## Directory

# 1 Introduction

## 1.1 What is the Hadoop

Hadoop is a distributed system infrastructure developed by the Apache Foundation. Users can develop distributed programs without knowing the underlying details of distribution. Take full advantage of the power of clustering for high-speed computing and storage. Hadoop implements a Distributed File System, one of these components is HDFS (Hadoop Distributed File System). HDFS is fault-tolerant and designed to be deployed on low-cost hardware. Moreover, it provides high throughput to access the application data, which is suitable for applications with large data sets. HDFS relaxes POSIX requirements and allows streaming access to data in the file system. The core design of the Hadoop framework is HDFS and MapReduce. HDFS provides storage for massive amounts of data, while MapReduce provides computing for massive amounts of data [1]. Hadoop is mainly used to solve the storage and analysis and calculation problems of massive data.

Sometimes, Hadoop also refers to a broader concept, that is the Hadoop ecosystem.

## 1.2 Features

### 1.2.1Advantages

Hadoop is a software framework that makes it easy for users to structure and distribute large amounts of data. Users can easily develop and run applications that handle large amounts of data on Hadoop. It mainly has the following advantages [2]:

1. High reliability. Hadoop can be trusted with its ability to store and process data bit by bit.
2. High scalability. Hadoop distributes data and performs computing tasks among available clusters of computers that can be easily scaled up to thousands of nodes.
3. High efficiency. Hadoop's ability to dynamically move data between nodes and ensure dynamic balance between nodes makes it very fast to process.
4. High fault tolerance. Hadoop can automatically save multiple copies of data and can automatically reassign failed tasks.
5. Low cost. Compared with all-in-one machines, commercial data warehouses and data marts such as Qlikview and Yonghong Z-Suite, Hadoop is open source, so the software cost of the project will be greatly reduced. And it relies on community services that anyone can use.

### 1.2.2 Application

One of the most common uses of Hadoop is Web search. While it is not the only software framework application, it does a very good job as a parallel data processing engine. One of the most interesting aspects of Hadoop is the Map and Reduce process [3], which was inspired by Google's development. This process, called indexing, takes as input the text Web pages retrieved by the Web spider and reports as a result the frequency of the words on those pages. This result can then be used throughout the Web search process to identify the content from the defined search parameters.

The simplest MapReduce application has at least three parts: A Map function, a Reduce function, and a main function. The main function combines job control with file input/output. At this point, Hadoop provides a large number of interfaces and abstract classes that provide Hadoop application developers with a number of tools for debugging and performance measurement, among other things.

MapReduce itself is a software framework for parallel processing of large data sets. The roots of MapReduce are the map and reduce functions in functional programming. It consists of two operations that may contain many instances (many maps and reduces). The Map function takes a set of data and converts it into a list of key/value pairs, one for each element in the input field. The Reduce function takes the list generated by the Map function and shrinks the list of key/value pairs based on their keys (one key/value pair for each key).

### 1.2.3 Compare

(1) Hadoop and MPI have different positions in data storage and data processing in the system.

MPI is the separation of computing and storage, and Hadoop is the migration of computing to storage, so in Hadoop system, the location of data storage is more important.

One drawback of MPI is that it does not have an underlying distributed file system to support it. In MPI, the nodes for data storage and data processing are often different. Generally, at the beginning of each calculation, MPI needs to read the data to be processed from the data storage node and distribute it to each computing node, and then carry out data processing. In other words, MPI's data storage and data processing are separated. For computation-intensive applications, MPI can show good performance, but for data-intensive applications dealing with terabyte data, where a large amount of data is exchanged between nodes, network communication time will become an important factor affecting system performance, and the performance will be greatly reduced. Using "computing for communication" is also a basic principle in MPI parallel program design.

In Hadoop, there is HDFS file system support. Data is distributed stored in each node, and each node reads the data stored in its own node for processing during calculation, thus avoiding the transmission of a large amount of data on the network and realizing the "migration from computing to storage". This is a huge advantage for dealing with terabytes of data.

From the perspective of the upper structure, Hadoop is a typical master-slave structure, which is also an important parallel programming method in MPI parallel programming design. The master node is responsible for the management and distribution of the data and work of the whole system. The most fundamental difference between Hadoop and MPI is that Hadoop has a master-slave file system, HDFS, underpinning its Map/Reduce data processing capabilities. With HDFS, Hadoop can easily achieve the "migration of computing to data storage location", thus greatly improving the efficiency of the system computing. Master-slave basic storage and master-slave data processing constitute the basic architectural model of Hadoop.

