information about the current status of the cluster. With the default configuration, the NameNode front page is at http://namenode-name:50070/. It lists the DataNodes in the cluster and basic statistics of the cluster. The web interface can also be used to browse the file system (using "Browse the file system" link on the NameNode front page).

Shell Commands

Hadoop includes various shell-like commands that directly interact with HDFS and other file systems that Hadoop supports. The command bin/hdfs dfs -help lists the commands supported by Hadoop shell. Furthermore, the command bin/hdfs dfs -help command-name displays more detailed help for a command. These commands support most of the normal files system operations like copying files, changing file permissions, etc. It also supports a few HDFS specific operations like changing replication of files. For more information see File System Shell Guide.

DFSAdmin Command

The bin/hadoop dfsadmin command supports a few HDFS administration related operations. The bin/hadoop dfsadmin -help command lists all the commands currently supported. For e.g.:

- -report : reports basic statistics of HDFS. Some of this information is also available on the NameNode front page.
- -safemode : though usually not required, an administrator can manually enter or leave Safemode.
- -finalizeUpgrade : removes previous backup of the cluster made during last upgrade.
- -refreshNodes : Updates the set of hosts allowed to connect to namenode. Re-reads the config file to update values defined by dfs.hosts and dfs.host.exclude and reads the entires (hostnames) in those files. Each entry not defined in dfs.hosts but in dfs.hosts.exclude is decommissioned. Each entry defined in dfs.hosts and also in dfs.host.exclude is stopped from decommissioning if it has aleady been marked for decommission. Entires not present in both the lists are decommissioned.
- -printTopology : Print the topology of the cluster. Display a tree of racks and datanodes attached to the tracks as viewed by the NameNode.

For command usage, see dfsadmin.

Secondary NameNode

Note

The Secondary NameNode has been deprecated. Instead, consider using the Checkpoint Node or Backup Node.

The NameNode stores modifications to the file system as a log appended to a native file system file, edits. When a NameNode starts up, it reads HDFS state from an image file, fsimage, and then applies edits from the edits log file. It then writes new HDFS state to the fsimage and starts normal operation with an empty edits file. Since NameNode merges fsimage and edits files only during start up, the edits log file could get very large over time on a busy cluster. Another side effect of a larger edits file is that next restart of NameNode takes longer.

The secondary NameNode merges the fsimage and the edits log files periodically and keeps edits log size within a limit. It is usually run on a different machine than the primary NameNode since its memory requirements are on the same order as the primary NameNode. The secondary NameNode is started by bin/start-dfs.sh on the nodes specified in conf/masters file.

The start of the checkpoint process on the secondary NameNode is controlled by two configuration parameters.

- fs.checkpoint.period, set to 1 hour by default, specifies the maximum delay between two consecutive checkpoints, and
- fs.checkpoint.size, set to 64MB by default, defines the size of the edits log file that forces an urgent checkpoint even if the maximum checkpoint delay is not reached.

The secondary NameNode stores the latest checkpoint in a directory which is structured the same way as the primary NameNode's directory. So that the check pointed image is always ready to be read by the primary NameNode if necessary.

For command usage, see secondarynamenode.

Checkpoint Node

NameNode persists its namespace using two files: fsimage, which is the latest checkpoint of the namespace and edits, a journal (log) of changes to the namespace since the checkpoint. When a NameNode starts up, it merges the fsimage and edits journal to provide an up-to-date view of the file system metadata. The NameNode then overwrites fsimage with the new HDFS state and begins a new edits journal.

The Checkpoint node periodically creates checkpoints of the namespace. It downloads fsimage and edits from the active NameNode, merges them locally, and uploads the new image back to the active NameNode. The Checkpoint node usually runs on a different machine than the NameNode since its memory requirements are on the same order as the NameNode. The Checkpoint node is started by bin/hdfs namenode -checkpoint on the node specified in the configuration file.

The location of the Checkpoint (or Backup) node and its accompanying web interface are configured via the dfs.backup.address and dfs.backup.http.address configuration variables.

The start of the checkpoint process on the Checkpoint node is controlled by two configuration parameters.

- fs.checkpoint.period, set to 1 hour by default, specifies the maximum delay between two consecutive checkpoints
- fs.checkpoint.size, set to 64MB by default, defines the size of the edits log file that forces an urgent checkpoint even if the maximum checkpoint delay is not reached.

The Checkpoint node stores the latest checkpoint in a directory that is structured the same as the NameNode's directory. This allows the checkpointed image to be always available for reading by the NameNode if necessary. See Import Checkpoint.

Multiple checkpoint nodes may be specified in the cluster configuration file.

For command usage, see namenode.

Backup Node

The Backup node provides the same checkpointing functionality as the Checkpoint node, as well as maintaining an in-memory, up-to-date copy of the file system namespace that is always synchronized with the active NameNode state. Along with accepting a journal stream of file system edits from the NameNode and persisting this to disk, the Backup node also applies those edits into its own copy of the namespace in memory, thus creating a backup of the namespace.

