CHAPTER 1

Hadoop Distributed File System (HDFS)

The Hadoop Distributed File System (HDFS) is a Java-based distributed, scalable, and portable filesystem designed to span large clusters of commodity servers. The design of HDFS is based on GFS, the Google File System, which is described in a paper published by Google. Like many other distributed filesystems, HDFS holds a large amount of data and provides transparent access to many clients distributed across a network. Where HDFS excels is in its ability to store very large files in a reliable and scalable manner.

HDFS is designed to store a lot of information, typically petabytes (for very large files), gigabytes, and terabytes. This is accomplished by using a block-structured filesystem. Individual files are split into fixed-size blocks that are stored on machines across the cluster. Files made of several blocks generally do not have all of their blocks stored on a single machine.

HDFS ensures reliability by replicating blocks and distributing the replicas across the cluster. The default replication factor is three, meaning that each block exists three times on the cluster. Block-level replication enables data availability even when machines fail.

This chapter begins by introducing the core concepts of HDFS and explains how to interact with the filesystem using the native built-in commands. After a few examples, a Python client library is introduced that enables HDFS to be accessed programmatically from within Python applications.

Overview of HDFS

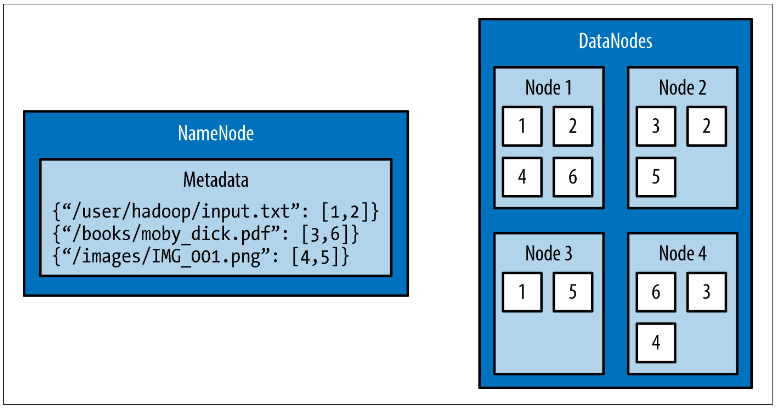
The architectural design of HDFS is composed of two processes: a process known as the NameNode holds the metadata for the filesystem, and one or more DataNode processes store the blocks that make up the files. The NameNode and DataNode processes can run on a single machine, but HDFS clusters commonly consist of a dedicated server running the NameNode process and possibly thousands of machines running the DataNode process.

The NameNode is the most important machine in HDFS. It stores metadata for the entire filesystem: filenames, file permissions, and the location of each block of each file. To allow fast access to this information, the NameNode stores the entire metadata structure in memory. The NameNode also tracks the replication factor of blocks, ensuring that machine failures do not result in data loss. Because the NameNode is a single point of failure, a secondary NameNode can be used to generate snapshots of the primary NameNode's memory structures, thereby reducing the risk of data loss if the NameNode fails.

The machines that store the blocks within HDFS are referred to as DataNodes. DataNodes are typically commodity machines with large storage capacities. Unlike the NameNode, HDFS will continue to operate normally if a DataNode fails. When a DataNode fails, the NameNode will replicate the lost blocks to ensure each block meets the minimum replication factor.

The example in Figure 1-1 illustrates the mapping of files to blocks in the NameNode, and the storage of blocks and their replicas within the DataNodes.

The following section describes how to interact with HDFS using the built-in commands.



NameNode
Metadata
{"/user/hadoop/input.txt": [1,2]}
{"/books/moby_dick.pdf": [3,6]}
{"/images/IMG_001.png": [4,5]}
DataNodes
Node 1
1
2
4
6
Node 2
3
2
5
Node 3
1
5
Node 4
6
3
4

Figure 1-1. An HDFS cluster with a replication factor of two; the NameNode contains the mapping of files to blocks, and the DataNodes store the blocks and their replicas

Interacting with HDFS

Interacting with HDFS is primarily performed from the command line using the script named `hdfs`. The `hdfs` script has the following usage:

```
$ hdfs COMMAND [-option <arg>]
```

The `COMMAND` argument instructs which functionality of HDFS will be used. The `-option` argument is the name of a specific option for the specified command, and `<arg>` is one or more arguments that that are specified for this option.