(2) MPI cannot cope with node failure.

If the MPI occurs node failure and network communication interruption during operation, it can only return and exit. MPI does not provide a mechanism to deal with the problem of backup handling when a node fails, so if there is a problem in the middle of the process, all the computation will start over, which can be very time-consuming. In order to deal with the failure of the server, Hadoop has made great efforts in data backup. Data blocks will form multiple copies and store them in different places, usually with three copies. The simple cross-rack data block storage is adopted to avoid data loss to the greatest extent and the security of data is guaranteed.

(3) MPI vs OpenMP vs GPU vs Hadoop

MPI is an example of messaging parallelism. Here is a root computer that generates programs on all the computers in its MPI World. All threads in the system are independent, so the only way they can communicate with each other is through messages over the network. Network bandwidth and throughput are one of the most critical factors for MPI performance.

If there is only one thread on each machine but many kernels, you can use the OpenMP shared memory paradigm to solve a subset of the problem on a single machine.

CUDA is an example of SMT parallelism. It uses the most advanced GPU architecture to provide Parallelism. The GPU consists of a set of kernels (similar to the SIMD model) working on the same instruction in a locked step size manner. Therefore, if all threads in the system are doing a lot of the same work, you can use CUDA. But there is a limited amount of shared and global memory in the GPU, so you shouldn't use just one GPU to solve a huge problem.

Hadoop is used to solve big problems on commodity hardware using the Map Reduce paradigm. As a result, users do not have to worry about distributing data or managing extreme cases. Hadoop also provides a file system, HDFS, for storing data on compute nodes.

## 1.3 Hadoop components

In Hadoop1.x era, MapReduce in Hadoop deals with business logic operation and resource scheduling at the same time, which is highly coupled. In the age of hadoop2.x, Yarn was added. YARN is only responsible for scheduling resources, and MapReduce is only responsible for computing. Hadoop3.x has no change in composition.

Hadoop is made up of many elements. At the very bottom is the Hadoop Distributed File System (HDFS), which stores files on all storage nodes in the Hadoop cluster. Next to HDFS is the MapReduce engine, which is made up of JobTrackers and TaskTrackers [4].

### 1.3.1 HDFS

To external clients, HDFS is like a traditional hierarchical file system. You can create, delete, move, or rename files, and so on. But the architecture of HDFS is based on a specific set of nodes, which is determined by its own characteristics. These nodes include the NameNode (just one), which provides metadata services inside HDFS; The DataNode, which provides blocks of storage for HDFS. Since only one NameNode exists, this is a disadvantage of the HDFS 1.x version (single point of failure). In version 2.x of Hadoop, two NameNodes can exist, which solves the problem of single node failure.

Files stored in HDFS are divided into blocks, and then these blocks are copied to multiple datanodes. This is quite different from a traditional RAID architecture. The size of the blocks (64MB by default for the 1.x version and 128MB for the 2.x version) and the number of blocks copied are determined by the client at file creation time. The NameNode can control all file operations. All communication within HDFS is based on the standard TCP/IP protocol [5].

### 1.3.2 MapReduce

Hadoop Map/Reduce [6] is an easy-to-use software framework, based on which applications can run on a large cluster consisting of thousands of business machines, and parallel processing of T-level data sets in a reliable fault-tolerant manner.

A Map/Reduce job typically splits the input data set into separate chunks, which are processed in full parallel by the Map task. The framework sorts the output of the map and then inputs the results to the reduce task. Usually, the input and output of the job are stored in the file system. The overall framework is responsible for scheduling and monitoring tasks, as well as reexecuting tasks that have failed.

Typically, Map/Reduce frameworks and distributed file systems run on the same set of nodes, that is, compute nodes and storage nodes are usually together. This configuration allows the framework to efficiently schedule tasks on those nodes where data is already stored, which allows the network bandwidth of the entire cluster to be used very efficiently.

