Cloudera Runtime 7.1.0

HDFS Overview

Date published: 2020-02-20
Date modified:

https://docs.cloudera.com/

Legal Notice

Contents

Introduction 4
Overview of HDFS 4

NameNodes 4
Moving NameNode roles 5
Moving highly available NameNode, failover controller, and JournalNode roles using the Migrate Roles wizard 5
Moving a NameNode to a different host using Cloudera Manager 6
Sizing NameNode heap memory 7
Environment variables for sizing NameNode heap memory 7
Monitoring heap memory usage 7
Files and directories 7
Disk space versus namespace 8
Replication 8
Examples of estimating NameNode heap memory 8

DataNodes 9
How NameNode manages blocks on a failed DataNode 10
Remove a DataNode 10
Add storage directories using Cloudera Manager 11
Remove storage directories using Cloudera Manager 11

JournalNodes 11
Moving the JournalNode edits directory for a role group using Cloudera Manager 12
Moving the JournalNode edits directory for a role instance using Cloudera Manager 12

Introduction

Hadoop Distributed File System (HDFS) is a Java-based file system that provides scalable and reliable data storage. An HDFS cluster contains a NameNode to manage the cluster namespace and DataNodes to store data.

Overview of HDFS

Hadoop Distributed File System (HDFS) is a Java-based file system for storing large volumes of data. Designed to span large clusters of commodity servers, HDFS provides scalable and reliable data storage.

HDFS forms the data management layer of Apache Hadoop. YARN provides the resource management while HDFS provides the storage.

HDFS is a scalable, fault-tolerant, distributed storage system that works closely with a wide variety of concurrent data access applications. By distributing storage and computation across many servers, the combined storage resource grows linearly with demand.

Components of an HDFS cluster

An HDFS cluster contains the following main components: a NameNode and DataNodes.

The NameNode manages the cluster metadata that includes file and directory structures, permissions, modifications, and disk space quotas. The file content is split into multiple data blocks, with each block replicated at multiple DataNodes.

The NameNode actively monitors the number of replicas of a block. In addition, the NameNode maintains the namespace tree and the mapping of blocks to DataNodes, holding the entire namespace image in RAM.

High-Availability (HA) clusters contain a standby for the active NameNode to avoid a single point of failure. These clusters use JournalNodes to synchronize the active and standby NameNodes.

Benefits of HDFS

HDFS provides the following benefits as a result of which data is stored efficiently and is highly available in the cluster:

- Rack awareness: A node's physical location is considered when allocating storage and scheduling tasks.
- Minimal data motion: Hadoop moves compute processes to the data on HDFS. Processing tasks can occur on the physical node where the data resides. This significantly reduces network I/O and provides very high aggregate bandwidth.
- Utilities: Dynamically diagnose the health of the file system and rebalance the data on different nodes.
- Version rollback: Allows operators to perform a rollback to the previous version of HDFS after an upgrade, in case of human or systemic errors.
- Standby NameNode: Provides redundancy and supports high availability (HA).
- Operability: HDFS requires minimal operator intervention, allowing a single operator to maintain a cluster of thousands of nodes

Related Information

Apache Hadoop HDFS

NameNodes

NameNodes maintain the namespace tree for HDFS and a mapping of file blocks to DataNodes where the data is stored.

A simple HDFS cluster can have only one primary NameNode, supported by a secondary NameNode that periodically compresses the NameNode edits log file that contains a list of HDFS metadata modifications. This reduces the amount of disk space consumed by the log file on the NameNode, which also reduces the restart time for the primary NameNode. A high-availability cluster contains two NameNodes: active and standby.

Moving NameNode roles

Depending on your cluster configuration, you can follow either of the two approaches to migrate the NameNode role: using the Cloudera Manager wizard in a highly available cluster to automate the migration process, or manually moving the NameNode role to a new host.

Important: Both procedures require cluster downtime.

Moving highly available NameNode, failover controller, and JournalNode roles using the Migrate Roles wizard

The Migrate Roles wizard allows you to move roles of a highly available HDFS service from one host to another. You can use it to move NameNode, JournalNode, and Failover Controller roles.

