|  |
| --- |
|  |
| Hadoop and HDFS |
|  |
|  |
|  |
|  |

Table of Contents

[Hadoop 3](#_Toc344298275)

[Hadoop Platform Brief 3](#_Toc344298276)

[Hadoop Distributed File System (HDFS) 4](#_Toc344298277)

[HDFS Architecture 4](#_Toc344298278)

[NameNode 4](#_Toc344298279)

[DataNode 5](#_Toc344298280)

[SecondaryNameNode 5](#_Toc344298281)

[MapReduce implementation 5](#_Toc344298282)

[JobTracker 5](#_Toc344298283)

[TaskTracker 5](#_Toc344298284)

[How to determine number of DataNodes? 6](#_Toc344298285)

[Why do HDFS clusters have only a single NameNode? 6](#_Toc344298286)

[Data Replication 6](#_Toc344298287)

[Replication Factor 7](#_Toc344298288)

[How to set the Replication Factor in the conf/hdfs-site.xml in the master? 7](#_Toc344298289)

[How to change replication factor of existing files in HDFS? 7](#_Toc344298290)

[NameNodeFailOver 8](#_Toc344298291)

[When a Failure Occurs 8](#_Toc344298292)

[Manual NameNode Metadata Recovery 8](#_Toc344298293)

[NameNode Recovery in Action 8](#_Toc344298294)

[DataNodeFailOver 9](#_Toc344298295)

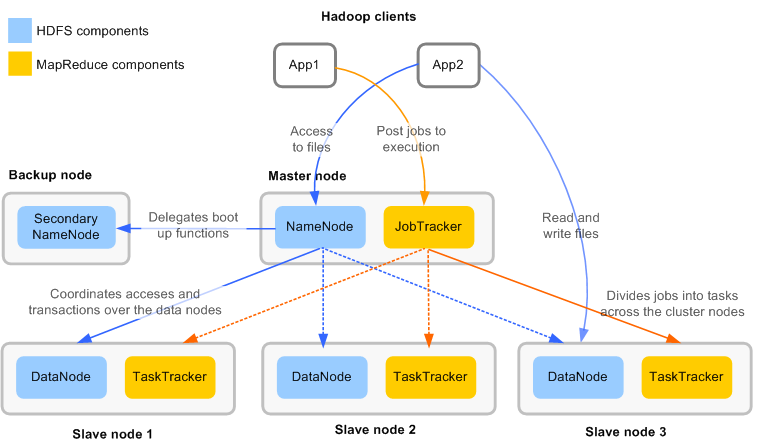
# Hadoop

The Apache Hadoop software library is a framework that allows for the distributed processing of large data sets across clusters of computers. It is designed to scale up from single servers to thousands of machines, each offering local computation and storage.

Hadoop analyze and process large amount of data i.e peta bytes of data in parallel with less time located in distributed environment.

Hadoop provides a distributed filesystem (**HDFS**) and a [**MapReduce**](http://www.vogella.com/articles/MapReduce/article.html)implementation.

# Hadoop Platform Brief



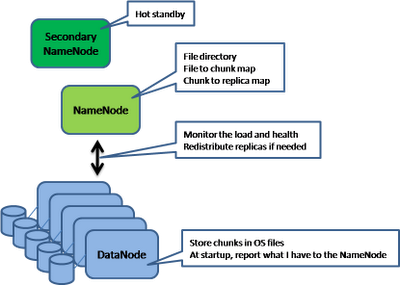
# Hadoop Distributed File System (HDFS)

Hadoop Distributed File System is a distributed file system which is designed to run on commodity hardware.

* HDFS is designed to reliably store very large files across machines in a large cluster.
* HDFS stores each file as a sequence of blocks; all blocks in a file except the last block are the same size.
* The blocks of a file are replicated for fault tolerance and this replication is configurable.

# HDFS Architecture

HDFS has master/slave architecture. An HDFS cluster consists of a single NameNode, a master server that manages the file system namespace and regulates access to files by clients. In addition, there are a number of Data Nodes, usually one per node in the cluster, which manage storage attached to the nodes that they run on.



## NameNode

* Namenode acts as master in HDFS.
* Manages the filesystem namespace and metadata.
* Types of metadata

• List of files

• List of blocks per file

• List of DataNodes for each block

• File attributes e.g: creation time, replication factor

* No data goes through the NameNode.
* **Only one** per Hadoop cluster**.**
* NameNode mainly consists of:

• fsimage: Contains the metadata on disk

• edit logs: Records all write operations, synchronizes with metadata in RAM after each write

## DataNode

* Datanode is the actual storage component in HDFS.
* Datanode store data in HDFS file system.
* Many per Hadoop cluster.
* Manages blocks with data and serves them to clients.
* Periodically reports to NameNode the list of blocks it stores.
* Use inexpensive commodity hardware for this node.

## SecondaryNameNode

* Helper to the primary NameNode.
* Responsible for supporting periodic checkpoints of the HDFS metadata.
* Only one Secondary NameNode per HDFS cluster.

# [MapReduce](http://www.vogella.com/articles/MapReduce/article.html)implementation

Above the file systems comes the [MapReduce](http://en.wikipedia.org/wiki/MapReduce) engine, which consists of JobTracker and TaskTracker.

## JobTracker

* One per Hadoop cluster.
* Receives job requests submitted by client.
* Schedules and monitors MapReduce jobs on task trackers.

## TaskTracker

* Many per Hadoop cluster.
* Accepts tasks – Map and Reduce operations - from a [JobTracker](http://wiki.apache.org/hadoop/JobTracker).
* Every TaskTracker is configured with a set of slots; these indicate the number of tasks that it can accept.
* Reads blocks from DataNodes.