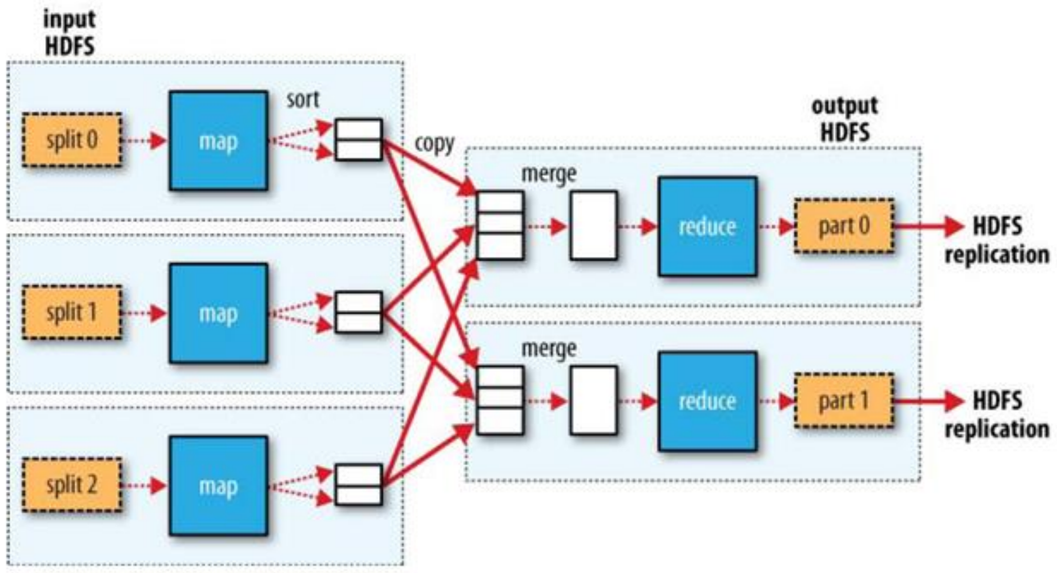
The Map/Reduce framework consists of a single Master JobTracker and a slave TaskTracker for each cluster node. The master is responsible for scheduling all the tasks that constitute a job. These tasks are distributed on different slaves. The master monitors their execution and reexecutes the failed tasks. The slave is only responsible for performing tasks assigned by the master.

At a minimum, the application should specify the location (path) of the input/output and provide the map and reduce functions by implementing the appropriate interfaces or abstract classes. This, combined with other job parameters, forms a Job Configuration. Hadoop's Job Client then submits jobs (JARs/executables, etc.) and configuration information to the JobTracker, which distributes the software and configuration information to the slave, dispatches the tasks and monitors their execution, and provides status and diagnostic information to the Job-Client.

### 1.3.3 Yarn

Apache Hadoop Yarn [7] (Yet Another Resource Negotiator, Another Resource Negotiator) is a new Hadoop Resource manager that is a universal Resource management system that provides unified Resource management and scheduling for upper-level applications. Its introduction brings great benefits for clustering in terms of utilization, unified management of resources and data sharing.