Common File Operations

To perform basic file manipulation operations on HDFS, use the `dfs` command with the `hdfs` script. The `dfs` command supports many of the same file operations found in the Linux shell.

It is important to note that the `hdfs` command runs with the permissions of the system user running the command. The following examples are run from a user named "hduser."

List Directory Contents

To list the contents of a directory in HDFS, use the `-ls` command:

```
$ hdfs dfs -ls
$
```

Running the `-ls` command on a new cluster will not return any results. This is because the `-ls` command, without any arguments, will attempt to display the contents of the user's home directory on HDFS. This is not the same home directory on the host machine (e.g., */home/$USER*), but is a directory within HDFS.

Providing `-ls` with the forward slash (`/`) as an argument displays the contents of the root of HDFS:

```
$ hdfs dfs -ls /
Found 2 items
drwxr-xr-x   - hadoop supergroup    0 2015-09-20 14:36 /hadoop
drwx------   - hadoop supergroup    0 2015-09-20 14:36 /tmp
```

The output provided by the `hdfs dfs` command is similar to the output on a Unix filesystem. By default, `-ls` displays the file and folder permissions, owners, and groups. The two folders displayed in this example are automatically created when HDFS is formatted. The `hadoop` user is the name of the user under which the Hadoop daemons were started (e.g., NameNode and DataNode), and the `supergroup` is the name of the group of superusers in HDFS (e.g., hadoop).

Creating a Directory

Home directories within HDFS are stored in */user/$HOME*. From the previous example with `-ls`, it can be seen that the */user* directory does not currently exist. To create the */user* directory within HDFS, use the `-mkdir` command:

```
$ hdfs dfs -mkdir /user
```

To make a home directory for the current user, `hduser`, use the `-mkdir` command again:

```
$ hdfs dfs -mkdir /user/hduser
```

Use the `-ls` command to verify that the previous directories were created:

```
$ hdfs dfs -ls -R /user
drwxr-xr-x    - hduser supergroup     0 2015-09-22 18:01 /user/
hduser
```

Copy Data onto HDFS

After a directory has been created for the current user, data can be uploaded to the user's HDFS home directory with the `-put` command:

```
$ hdfs dfs -put /home/hduser/input.txt /user/hduser
```

This command copies the file */home/hduser/input.txt* from the local filesystem to */user/hduser/input.txt* on HDFS.

Use the `-ls` command to verify that *input.txt* was moved to HDFS:

```
$ hdfs dfs -ls
Found 1 items
-rw-r--r--   1 hduser supergroup          52 2015-09-20 13:20
input.txt
```

Retrieving Data from HDFS

Multiple commands allow data to be retrieved from HDFS. To simply view the contents of a file, use the `-cat` command. `-cat` reads a file on HDFS and displays its contents to stdout. The following command uses `-cat` to display the contents of */user/hduser/input.txt*:

```
$ hdfs dfs -cat input.txt
jack be nimble
jack be quick
jack jumped over the candlestick
```

Data can also be copied from HDFS to the local filesystem using the `-get` command. The `-get` command is the opposite of the `-put` command:

```
$ hdfs dfs -get input.txt /home/hduser
```

This command copies *input.txt* from */user/hduser* on HDFS to */home/hduser* on the local filesystem.