The Backup node does not need to download fsimage and edits files from the active NameNode in order to create a checkpoint, as would be required with a Checkpoint node or Secondary NameNode, since it already has an up-to-date state of the namespace state in memory. The Backup node checkpoint process is more efficient as it only needs to save the namespace into the local fsimage file and reset edits.

As the Backup node maintains a copy of the namespace in memory, its RAM requirements are the same as the NameNode.

The NameNode supports one Backup node at a time. No Checkpoint nodes may be registered if a Backup node is in use. Using multiple Backup nodes concurrently will be supported in the future.

The Backup node is configured in the same manner as the Checkpoint node. It is started with bin/hdfs namenode -checkpoint.

The location of the Backup (or Checkpoint) node and its accompanying web interface are configured via the dfs.backup.address and dfs.backup.http.address configuration variables.

Use of a Backup node provides the option of running the NameNode with no persistent storage, delegating all responsibility for persisting the state of the namespace to the Backup node. To do this, start the NameNode with the -importCheckpoint option, along with specifying no persistent storage directories of type edits dfs.name.edits.dir for the NameNode configuration.

For a complete discussion of the motivation behind the creation of the Backup node and Checkpoint node, see HADOOP-4539. For command usage, see namenode.

Import Checkpoint

The latest checkpoint can be imported to the NameNode if all other copies of the image and the edits files are lost. In order to do that one should:

- Create an empty directory specified in the dfs.name.dir configuration variable;
- Specify the location of the checkpoint directory in the configuration variable fs.checkpoint.dir;
- and start the NameNode with -importCheckpoint option.

The NameNode will upload the checkpoint from the fs.checkpoint.dir directory and then save it to the NameNode directory(s) set in dfs.name.dir. The NameNode will fail if a legal image is contained in dfs.name.dir. The NameNode verifies that the image in fs.checkpoint.dir is consistent, but does not modify it in any way.

For command usage, see namenode.

Rebalancer

HDFS data might not always be be placed uniformly across the DataNode. One common reason is addition of new DataNodes to an existing cluster. While placing new blocks (data for a file is stored as a series of blocks), NameNode considers various parameters before choosing the DataNodes to receive these blocks. Some of the considerations are:

- Policy to keep one of the replicas of a block on the same node as the node that is writing the block.
- Need to spread different replicas of a block across the racks so that cluster can survive loss of whole rack.
- One of the replicas is usually placed on the same rack as the node writing to the file so that cross-rack network I/O is reduced.
- Spread HDFS data uniformly across the DataNodes in the cluster.

Due to multiple competing considerations, data might not be uniformly placed across the DataNodes. HDFS provides a tool for administrators that analyzes block placement and rebalanaces data across the DataNode. A brief administrator's guide for rebalancer as a PDF is attached to HADOOP-1652.

For command usage, see balancer.

Rack Awareness

Typically large Hadoop clusters are arranged in racks and network traffic between different nodes with in the same rack is much more desirable than network traffic across the racks. In addition NameNode tries to place replicas of block on multiple racks for improved fault tolerance. Hadoop lets the cluster administrators decide which rack a node belongs to through configuration variable dfs.network.script. When this script is configured, each node runs the script to determine its rack id. A default installation assumes all the nodes belong to the same rack. This feature and configuration is further described in PDF attached to HADOOP-692.

Safemode

During start up the NameNode loads the file system state from the fsimage and the edits log file. It then waits for DataNodes to report their blocks so that it does not prematurely start replicating the blocks though enough replicas already exist in the cluster. During this time NameNode stays in Safemode. Safemode for the NameNode is essentially a read-only mode for the HDFS cluster, where it does not allow any modifications to file system or blocks. Normally the NameNode leaves Safemode automatically after the DataNodes have reported that most file system blocks are available. If required, HDFS could be placed in Safemode explicitly using 'bin/hadoop dfsadmin -safemode' command. NameNode front page shows whether Safemode is on or off. A more detailed description and configuration is maintained as JavaDoc for NameNode.setSafeMode().

fsck

HDFS supports the fsck command to check for various inconsistencies. It it is designed for reporting problems with various files, for example, missing blocks for a file or under-replicated blocks. Unlike a traditional fsck utility for native file systems, this command does not correct the errors it detects. Normally NameNode automatically corrects most of the recoverable failures. By default fsck ignores open files but provides an option to select all files during reporting. The HDFS fsck command is not a Hadoop shell command. It can be run as 'bin/hadoop fsck'. For command usage, see fsck. fsck can be run on the whole file system or on a subset of files.

fetchdt

HDFS supports the fetchdt command to fetch Delegation Token and store it in a file on the local system. This token can be later used to access secure server (NameNode for example) from a non secure client. Utility uses either RPC or HTTPS (over Kerberos) to get the token, and thus requires kerberos tickets to be present before the run (run kinit to get the tickets). The HDFS fetchdt command is not a Hadoop shell command. It can be run as 'bin/hadoop fetchdt DTfile '. After you got the token you can run an HDFS command without having Kerberos tickets, by pointing HADOOP_TOKEN_FILE_LOCATION environmental variable to the delegation token file. For command usage, see fetchdt command.