### 1.3.4 The relationship between HDFS and MapReduce[8]



## 1.4How Hadoop Works [9]

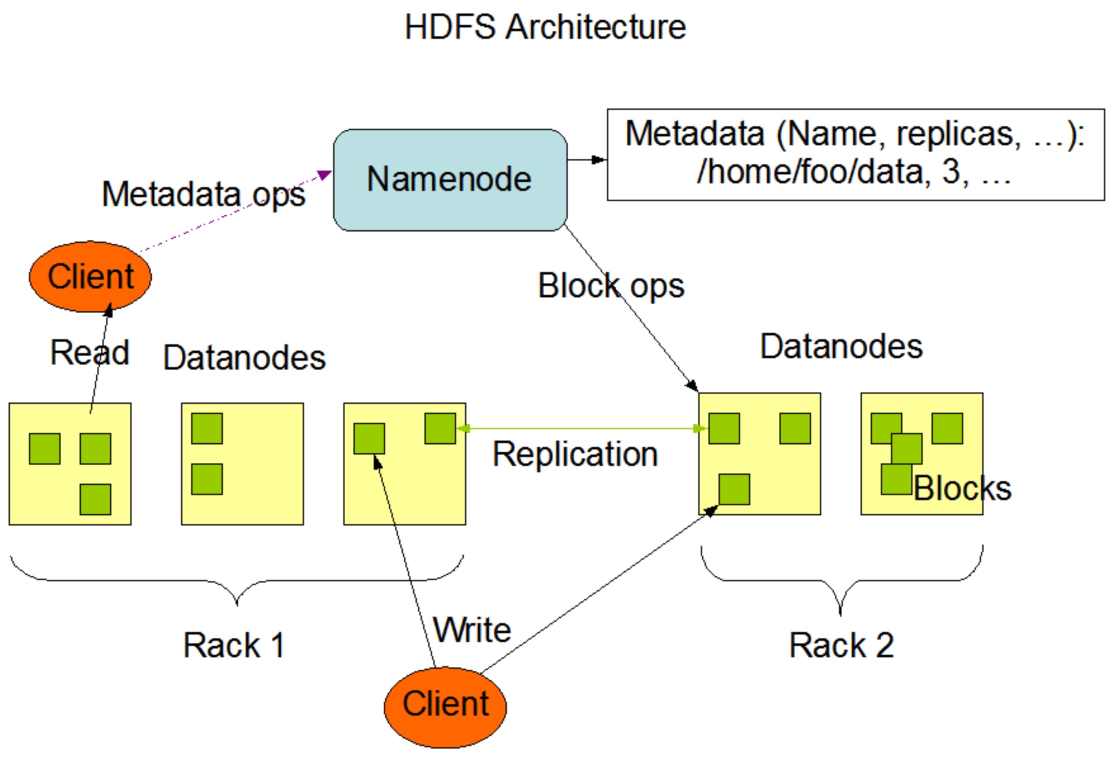
The Apache Hadoop software library is a framework that allows for the distributed processing of large data sets across clusters of computers using simple programming models. It is designed to scale up from single servers to thousands of machines, each offering local computation and storage.

Rather than rely on hardware to deliver high-availability, the library itself is designed to detect and handle failures at the application layer, so delivering a highly-available service on top of a cluster of computers, each of which may be prone to failures. The

Hadoop library contains two major components HDFS and MapReduce, in this post we will go inside each HDFS part and discover how it works internally.

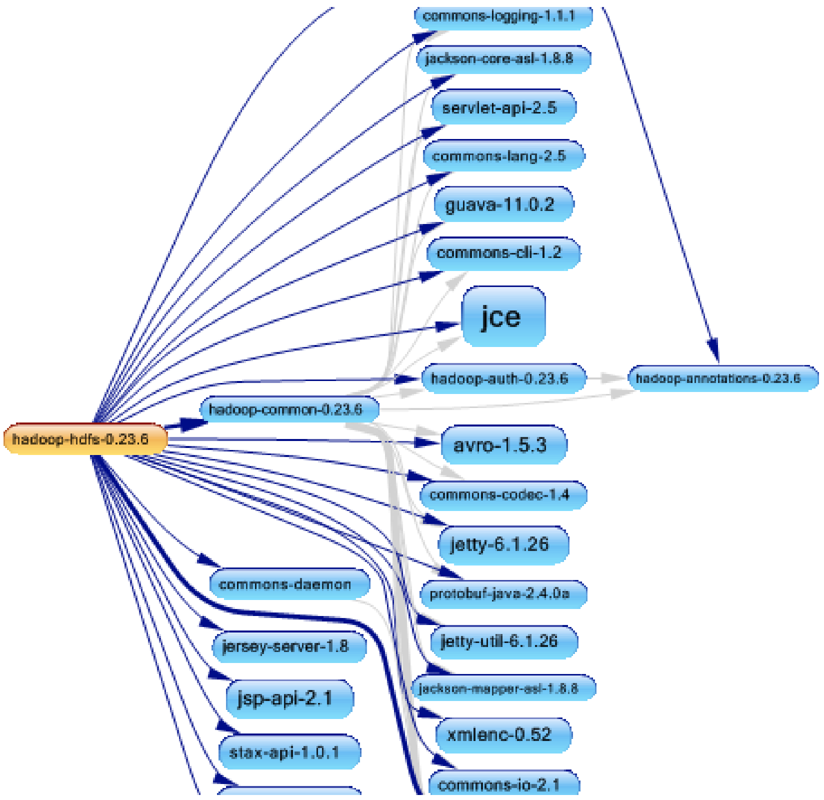
HDFS has a master/slave architecture. An HDFS cluster consists of a single NameNode, a master server that manages the file system namespace and regulates access to files by clients. In addition, there are a number of DataNodes, usually one per node in the cluster, which manage storage attached to the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

HDFS exposes a file system namespace and allows user data to be stored in files. Internally, a file is split into one or more blocks and these blocks are stored in a set of DataNodes. The NameNode executes file system namespace operations like opening, closing, and renaming files and directories. It also determines the mapping of blocks to DataNodes. The DataNodes are responsible for serving read and write requests from the file system's clients. The DataNodes also perform block creation, deletion, and replication upon instruction from the NameNode.