Before you begin

- This procedure requires cluster downtime, not a shutdown. The services discussed in this list must be running for the migration to complete.
- The configuration of HDFS and services that depend on it must be valid.
- The destination host must be commissioned and healthy.
- The NameNode must be highly available using quorum-based storage.
- HDFS automatic failover must be enabled, and the cluster must have a running ZooKeeper service.
- If a Hue service is present in the cluster, its HDFS Web Interface Role property must refer to an HttpFS role, not to a NameNode role.
- A majority of configured JournalNode roles must be running.
- The Failover Controller role that is not located on the source host must be running.
- On hosts running active and standby NameNodes, back up the data directories.
- On hosts running JournalNodes, back up the JournalNode edits directory.
- If the source host is not functioning properly, or is not reliably reachable, decommission the host.
- If CDP and HDFS metadata was recently upgraded, and the metadata upgrade was not finalized, finalize the metadata upgrade.

About this task

Note: Nameservice federation (multiple namespaces) is not supported.

Procedure

1. If the host to which you want to move the NameNode is not in the cluster, add the required host.
2. Go to the HDFS service.
3. Click the Instances tab.
4. Click the Migrate Roles button.

5. Click the Source Host text field and specify the host running the roles to migrate.

   In the Search field optionally enter hostnames to filter the list of hosts and click Search.

   Select the checkboxes next to the desired host. The list of available roles to migrate displays. Clear any roles you do not want to migrate. When migrating a NameNode, the co-located Failover Controller must be migrated as well.
6. Click the Destination Host text field and specify the host to which the roles will be migrated.

   On destination hosts, indicate whether to delete data in the NameNode data directories and JournalNode edits directory. If you choose not to delete data and such role data exists, the Migrate Roles command will not complete successfully.
7. Acknowledge that the migration process incurs service unavailability by selecting the Yes, I am ready to restart the cluster now checkbox.
8. Click Continue. The Command Progress screen displays listing each step in the migration process.
9. When the migration completes, click Finish.

What to do next

After moving a NameNode, if you have a Hive or Impala service, perform the following steps:

1. Go to the Hive service.
2. Stop the Hive service.
3. Select Actions Update Hive Metastore NameNodes .
4. If you have an Impala service, restart the Impala service or run an INVALIDATE METADATA query.

Moving a NameNode to a different host using Cloudera Manager

You can use Cloudera Manager to manually move a NameNode from one host to another.

Before you begin

This procedure requires cluster downtime.

Procedure

1. If the host to which you want to move the NameNode is not in the cluster, add the required host.
2. Stop all the cluster services.
3. Make a backup of the dfs.name.dir directories on the existing NameNode host.

   Make sure you back up the fsimage and edits files. They should be the same across all of the directories specified by the dfs.name.dir property.
4. Copy the files you backed up from dfs.name.dir directories on the old NameNode host to the host where you want to run the NameNode.
5. Go to the HDFS service.
6. Click the Instances tab.
7. Select the checkbox next to the NameNode role instance and then click the Delete button. Click Delete again to confirm.
8. In the Review configuration changes page that appears, click Skip.
9. Click Add Role Instances to add a NameNode role instance.
10. Select the host where you want to run the NameNode and then click Continue.
11. Specify the location of the dfs.name.dir directories where you copied the data on the new host, and then click Accept Changes.
12. Start the cluster services.

    After the HDFS service has started, Cloudera Manager distributes the new configuration files to the DataNodes, which will be configured with the IP address of the new NameNode host.

Sizing NameNode heap memory

Each workload has a unique byte-distribution profile. Some workloads can use the default JVM settings for heap memory and garbage collection, but others require tuning. You can size your NameNode JVM if the dynamic heap settings cause a bottleneck.

All Hadoop processes run on a Java Virtual Machine (JVM). Each daemon runs on its own JVM across a cluster of hosts.

The legacy NameNode configuration is one active (and primary) NameNode for the entire namespace and one Secondary NameNode for checkpoints (but not failover). The recommended high-availability configuration replaces the Secondary NameNode with a standby NameNode that prevents a single point of failure. Each NameNode uses its own JVM.

Environment variables for sizing NameNode heap memory

You can configure the values of *HADOOP_HEAPSIZE* and *HADOOP_NAMENODE_OPTS* to size the NameNode heap memory.