HDFS Command Reference

The commands demonstrated in this section are the basic file operations needed to begin using HDFS. Below is a full listing of file manipulation commands possible with `hdfs dfs`. This listing can also be displayed from the command line by specifying `hdfs dfs` without any arguments. To get help with a specific option, use either `hdfs dfs -usage <option>` or `hdfs dfs -help <option>`.

```
Usage: hadoop fs [generic options]
    [-appendToFile <localsrc> ... <dst>]
    [-cat [-ignoreCrc] <src> ...]
    [-checksum <src> ...]
    [-chgrp [-R] GROUP PATH...]
    [-chmod [-R] <MODE[,MODE]... | OCTALMODE> PATH...]
    [-chown [-R] [OWNER][:[GROUP]] PATH...]
    [-copyFromLocal [-f] [-p] [-l] <localsrc> ... <dst>]
        [-copyToLocal  [-p]  [-ignoreCrc]  [-crc]  <src>  ...
<localdst>]
    [-count [-q] [-h] <path> ...]
    [-cp [-f] [-p | -p[topax]] <src> ... <dst>]
    [-createSnapshot <snapshotDir> [<snapshotName>]]
    [-deleteSnapshot <snapshotDir> <snapshotName>]
    [-df [-h] [<path> ...]]
    [-du [-s] [-h] <path> ...]
    [-expunge]
    [-find <path> ... <expression> ...]
    [-get [-p] [-ignoreCrc] [-crc] <src> ... <localdst>]
    [-getfacl [-R] <path>]
    [-getfattr [-R] {-n name | -d} [-e en] <path>]
    [-getmerge [-nl] <src> <localdst>]
    [-help [cmd ...]]
    [-ls [-d] [-h] [-R] [<path> ...]]
    [-mkdir [-p] <path> ...]
    [-moveFromLocal <localsrc> ... <dst>]
    [-moveToLocal <src> <localdst>]
    [-mv <src> ... <dst>]
    [-put [-f] [-p] [-l] <localsrc> ... <dst>]
    [-renameSnapshot <snapshotDir> <oldName> <newName>]
    [-rm [-f] [-r|-R] [-skipTrash] <src> ...]
    [-rmdir [--ignore-fail-on-non-empty] <dir> ...]
     [-setfacl [-R] [{-b|-k} {-m|-x <acl_spec>} <path>]|[--set
<acl_spec> <path>]]
    [-setfattr {-n name [-v value] | -x name} <path>]
    [-setrep [-R] [-w] <rep> <path> ...]
    [-stat [format] <path> ...]
    [-tail [-f] <file>]
    [-test -[defsz] <path>]
    [-text [-ignoreCrc] <src> ...]
    [-touchz <path> ...]
    [-truncate [-w] <length> <path> ...]
    [-usage [cmd ...]]

Generic options supported are
-conf <configuration file>      specify an application configu-
ration file
-D <property=value>            use value for given property
-fs <local|namenode:port>      specify a namenode
-jt <local|resourcemanager:port>    specify a ResourceManager
-files <comma separated list of files>    specify comma separa-
```

```
ted files to be copied to the map reduce cluster
-libjars <comma separated list of jars>     specify comma sepa-
rated jar files to include in the classpath.
-archives <comma separated list of archives>    specify comma
separated archives to be unarchived on the compute machines.

The general command line syntax is
bin/hadoop command [genericOptions] [commandOptions]
```

The next section introduces a Python library that allows HDFS to be accessed from within Python applications.

Snakebite

Snakebite is a Python package, created by Spotify, that provides a Python client library, allowing HDFS to be accessed programmatically from Python applications. The client library uses protobuf messages to communicate directly with the NameNode. The Snakebite package also includes a command-line interface for HDFS that is based on the client library.

This section describes how to install and configure the Snakebite package. Snakebite's client library is explained in detail with multiple examples, and Snakebite's built-in CLI is introduced as a Python alternative to the `hdfs dfs` command.

Installation

Snakebite requires Python 2 and python-protobuf 2.4.1 or higher. Python 3 is currently not supported.

Snakebite is distributed through PyPI and can be installed using `pip`:

```
$ pip install snakebite
```

Client Library

The client library is written in Python, uses protobuf messages, and implements the Hadoop RPC protocol for talking to the NameNode. This enables Python applications to communicate directly with HDFS and not have to make a system call to `hdfs dfs`.

List Directory Contents

Example 1-1 uses the Snakebite client library to list the contents of the root directory in HDFS.