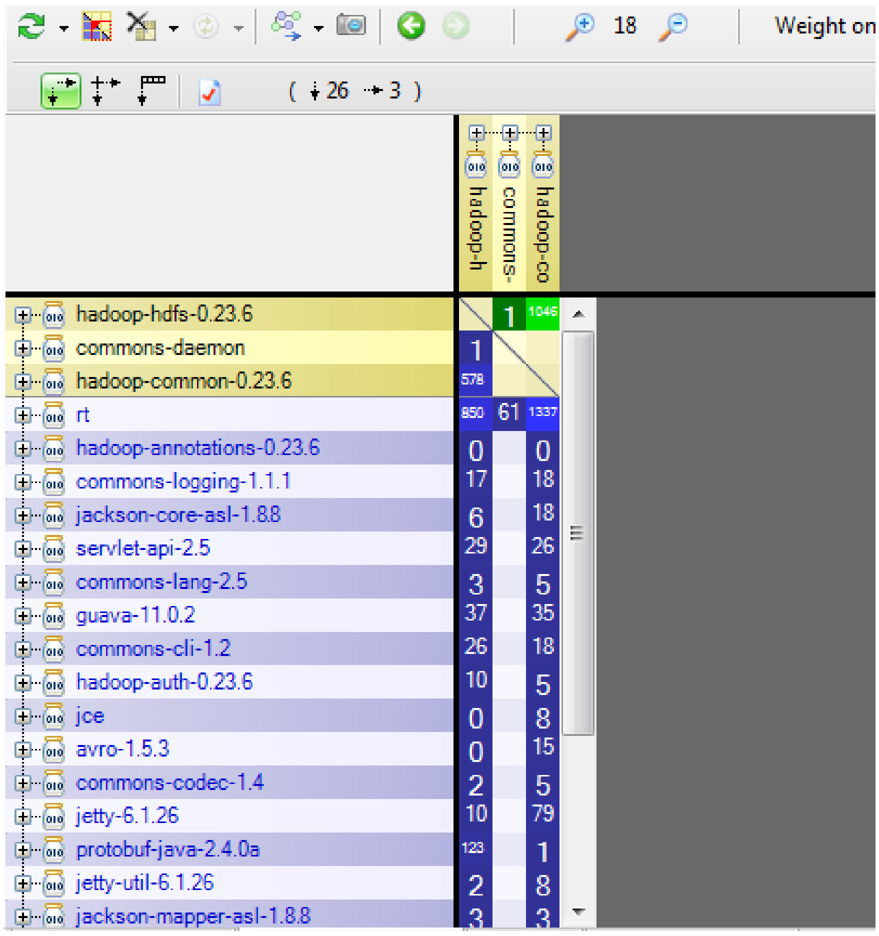


## 1.4.1 HDFS analysis

After the analysis of the Hadoop with JArchitect, here's the dependency graph of the HDFS project.



To achieve its job, HDFS uses many third-party libs like guava, jetty, jackson and others. The DSM (Design Structure Matrix) give us more info about the weight of using each lib.