# How to determine number of DataNodes?

A good way to determine the number of datanodes is to start from the planned data input of the cluster.

It is also important to note that for every disk, 30% of its capacity is reserved to non HDFS use.

Let’s consider the following hypothesis:

* Daily data input: 100Gb
* HDFS replication factor: 3
* Non HDFS reserved space per disk: 30%
* Size of a disk: 3Tb

With this hypothesis, we are able to determine the storage needed and the number of DataNodes.

Therefore we have:

* Storage space used by daily data input : <*daily data input*> \* <*replication factor*> = 300GB
* Size of a hard drive dedicated to HDFS : <*Size of the hard drive* > \* (1 – <*Non HDFS reserved space per disk*>) = 2.1TB
* Number of DataNodes after 1 year (no monthly growth) : <*storage space used by daily data input*> \* 365 / <*HDFS reserved space in a disk*> = 100TB / 2.1TB = 48 DataNodes

# [Why do HDFS clusters have only a single NameNode?](http://programmers.stackexchange.com/questions/142880/why-do-hdfs-clusters-have-only-a-single-namenode)

The NameNode is a Single Point of Failure for the HDFS Cluster.  It's a single point of failure because Hadoop does not support having two of them running cooperatively. Just having a secondary NameNode running isn't enough to provide high availability because there is no way to ensure consistency when the main one goes down (the secondary NameNode may be missing recent changes).

The existence of a single NameNode in a cluster greatly simplifies the architecture of the system. The NameNode is the arbitrator and repository for all HDFS metadata.

# Data Replication

HDFS is designed to reliably store very large files across machines in a large cluster. It stores each file as a sequence of blocks; all blocks in a file except the last block are the same size. The blocks of a file are replicated for fault tolerance. The block size and replication factor are configurable per file. An application can specify the number of replicas of a file. The replication factor can be specified at file creation time and can be changed later. Files in HDFS are write-once and have strictly one writer at any time.

The NameNode makes all decisions regarding replication of blocks.

## Replication Factor

The default replication factor in HDFS is 3.Replication factor can be configured in the conf/hdfs-site.xml in the master or we can set the replication level individually for each file in HDFS.

A replication factor of 3 means that you have 3 *total* copies of your data. More specifically, there will be 3 copies of each block for your file, so if your file is made up of 10 blocks there will be 30 total blocks across your 10 nodes, or about 3 blocks per node.

One advantage of writing to multiple nodes is that even if one of the nodes goes down, a couple of splits might be down, but at least some data can be recovered somehow from the remaining splits.

### How to set the Replication Factor in the conf/hdfs-site.xml in the master?

Having the replication config on the master should be sufficient (it is used at the NameNode, for default value purposes).

In file conf/hdfs-site.xml:

<!-- In: conf/hdfs-site.xml -->

<property>

<name>dfs.replication</name>

<value>1</value>

<description>Default block replication.

The actual number of replications can be specified when the file is created.

The default is used if replication is not specified in create time.

</description>

</property>

### How to change replication factor of existing files in HDFS?

To set replication of an individual file to 4:

./bin/hadoop dfs -setrep -w 4 /path/to/file

You can also do this recursively. To change replication of entire HDFS to 1:

./bin/hadoop dfs -setrep -R -w 1 /

# NameNodeFailOver

Hadoop does not support automatic recovery in the case of a [NameNode](http://wiki.apache.org/hadoop/NameNode) failure. This is a well-known and recognized single point of failure in Hadoop.

Experience at Yahoo! shows that [NameNodes](http://wiki.apache.org/hadoop/NameNodes) are more likely to fail due to misconfiguration, network issues, and bad behavior than actual hardware problems.

## When a Failure Occurs

Now the recovery steps:

* Just to be safe, make a copy of the data on the remote NFS mount for safe keeping.
* Pick a target machine on the same network.
* Change the IP address of that machine to match the [NameNode](http://wiki.apache.org/hadoop/NameNode)'s IP address.
* Install Hadoop similarly to how you did the [NameNode](http://wiki.apache.org/hadoop/NameNode).
* Do not format this node!
* Mount the remote NFS directory in the same location.
* Startup the [NameNode](http://wiki.apache.org/hadoop/NameNode).
* The [NameNode](http://wiki.apache.org/hadoop/NameNode) should start replaying the edits file, updating the image, block reports should come in, etc.
* At this point, your [NameNode](http://wiki.apache.org/hadoop/NameNode) should be up.

## Manual NameNode Metadata Recovery

When properly configured, HDFS is much more robust against metadata corruption than a local filesystem, because it stores multiple copies of everything. However, because HDFS is a truly robust system, we added the capability for an administrator to recover a partial or corrupted edit log. This new functionality is called manual NameNode recovery.

NameNode recovery is an offline process. An administrator can run NameNode recovery to recover a corrupted edit log. This can be very helpful for getting corrupted filesystems on their feet again.

## NameNode Recovery in Action

Let’s test out recovery mode. To activate recovery mode, you start the NameNode with the -recover flag, like so:

./bin/hadoop namenode -recover

At this point, the NameNode will ask you whether you want to continue.

You have selected Metadata Recovery mode. This mode is intended to recover

lost metadata on a corrupt filesystem. Metadata recovery mode often

permanently deletes data from your HDFS filesystem. Please back up your edit

log and fsimage before trying this!

Are you ready to proceed? (Y/N)

(Y or N)

Once you answer yes, the recovery process will read as much of the edit log as possible.

# DataNodeFailOver

We have replication in place to account for cases when a datanode is notreachable. The namenode (master) starts replicating files that were on that data node. You can also tell the dfs to maintain more copies (than usual) of certain important files using a replication factor.