Example 1-1. python/HDFS/list_directory.py

```
from snakebite.client import Client

client = Client('localhost', 9000)
for x in client.ls(['/']):
   print x
```

The most important line of this program, and every program that uses the client library, is the line that creates a client connection to the HDFS NameNode:

```
client = Client('localhost', 9000)
```

The `Client()` method accepts the following parameters:

`host` (*string*)
: Hostname or IP address of the NameNode

`port` (*int*)
: RPC port of the NameNode

`hadoop_version` (*int*)
: The Hadoop protocol version to be used (default: 9)

`use_trash` (*boolean*)
: Use trash when removing files

`effective_use` (*string*)
: Effective user for the HDFS operations (default: None or current user)

The `host` and `port` parameters are required and their values are dependent upon the HDFS configuration. The values for these parameters can be found in the *hadoop/conf/core-site.xm*l configuration file under the property `fs.defaultFS`:

```
<property>
   <name>fs.defaultFS</name>
   <value>hdfs://localhost:9000</value>
</property>
```

For the examples in this section, the values used for `host` and `port` are `localhost` and `9000`, respectively.

After the client connection is created, the HDFS filesystem can be accessed. The remainder of the previous application used the `ls` command to list the contents of the root directory in HDFS:

```
    for x in client.ls(['/']):
       print x
```

It is important to note that many of methods in Snakebite return generators. Therefore they must be consumed to execute. The `ls` method takes a list of paths and returns a list of maps that contain the file information.

Executing the *list_directory.py* application yields the following results:

```
$ python list_directory.py
{'group': u'supergroup', 'permission': 448, 'file_type': 'd',
'access_time':    0L,    'block_replication':    0,    'modifica-
tion_time':  1442752574936L,  'length':  0L,  'blocksize':  0L,
'owner': u'hduser', 'path': '/tmp'}
{'group': u'supergroup', 'permission': 493, 'file_type': 'd',
'access_time':    0L,    'block_replication':    0,    'modifica-
tion_time':  1442742056276L,  'length':  0L,  'blocksize':  0L,
'owner': u'hduser', 'path': '/user'}
```

Create a Directory

Use the `mkdir()` method to create directories on HDFS. Example 1-2 creates the directories */foo/bar* and */input* on HDFS.

Example 1-2. python/HDFS/mkdir.py

```
from snakebite.client import Client

client = Client('localhost', 9000)
for p in client.mkdir(['/foo/bar', '/input'], create_parent=True):
   print p
```

Executing the *mkdir.py* application produces the following results:

```
$ python mkdir.py
{'path': '/foo/bar', 'result': True}
{'path': '/input', 'result': True}
```

The `mkdir()` method takes a list of paths and creates the specified paths in HDFS. This example used the `create_parent` parameter to ensure that parent directories were created if they did not already exist. Setting `create_parent` to `True` is analogous to the `mkdir -p` Unix command.

Deleting Files and Directories

Deleting files and directories from HDFS can be accomplished with the `delete()` method. Example 1-3 recursively deletes the */foo* and */bar* directories, created in the previous example.

Example 1-3. python/HDFS/delete.py

```
from snakebite.client import Client

client = Client('localhost', 9000)
for p in client.delete(['/foo', '/input'], recurse=True):
   print p
```

Executing the *delete.py* application produces the following results:

```
$ python delete.py
{'path': '/foo', 'result': True}
{'path': '/input', 'result': True}
```

Performing a recursive delete will delete any subdirectories and files that a directory contains. If a specified path cannot be found, the delete method throws a `FileNotFoundException`. If recurse is not specified and a subdirectory or file exists, `DirectoryException` is thrown.

The `recurse` parameter is equivalent to `rm -rf` and should be used with care.

Retrieving Data from HDFS

Like the `hdfs dfs` command, the client library contains multiple methods that allow data to be retrieved from HDFS. To copy files from HDFS to the local filesystem, use the `copyToLocal()` method. Example 1-4 copies the file */input/input.txt* from HDFS and places it under the */tmp* directory on the local filesystem.

Example 1-4. python/HDFS/copy_to_local.py

```
from snakebite.client import Client

client = Client('localhost', 9000)
for f in client.copyToLocal(['/input/input.txt'], '/tmp'):
   print f
```

Executing the *copy_to_local.py* application produces the following result:

```
$ python copy_to_local.py
{'path': '/tmp/input.txt', 'source_path': '/input/input.txt',
'result': True, 'error': ''}
```

To simply read the contents of a file that resides on HDFS, the `text()` method can be used. Example 1-5 displays the content of */input/input.txt*.