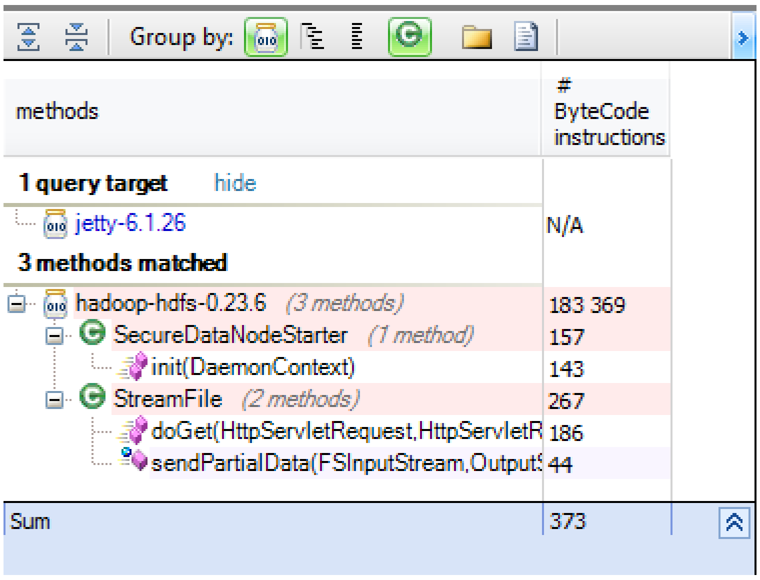


| | hadoop-h | commons- | hadoop-co |
|---|---|---|---|
| hadoop-hdfs-0.23.6 | | 1 | 1046 |
| commons-daemon | 1 | | |
| hadoop-common-0.23.6 | 578 | | |
| rt | 850 | 61 | 1337 |
| hadoop-annotations-0.23.6 | 0 | | 0 |
| commons-logging-1.1.1 | 17 | | 18 |
| jackson-core-asl-1.8.8 | 6 | | 18 |
| servlet-api-2.5 | 29 | | 26 |
| commons-lang-2.5 | 3 | | 5 |
| guava-11.0.2 | 37 | | 35 |
| commons-cli-1.2 | 26 | | 18 |
| hadoop-auth-0.23.6 | 10 | | 5 |
| jce | 0 | | 8 |
| avro-1.5.3 | 0 | | 15 |
| commons-codec-1.4 | 2 | | 5 |
| jetty-6.1.26 | 10 | | 79 |
| protobuf-java-2.4.0a | 123 | | 1 |
| jetty-util-6.1.26 | 2 | | 8 |
| jackson-mapper-asl-1.8.8 | 3 | | 3 |

HDFS use mostly rt, Hadoop-common and protobuf libraries. When external libs are used, it's better to check if we can easily change a third-party lib by another one without impacting the whole application, there are many reasons that can encourage us to change a third-party lib. The other lib could:

- Have more features
- More performance
- More secure

Let's take the example of jetty lib and search which methods from HDFS use it directly. from m in Methods where m.IsUsing ("jetty-6.1.26" ) && m.ParentProject.Name=="hadoop-hdfs-0.23.6" select new {m, m. NbBCInstructions}

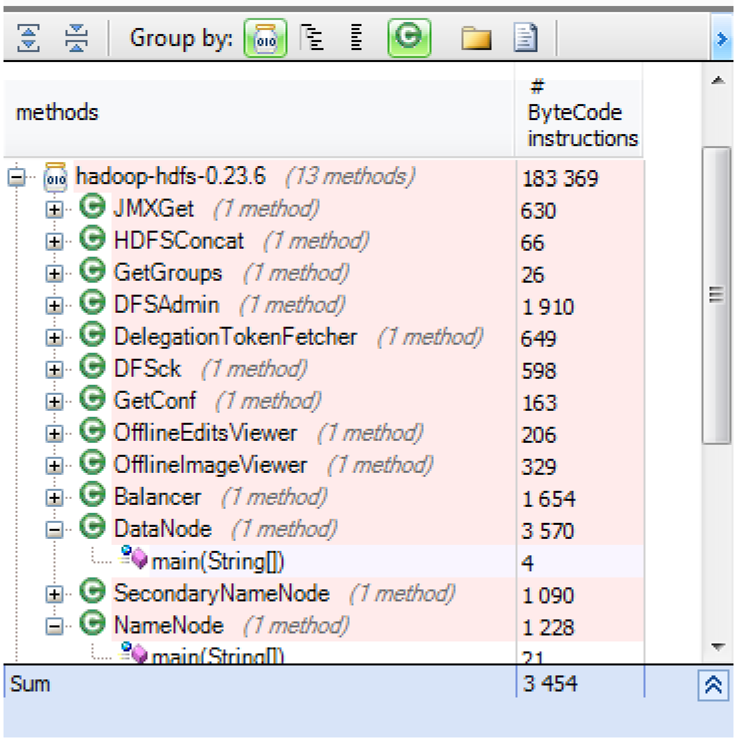


| methods | # ByteCode instructions |
|---|---|
| **1 query target** hide | |
| jetty-6.1.26 | N/A |
| **3 methods matched** | |
| hadoop-hdfs-0.23.6 *(3 methods)* | 183 369 |
| SecureDataNodeStarter *(1 method)* | 157 |
| init(DaemonContext) | 143 |
| StreamFile *(2 methods)* | 267 |
| doGet(HttpServletRequest,HttpServletR | 186 |
| sendPartialData(FSInputStream,Output! | 44 |
| Sum | 373 |

