HDFS

HDFS is a Filesystem of Hadoop designed for storing very large files running on a cluster of commodity hardware. It is designed on the principle of storage of a smaller number of large files rather than the huge number of small files.

Hadoop HDFS provides a fault-tolerant storage layer for Hadoop and its other components. HDFS Replication of data helps us to attain this feature. It stores data reliably, even in the case of hardware failure. It provides high throughput access to application data by providing the data access in parallel.

HDFS also has two types of nodes,

1. **NameNode**
2. **DataNodes**

1. HDFS Master (Namenode)

NameNode regulates file access to the clients. It maintains and manages the slave nodes and assigns tasks to them. NameNode executes file system namespace operations like opening, closing, and renaming files and directories.

2. HDFS Slave (Datanode)

There are n number of slaves (where n can be up to 1000) or DataNodes in the Hadoop Distributed File System that manages storage of data. These slave nodes are the actual worker nodes that do the tasks and serve read and write requests from the file system's clients.

They perform block creation, deletion, and replication upon instruction from the NameNode. Once a block is written on a DataNode, it replicates it to other DataNode, and the process continues until creating the required number of replicas.

Data storage in HDFS

Hadoop HDFS broke the files into small pieces of data known as blocks. The default block size in HDFS is 128 MB. We can configure the size of the block as per the requirements. These blocks are stored in the cluster in a distributed manner on different nodes. This provides a mechanism for MapReduce to process the data in parallel in the cluster.

Racks in Hadoop HDFS

Hadoop runs on a cluster of computers spread commonly across many racks.

NameNode places replicas of a block on multiple racks for improved fault tolerance.

NameNode tries to place at least one replica of a block in a different rack so that if a complete rack goes down, then also the system will be highly available.

HDFS Architecture

NameNode stores metadata, and DataNode stores actual data. The client interacts with NameNode for performing any tasks, as NameNode is the centerpiece in the cluster.

There are several DataNodes in the cluster which store HDFS data in the local disk. DataNode sends a heartbeat message to NameNode periodically to indicate that it is alive. Also, it replicates data to other DataNode as per the replication factor.

HDFS Read Operation

1. client needs to interact with NameNode as NameNode is the only place that stores metadata about DataNodes
2. NameNode specifies the address or the location of the slaves where data is stored.
3. The client will interact with the specified DataNodes and read the data from there.

AUTHENTICATION PROCESS

In the Hadoop HDFS read operation, if the client wants to read data that is stored in HDFS, it needs to interact with NameNode first. So, the client interacts with distributed file system API and sends a request to NameNode to send block location. Thus, NameNode checks if the client has sufficient privileges to access the data or not. If the client has sufficient privileges, then NameNode will share the address at which data is stored in the DataNode.

With the address, NameNode also shares a security token with the client, which it needs to show to DataNode before accessing the data for authentication purposes.

When a client goes to DataNode for reading the file, after checking the token, DataNode allows the client to read that block. A client then opens the input stream and starts reading data from the specified DataNodes. Hence, in this manner, the client reads data directly from DataNode.

During the reading of a file, if the DataNode goes down suddenly, then a client will again go to the NameNode, and the NameNode will share another location where that block is present.

HDFS Write Operation

1. for writing a file, the client needs to interact with the NameNode.
2. NameNode provides the address of the slaves on which data has to be written by the client.

3. Once the client finishes writing the block, the slave starts replicating the block into another slave, which then copies the block to the third slave. This is the case when the default replication factor of 3 is used.
4. After the required replicas are created, it sends a final acknowledgment to the client.

The authentication process is similar read process.