Example 1-5. python/HDFS/text.py

```
from snakebite.client import Client

client = Client('localhost', 9000)
for l in client.text(['/input/input.txt']):
   print l
```

Executing the *text.py* application produces the following results:

```
$ python text.py
jack be nimble
jack be quick
jack jumped over the candlestick
```

The `text()` method will automatically uncompress and display gzip and bzip2 files.

CLI Client

The CLI client included with Snakebite is a Python command-line HDFS client based on the client library. To execute the Snakebite CLI, the hostname or IP address of the NameNode and RPC port of the NameNode must be specified. While there are many ways to specify these values, the easiest is to create a *~.snakebiterc* configuration file. Example 1-6 contains a sample config with the NameNode hostname of `localhost` and RPC port of `9000`.

Example 1-6. ~/.snakebiterc

```
{
    "config_version": 2,
    "skiptrash": true,
    "namenodes": [
        {"host": "localhost", "port": 9000, "version": 9},
    ]
}
```

The values for host and port can be found in the *hadoop/conf/core-site.xml* configuration file under the property fs.defaultFS.

For more information on configuring the CLI, see the Snakebite CLI documentation online.

Usage

To use the Snakebite CLI client from the command line, simply use the command snakebite. Use the ls option to display the contents of a directory:

```
$ snakebite ls /
Found 2 items
drwx------    - hadoop    supergroup    0 2015-09-20 14:36 /tmp
drwxr-xr-x    - hadoop    supergroup    0 2015-09-20 11:40 /user
```

Like the hdfs dfs command, the CLI client supports many familiar file manipulation commands (e.g., ls, mkdir, df, du, etc.).

The major difference between snakebite and hdfs dfs is that snakebite is a pure Python client and does not need to load any Java libraries to communicate with HDFS. This results in quicker interactions with HDFS from the command line.

CLI Command Reference

The following is a full listing of file manipulation commands possible with the snakebite CLI client. This listing can be displayed from the command line by specifying snakebite without any arguments. To view help with a specific command, use snakebite [cmd] --help, where cmd is a valid snakebite command.

```
snakebite [general options] cmd [arguments]
general options:
  -D --debug                 Show debug information
  -V --version               Hadoop protocol version (default:9)
  -h --help                  show help
  -j --json                  JSON output
  -n --namenode              namenode host
  -p --port                  namenode RPC port (default: 8020)
  -v --ver                   Display snakebite version

commands:
  cat [paths]                    copy source paths to stdout
  chgrp <grp> [paths]            change group
  chmod <mode> [paths]           change file mode (octal)
  chown <owner:grp> [paths]      change owner
  copyToLocal [paths] dst        copy paths to local
```

```
                                  file system destination
      count [paths]             display stats for paths
      df                        display fs stats
      du [paths]                display disk usage statistics
      get file dst              copy files to local
                                  file system destination
      getmerge dir dst          concatenates files in source dir
                                  into destination local file
      ls [paths]                list a path
      mkdir [paths]             create directories
      mkdirp [paths]            create directories and their
                                  parents
      mv [paths] dst            move paths to destination
      rm [paths]                remove paths
      rmdir [dirs]              delete a directory
      serverdefaults            show server information
      setrep <rep> [paths]      set replication factor
      stat [paths]              stat information
      tail path                 display last kilobyte of the
                                  file to stdout
      test path                 test a path
      text path [paths]         output file in text format
      touchz [paths]            creates a file of zero length
      usage <cmd>               show cmd usage

    to see command-specific options use: snakebite [cmd] --help
```

Chapter Summary

This chapter introduced and described the core concepts of HDFS. It explained how to interact with the filesystem using the built-in `hdfs dfs` command. It also introduced the Python library, Snakebite. Snakebite's client library was explained in detail with multiple examples. The `snakebite` CLI was also introduced as a Python alternative to the `hdfs dfs` command.