Only few methods use directly jetty lib, and changing it with another one will be very easy. In general, it's very interesting to isolate when you can the using of an external lib in only some classes, it can help to maintain and evolve the project easily. Let's discover now the major HDFS components:

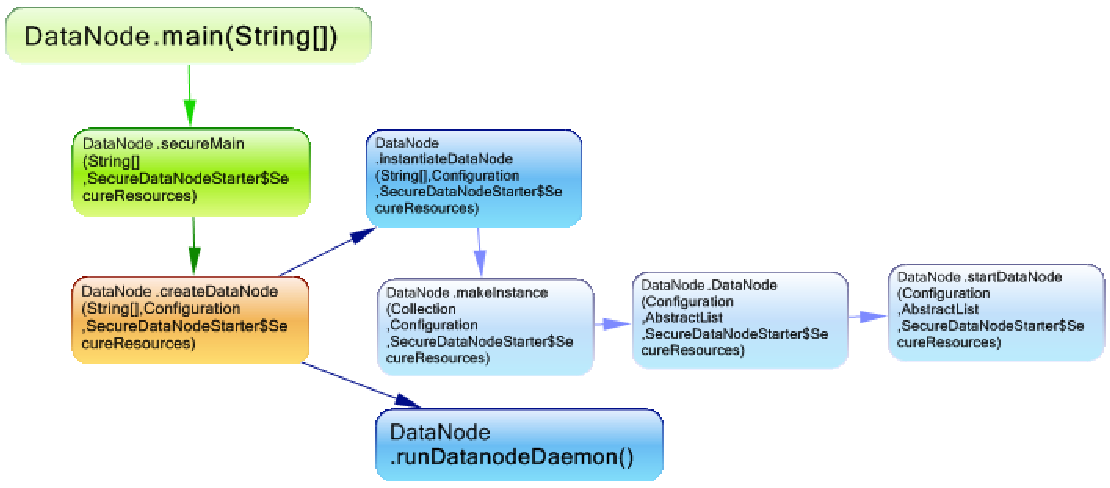
## 1.4.2 DataNode

To discover how to launch a data node, let's search before all entry points of the HDFS jar. from m in Methods where m.Name.Contains("main(String[])") && m.IsStatic select new {m, m.NbBCInstructions}



| methods | # ByteCode instructions |
|---|---|
| hadoop-hdfs-0.23.6 *(13 methods)* | 183 369 |
| JMXGet *(1 method)* | 630 |
| HDFSConcat *(1 method)* | 66 |
| GetGroups *(1 method)* | 26 |
| DFSAdmin *(1 method)* | 1 910 |
| DelegationTokenFetcher *(1 method)* | 649 |
| DFSck *(1 method)* | 598 |
| GetConf *(1 method)* | 163 |
| OfflineEditsViewer *(1 method)* | 206 |
| OfflineImageViewer *(1 method)* | 329 |
| Balancer *(1 method)* | 1 654 |
| DataNode *(1 method)* | 3 570 |
| main(String[]) | 4 |
| SecondaryNameNode *(1 method)* | 1 090 |
| NameNode *(1 method)* | 1 228 |
| main(String[]) | 21 |
| Sum | 3 454 |