Upgrade and Rollback

When Hadoop is upgraded on an existing cluster, as with any software upgrade, it is possible there are new bugs or incompatible changes that affect existing applications and were not discovered earlier. In any non-trivial HDFS installation, it is not an option to loose any data, let alone to restart HDFS from scratch. HDFS allows administrators to go back to earlier version of Hadoop and rollback the cluster to the state it was in before the upgrade. HDFS upgrade is described in more detail in Hadoop Upgrade Wiki page. HDFS can have one such backup at a time. Before upgrading, administrators need to remove existing backup using bin/hadoop dfsadmin -finalizeUpgrade command. The following briefly describes the typical upgrade procedure:

- Before upgrading Hadoop software, *finalize* if there an existing backup. dfsadmin -upgradeProgress status can tell if the cluster needs to be *finalized*.
- Stop the cluster and distribute new version of Hadoop.
- Run the new version with -upgrade option (bin/start-dfs.sh -upgrade).
- Most of the time, cluster works just fine. Once the new HDFS is considered working well (may be after a few days of operation), finalize the upgrade. Note that until the cluster is finalized, deleting the files that existed before the upgrade does not free up real disk space on the DataNodes.
- If there is a need to move back to the old version,

o stop the cluster and distribute earlier version of Hadoop.
o start the cluster with rollback option. (bin/start-dfs.h -rollback).

File Permissions and Security

The file permissions are designed to be similar to file permissions on other familiar platforms like Linux. Currently, security is limited to simple file permissions. The user that starts NameNode is treated as the superuser for HDFS. Future versions of HDFS will support network authentication protocols like Kerberos for user authentication and encryption of data transfers. The details are discussed in the Permissions Guide.

Scalability

Hadoop currently runs on clusters with thousands of nodes. The PoweredBy Wiki page lists some of the organizations that deploy Hadoop on large clusters. HDFS has one NameNode for each cluster. Currently the total memory available on NameNode is the primary scalability limitation. On very large clusters, increasing average size of files stored in HDFS helps with increasing cluster size without increasing memory requirements on NameNode. The default configuration may not suite very large clustes. The FAQ Wiki page lists suggested configuration improvements for large Hadoop clusters.

Accessibility HDFS can be accessed from applications in many different ways. Natively, HDFS provides a Java API for applications to use. A C language wrapper for this Java API is also available. In addition, an

HTTP browser can also be used to browse the files of an HDFS instance. Work is in progress to expose HDFS through the WebDAV protocol.

10.1. FS Shell HDFS allows user data to be organized in the form of files and directories. It provides a commandline interface called FS shell that lets a user interact with the data in HDFS. The syntax of this command set is similar to other shells (e.g. bash, csh) that users are already familiar with. Here are some sample action/command pairs: Action Command Create a directory named /foodir bin/hadoop dfs -mkdir /foodir HDFS Architecture Guide Page 12 Copyright © 2008 The Apache Software Foundation. All rights reserved. Remove a directory named /foodir bin/hadoop dfs -rmr /foodir View the contents of a file named /foodir/myfile.txt bin/hadoop dfs -cat /foodir/myfile.txt FS shell is targeted for applications that need a scripting language to interact with the stored data. 10.2. DFSAdmin The DFSAdmin command set is used for administering an HDFS cluster. These are commands that are used only by an HDFS administrator. Here are some sample action/command pairs: Action Command Put the cluster in Safemode bin/hadoop dfsadmin -safemode enter Generate a list of DataNodes bin/hadoop dfsadmin -report Recommission or decommission DataNode(s) bin/hadoop dfsadmin -refreshNodes 10.3. Browser Interface A typical HDFS install configures a web server to expose the HDFS namespace through a configurable TCP port. This allows a user to navigate the HDFS namespace and view the contents of its files using a web browser. 11. Space Reclamation 11.1. File Deletes and Undeletes When a file is deleted by a user or an application, it is not immediately removed from HDFS. Instead, HDFS first renames it to a file in the /trash directory. The file can be restored quickly as long as it remains in /trash. A file remains in /trash for a configurable amount of time. After the expiry of its life in /trash, the NameNode deletes the file from the HDFS namespace. The deletion of a file causes the blocks associated with the file to be freed. Note that there could be an appreciable time delay between the time a file is deleted by a user and the time of the corresponding increase in free space in HDFS. A user can Undelete a file after deleting it as long as it remains in the /trash directory. If a user wants to undelete a file that he/she has deleted, he/she can navigate the /trash directory and retrieve the file. The /trash directory contains only the latest copy of the file HDFS Architecture Guide Page 13 Copyright © 2008 The Apache Software Foundation. All rights reserved. that was deleted. The /trash directory is just like any other directory with one special feature: HDFS applies specified policies to automatically delete files from this directory. The current default policy is to delete files from /trash that are more than 6 hours old. In the future, this policy will be configurable through a well defined interface