HADOOP_HEAPSIZE sets the JVM heap size for all Hadoop project servers such as HDFS, YARN, and MapReduce. *HADOOP_HEAPSIZE* is an integer passed to the JVM as the maximum memory (Xmx) argument. For example:

```
HADOOP_HEAPSIZE=1024
```

HADOOP_NAMENODE_OPTS is specific to the NameNode and sets all JVM flags, which must be specified. *HADOOP_NAMENODE_OPTS* overrides the *HADOOP_HEAPSIZE* Xmx value for the NameNode. For example:

```
HADOOP_NAMENODE_OPTS=-Xms1024m -Xmx1024m -XX:+UseParNewGC -XX:+UseConcMarkSw
eepGC -XX:CMSInitiatingOccupancyFraction=70 -XX:+CMSParallelRemarkEnabled -X
X:+PrintTenuringDistribution -XX:OnOutOfMemoryError={{AGENT_COMMON_DIR}}/kil
lparent.sh
```

Both *HADOOP_NAMENODE_OPTS* and *HADOOP_HEAPSIZE* are stored in /etc/hadoop/conf/hadoop-env.sh.

Monitoring heap memory usage

You can use several ways to monitor the heap memory usage: Cloudera Manager, NameNode web UI, or the command line.

- Cloudera Manager: Look at the NameNode chart for heap memory usage. If you need to build the chart from scratch, run:

  ```
  select jvm_max_memory_mb, jvm_heap_used_mb where roleType="NameNode"
  ```

- NameNode Web UI: Scroll down to the Summary and look for "Heap Memory used."
- Command line: Generate a heap dump.

Files and directories

Persistence of HDFS metadata is implemented using fsimage file and edits files.

Attention:

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 consist of two categories of files:

fsimage

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 file

Contains 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, to create a new fsimage. Checkpointing is triggered automatically by configuration policies or manually by HDFS administration commands.

Disk space versus namespace

The HDFS default block size (*dfs.blocksize*) is set to 128 MB. Each namespace object on the NameNode consumes approximately 150 bytes.

On DataNodes, data files are measured by disk space consumed—the actual data length—and not necessarily the full block size. For example, a file that is 192 MB consumes 192 MB of disk space and not some integral multiple of the block size. Using the default block size of 128 MB, a file of 192 MB is split into two block files, one 128 MB file and one 64 MB file. On the NameNode, namespace objects are measured by the number of files and blocks. The same 192 MB file is represented by three namespace objects (1 file inode + 2 blocks) and consumes approximately 450 bytes of memory.

Large files split into fewer blocks generally consume less memory than small files that generate many blocks. One data file of 128 MB is represented by two namespace objects on the NameNode (1 file inode + 1 block) and consumes approximately 300 bytes of memory. By contrast, 128 files of 1 MB each are represented by 256 namespace objects (128 file inodes + 128 blocks) and consume approximately 38,400 bytes. The optimal split size, then, is some integral multiple of the block size, for memory management as well as data locality optimization.

By default, Cloudera Manager allocates a maximum heap space of 1 GB for every million blocks (but never less than 1 GB). How much memory you actually need depends on your workload, especially on the number of files, directories, and blocks generated in each namespace. If all of your files are split at the block size, you could allocate 1 GB for every million files. But given the historical average of 1.5 blocks per file (2 block objects), a more conservative estimate is 1 GB of memory for every million blocks.

Important: Cloudera recommends 1 GB of NameNode heap space per million blocks to account for the namespace objects, necessary bookkeeping data structures, and the remote procedure call (RPC) workload. In practice, your heap requirements will likely be less than this conservative estimate.

Replication

The default block replication factor (*dfs.replication*) is three. Replication affects disk space but not memory consumption.

Replication changes the amount of storage required for each block but not the number of blocks. If one block file on a DataNode, represented by one block on the NameNode, is replicated three times, the number of block files is tripled but not the number of blocks that represent them.

With replication off, one file of 192 MB consumes 192 MB of disk space and approximately 450 bytes of memory. If you have one million of these files, or 192 TB of data, you need 192 TB of disk space and, without considering the RPC workload, 450 MB of memory: (1 million inodes + 2 million blocks) * 150 bytes. With default replication on, you need 576 TB of disk space: (192 TB * 3) but the memory usage stay the same, 450 MB. When you account for bookkeeping and RPCs, and follow the recommendation of 1 GB of heap memory for every million blocks, a much safer estimate for this scenario is 2 GB of memory (with or without replication).

Examples of estimating NameNode heap memory

You can estimate the heap memory by consider either the number of file inodes and blocks, or the cluster capacity.

Example 1: Estimating NameNode heap memory used

Alice, Bob, and Carl each have 1 GB (1024 MB) of data on disk, but sliced into differently sized files. Alice and Bob have files that are some integral of the block size and require the least memory. Carl does not and fills the heap with unnecessary namespace objects.