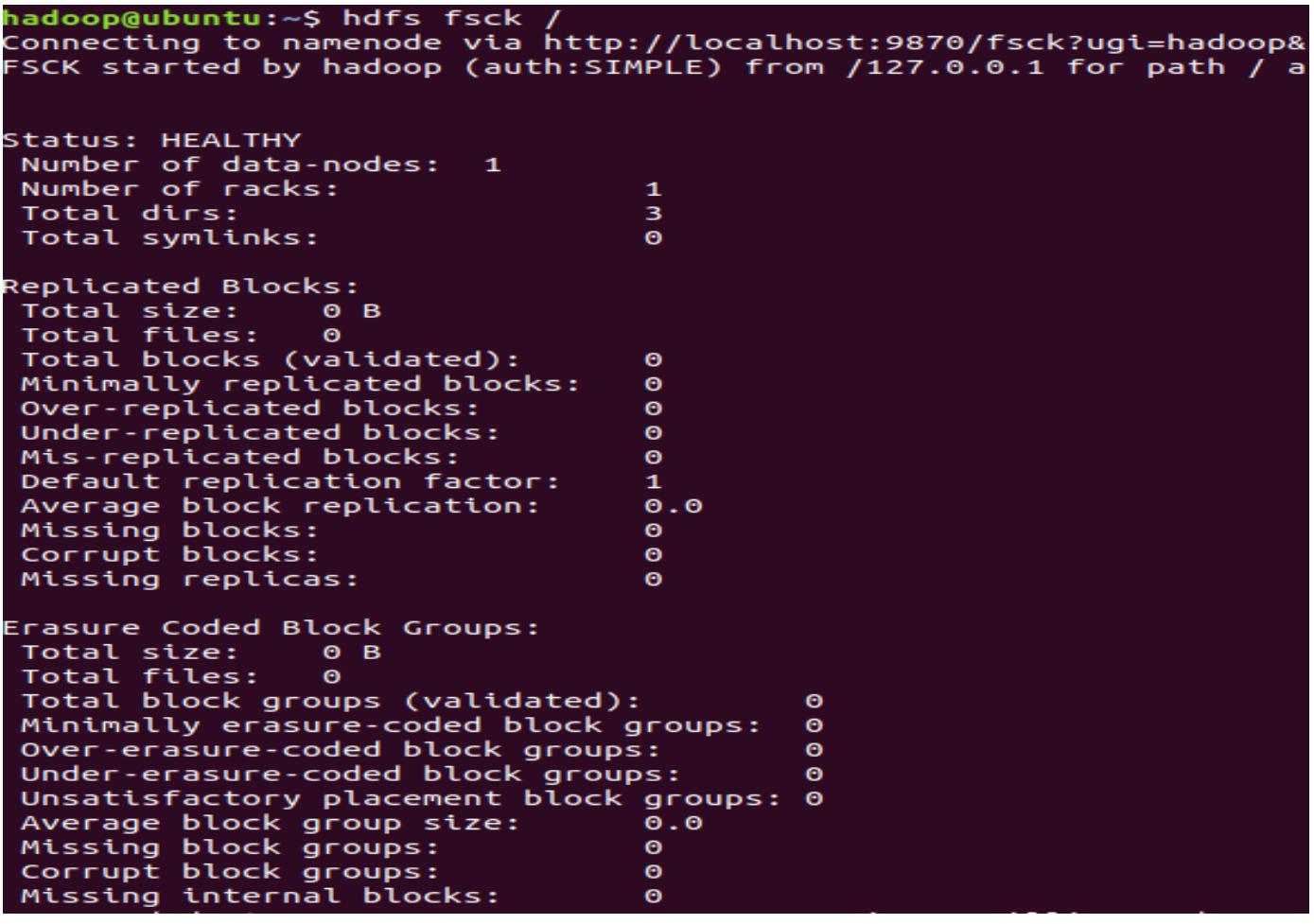
Hadoop HDFS Operations

Interacting with HDFS using command line interface.

All the important commands can be accessed using

1. hdfs dfs –help
2. To check summary of filesystem

   hdfs fsck /



3. To create and list directories

```
hadoop@ubuntu:~$ hdfs dfs -mkdir -p ~/test
hadoop@ubuntu:~$ hdfs dfs -ls ~/
Found 1 items
drwxr-xr-x   - hadoop supergroup          0 2020-12-08 16:54 /home/hadoop/test
hadoop@ubuntu:~$
```

4. To remove directory

```
hadoop@ubuntu:~$ hdfs dfs -rm -r ~/test/
Deleted /home/hadoop/test
hadoop@ubuntu:~$
```

5. To create empty file

```
hadoop@ubuntu:~$ clear

hadoop@ubuntu:~$ hdfs dfs -touchz ~/sample.txt
hadoop@ubuntu:~$ hdfs dfs -ls ~/
Found 1 items
-rw-r--r--   1 hadoop supergroup          0 2020-12-08 17:04 /home/hadoop/sample.txt
hadoop@ubuntu:~$
```

6. To append text into file

```
hadoop@ubuntu:~$ hdfs dfs -appendToFile - ~/sample.txt
I am writing to this file and excited to learn about
how hdfs file system works.hadoop@ubuntu:~$
```

7. To get size and read from file.

```
hadoop@ubuntu:~$ hdfs dfs -du -s ~/sample.txt
80  80  /home/hadoop/sample.txt
hadoop@ubuntu:~$ hdfs dfs -cat ~/sample.txt
I am writing to this file and excited to learn about
how hdfs file system works.hadoop@ubuntu:~$
```

8. Copying file from local system into hdfs directory

```
hadoop@ubuntu:~$ hdfs dfs -copyFromLocal host.txt ~/
hadoop@ubuntu:~$ hdfs dfs -ls ~/
Found 2 items
-rw-r--r--   1 hadoop supergroup         40 2020-12-08 18:40 /home/hadoop/host.txt
-rw-r--r--   1 hadoop supergroup         80 2020-12-08 18:33 /home/hadoop/sample.txt
hadoop@ubuntu:~$ vim readme.md
hadoop@ubuntu:~$ hdfs dfs -ls ~/
Found 2 items
-rw-r--r--   1 hadoop supergroup         40 2020-12-08 18:40 /home/hadoop/host.txt
-rw-r--r--   1 hadoop supergroup         80 2020-12-08 18:33 /home/hadoop/sample.txt
hadoop@ubuntu:~$ hdfs dfs -put readme.md ~/
hadoop@ubuntu:~$ hdfs dfs -ls ~/
Found 3 items
-rw-r--r--   1 hadoop supergroup         40 2020-12-08 18:40 /home/hadoop/host.txt
-rw-r--r--   1 hadoop supergroup         20 2020-12-08 18:43 /home/hadoop/readme.md
-rw-r--r--   1 hadoop supergroup         80 2020-12-08 18:33 /home/hadoop/sample.txt
```