Alice: 1 x 1024 MB file

- 1 file inode
- 8 blocks (1024 MB / 128 MB)

Total = 9 objects * 150 bytes = 1,350 bytes of heap memory

Bob: 8 x 128 MB files

- 8 file inodes
- 8 blocks

Total = 16 objects * 150 bytes = 2,400 bytes of heap memory

Carl: 1,024 x 1 MB files

- 1,024 file inodes
- 1,024 blocks

Total = 2,048 objects * 150 bytes = 307,200 bytes of heap memory

Example 2: Estimating NameNode heap memory needed

In this example, memory is estimated by considering the capacity of a cluster. Values are rounded. Both clusters physically store 4800 TB, or approximately 36 million block files (at the default block size). Replication determines how many namespace blocks represent these block files.

Cluster A: 200 hosts of 24 TB each = 4800 TB.

- Blocksize=128 MB, Replication=1
- Cluster capacity in MB: 200 * 24,000,000 MB = 4,800,000,000 MB (4800 TB)
- Disk space needed per block: 128 MB per block * 1 = 128 MB storage per block
- Cluster capacity in blocks: 4,800,000,000 MB / 128 MB = 36,000,000 blocks

At capacity, with the recommended allocation of 1 GB of memory per million blocks, Cluster A needs 36 GB of maximum heap space.

Cluster B: 200 hosts of 24 TB each = 4800 TB.

- Blocksize=128 MB, Replication=3
- Cluster capacity in MB: 200 * 24,000,000 MB = 4,800,000,000 MB (4800 TB)
- Disk space needed per block: 128 MB per block * 3 = 384 MB storage per block
- Cluster capacity in blocks: 4,800,000,000 MB / 384 MB = 12,000,000 blocks

At capacity, with the recommended allocation of 1 GB of memory per million blocks, Cluster B needs 12 GB of maximum heap space.

Both Cluster A and Cluster B store the same number of block files. In Cluster A, however, each block file is unique and represented by one block on the NameNode; in Cluster B, only one-third are unique and two-thirds are replicas.

DataNodes

DataNodes store data in a Hadoop cluster and is the name of the daemon that manages the data. File data is replicated on multiple DataNodes for reliability and so that localized computation can be executed near the data.

Within a cluster, DataNodes should be uniform. If they are not uniform, issues can occur. For example, DataNodes with less memory fill up more quickly than DataNodes with more memory, which can result in job failures.

Important: The default replication factor for HDFS is three. That is, three copies of data are maintained at all times. Cloudera recommends that you do not configure a lower replication factor when you have at least three DataNodes. A lower replication factor may lead to data loss.

How NameNode manages blocks on a failed DataNode

A DataNode is considered dead after a set period without any heartbeats (10.5 minutes by default).

When this happens, the NameNode performs the following actions to maintain the configured replication factor (3x replication by default):

1. The NameNode determines which blocks were on the failed DataNode.
2. The NameNode locates other DataNodes with copies of these blocks.
3. The DataNodes with block copies are instructed to copy those blocks to other DataNodes to maintain the configured replication factor.

Keep the following in mind when working with dead DataNodes:

- If a DataNode fails to heartbeat for reasons other than disk failure, it needs to be recommissioned to be added back to the cluster.
- If a DataNode rejoins the cluster, there is a possibility for surplus replicas of blocks that were on that DataNode. The NameNode will randomly remove excess replicas adhering to Rack-Awareness policies.

Remove a DataNode

Before removing a DataNode, ensure that all the prerequisites for removal are satisfied.

Before you begin

- The number of DataNodes in your cluster must be greater than or equal to the replication factor you have configured for HDFS. (This value is typically 3.)

  In order to satisfy this requirement, add the DataNode role on other hosts as required and start the role instances before removing any DataNodes.
- Ensure the DataNode that is to be removed is running.

Procedure

1. Decommission the DataNode role.

   When asked to select the role instance to decommission, select the DataNode role instance.

   The decommissioning process moves the data blocks to the other available DataNodes.

   

   Important: There must be at least as many DataNodes running as the replication factor or the decommissioning process will not complete.
2. Stop the DataNode role.

   When asked to select the role instance to stop, select the DataNode role instance.
3. Verify the integrity of the HDFS service.
   a) Run the following command to identify any problems in the HDFS file system:

   ```
   hdfs fsck /
   ```

   b) Fix any errors reported by the fsck command.

   If required, create a Cloudera support case.

4. After all errors are resolved, perform the following steps.
   a) Remove the DataNode role.
   b) Manually remove the DataNode data directories.

      You can determine the location of these directories by examining the DataNode Data Directory property in the HDFS configuration. In Cloudera Manager, go to the HDFS service, select the Configuration tab and search for the property.

Add storage directories using Cloudera Manager

You can add a new storage directory and specify the storage type using Cloudera Manager.

Procedure

1. Go to the HDFS service.
2. Click the Configuration tab.
3. Select Scope DataNode .
4. Add the new storage directory to the DataNode Data Directory property.

   To specify the storage type for the HDFS heterogeneous storage, add the storage type, surrounded by brackets, at the front of the path. For example: [SSD]/data/example_dir/.
5. Enter a Reason for change, and then click Save Changes to commit the changes.
6. Restart the DataNode.

Important: You must restart the DataNodes for heterogeneous storage configuration changes to take effect.

Remove storage directories using Cloudera Manager

You can use Cloudera Manager to remove existing storage directories and specify new directories.

Procedure

1. Stop the cluster.
2. Go to the HDFS service.
3. Click the Configuration tab.
4. Select Scope DataNode .
5. Remove the current directories and add new ones to the DataNode Data Directory property.
6. Enter a Reason for change, and then click Save Changes to commit the changes.
7. Copy the contents under the old directory to the new directory.
8. Start the cluster.

JournalNodes

High-availability clusters use JournalNodes to synchronize active and standby NameNodes. The active NameNode writes to each JournalNode with changes, or "edits," to HDFS namespace metadata. During failover, the standby NameNode applies all edits from the JournalNodes before promoting itself to the active state.

Moving the JournalNode edits directory for a role group using Cloudera Manager

Depending on your requirements, you can change the location of the edits directory for each JournalNode in the JournalNode Default Group.

Procedure

1. Stop all services on the cluster in Cloudera Manager.
   a) Go to the Cluster.
   b) Select Actions Stop .
2. Find the list of JournalNode hosts.
   a) Go to the HDFS service.
   b) Click JournalNode under Status Summary.
3. Move the location of each JournalNode (jn) directory at the command line.
   a) Connect to each host with a JournalNode.
   b) Per host, copy the JournalNode (jn) directory to its new location with the -a option to preserve permissions.

      ```
      cp -a /<old_path_to_jn_dir>/jn /<new_path_to_jn_dir>/jn
      ```

   c) Per host, rename the old jn directory to avoid confusion.

      ```
      mv /<old_path_to_jn_dir>/jn /<old_path_to_jn_dir>/jn_to_delete
      ```

4. Reconfigure the JournalNode Default Group.
   a) Go to the HDFS service.
   b) Click the Configuration tab.
   c) Click JournalNode under Scope.
   d) Set dfs.journalnode.edits.dir to the path of the new jn directory for all JournalNodes in the group.
   e) Click Save Changes.
5. Redeploy the client configuration for the cluster.
   a) Go to the Cluster.
   b) Select Actions Deploy Client Configuration .
6. Start all services on the cluster by selecting Actions Start .
7. Delete the old jn_to_delete directories from the command line.

Moving the JournalNode edits directory for a role instance using Cloudera Manager

Depending on your requirements, you can change the location of the edits directory for one JournalNode instance.

Procedure

1. Reconfigure the JournalNode Edits Directory.
   a) Go to the HDFS service in Cloudera Manager.
   b) Click JournalNode under Status Summary.
   c) Click the JournalNode link for the instance you are changing.
   d) Click the Configuration tab.
   e) Set dfs.journalnode.edits.dir to the path of the new jn directory.

2. Move the location of the JournalNode (jn) directory at the command line.
   a) Connect to host of the JournalNode.
   b) Copy the JournalNode (jn) directory to its new location with the -a option to preserve permissions.

      ```
      cp -a /<old_path_to_jn_dir>/jn /<new_path_to_jn_dir>/jn
      ```

   c) Rename the old jn directory to avoid confusion.

      ```
      mv /<old_path_to_jn_dir>/jn /<old_path_to_jn_dir>/jn_to_delete
      ```

3. Redeploy the HDFS client configuration.
   a) Go to the HDFS service.
   b) Select Actions Deploy Client Configuration .
4. Perform a rolling restart for HDFS by selecting Actions Rolling Restart .
   Use the default settings.
5. From the command line, delete the old jn_to_delete directory.