9. Copy from hdfs to local directory

```
hadoop@ubuntu:~$ hdfs dfs -copyToLocal ~/readme.md ~/
copyToLocal: `/home/hadoop/readme.md': File exists
hadoop@ubuntu:~$ hdfs dfs -copyToLocal ~/readme.md ~/sample/
hadoop@ubuntu:~$ cd sample
hadoop@ubuntu:~/sample$ ls
readme.md
hadoop@ubuntu:~/sample$ hdfs dfs -get ~/readme.md ~/sample/
get: `/home/hadoop/sample/readme.md': File exists
hadoop@ubuntu:~/sample$ hdfs dfs -get ~/host.txt ~/sample/
hadoop@ubuntu:~/sample$ ls
host.txt  readme.md
```

10. To move file

```
hadoop@ubuntu:~/sample$ hdfs dfs -mv ~/host.txt ~/test/
hadoop@ubuntu:~/sample$ hdfs dfs -ls ~/test/
Found 1 items
-rw-r--r--   1 hadoop supergroup         40 2020-12-08 18:40 /home/hadoop/test/host.txt
```

Example: Running wordcount program inside Hadoop

To run wordcount sample program provided by Hadoop,

Hadoop can run program using mapreduce and can be run using below command,

1. Specify jar filename
2. Input folder having input file to run
3. Output folder where output will be created

$hadoop jar jarname –classpath input output

Map reduce can only run jar files.

Step 1: Create input file with some content,

```
hadoop@ubuntu:~/sample$ hdfs dfs -mkdir ~/myinput
hadoop@ubuntu:~/sample$ hdfs dfs -touchz ~/myinput/input.txt
hadoop@ubuntu:~/sample$ hdfs dfs -appendToFile - ~/myinput/input.txt
Mary had a little lamb
Little lamb, little lamb
Mary had a little lamb
It's fleece was white as snow
Everywhere that Mary went
Mary went, Mary went
Everywhere that Mary went
The lamb was sure to gohadoop@ubuntu:~/sample$
```

Step 2: Run wordcount jar sample program provided by Hadoop

```
hadoop@ubuntu:/usr/local/hadoop/share/hadoop/mapreduce$ hadoop jar hadoop-mapreduce-examples-3.1.4.jar wordcount ~/myinput ~/myoutput
2020-12-08 19:35:25,579 INFO client.RMProxy: Connecting to ResourceManager at /0.0.0.0:8032
2020-12-08 19:35:26,191 INFO mapreduce.JobResourceUploader: Disabling Erasure Coding for path: /tmp/hadoop-yarn/staging/hadoop/.staging
2020-12-08 19:35:26,398 INFO input.FileInputFormat: Total input files to process : 1
2020-12-08 19:35:26,514 INFO mapreduce.JobSubmitter: number of splits:1
2020-12-08 19:35:26,701 INFO mapreduce.JobSubmitter: Submitting tokens for job: job_1607229528034_0001
2020-12-08 19:35:26,702 INFO mapreduce.JobSubmitter: Executing with tokens: []
2020-12-08 19:35:26,892 INFO conf.Configuration: resource-types.xml not found
2020-12-08 19:35:26,892 INFO resource.ResourceUtils: Unable to find 'resource-types.xml'.
2020-12-08 19:35:27,522 INFO impl.YarnClientImpl: Submitted application application_1607229528034_0001
2020-12-08 19:35:27,579 INFO mapreduce.Job: The url to track the job: http://ubuntu:8088/proxy/application_1607229528034_0001/
2020-12-08 19:35:27,580 INFO mapreduce.Job: Running job: job_1607229528034_0001
2020-12-08 19:35:35,730 INFO mapreduce.Job: Job job_1607229528034_0001 running in uber mode : false
2020-12-08 19:35:35,733 INFO mapreduce.Job:  map 0% reduce 0%
```

Step 3: Get the output from output folder,

Output is saved in file part-r-00000

```
hadoop@ubuntu:/usr/local/hadoop/share/hadoop/mapreduce$ hdfs dfs -ls ~/myoutput
Found 2 items
-rw-r--r--   1 hadoop supergroup          0 2020-12-08 19:35 /home/hadoop/myoutput/_SUCCESS
-rw-r--r--   1 hadoop supergroup        150 2020-12-08 19:35 /home/hadoop/myoutput/part-r-00000
hadoop@ubuntu:/usr/local/hadoop/share/hadoop/mapreduce$ hdfs dfs -cat ~/myoutput/part*
Everywhere      2
It's    1
Little  1
Mary    6
The     1
a       2
as      1
fleece  1
go      1
had     2
lamb    4
lamb,   1
little  3
snow    1
sure    1
that    2
to      1
was     2
went    3
went,   1
white   1
```