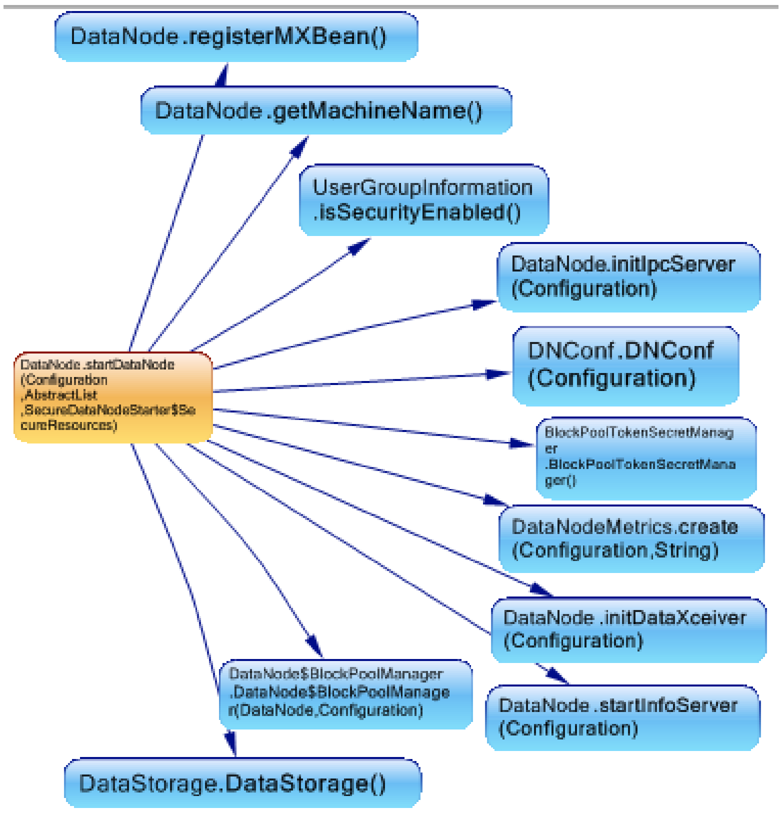
HDFS has many entries like DFSAdmin, DfSsc, Balancer and HDFSConcat. For the data node the entry point concerned is the DataNode class, and here's what happen when its main method is invoked.


DataNode.main(String[])
DataNode .secureMain (String[] ,SecureDataNodeStarter$SecureResources)
DataNode .createDataNode (String[],Configuration ,SecureDataNodeStarter$SecureResources)
DataNode .instantiateDataNode (String[],Configuration ,SecureDataNodeStarter$SecureResources)
DataNode .makeInstance (Collection ,Configuration ,SecureDataNodeStarter$SecureResources)
DataNode .DataNode (Configuration ,AbstractList ,SecureDataNodeStarter$SecureResources)
DataNode .startDataNode (Configuration ,AbstractList ,SecureDataNodeStarter$SecureResources)
DataNode .runDatanodeDaemon()

The main method invokes first secureMain and pass the param securityResources to it, when the node is started in a not secure cluster this param is null, however in the case of starting it in a secure environment, the param is assigned with the secure resources. The SecureResources class contains two attributes:

1. streamingSocket: secure port for data streaming to datanode.
2. listner: a secure listener for the web server.

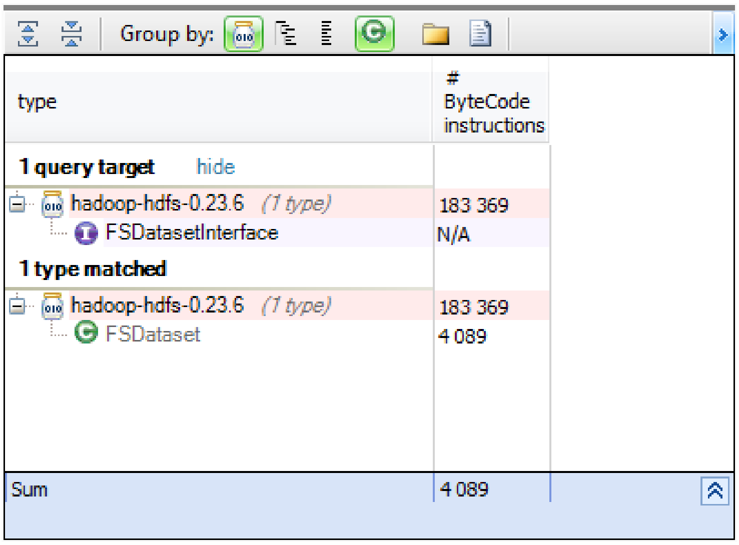
And here are the methods invoked from DataNode.StartDataNode.


DataNode.registerMXBean()
DataNode.getMachineName()
UserGroupInformation .isSecurityEnabled()
DataNode.initIpcServer (Configuration)
DNConf.DNConf (Configuration)
DataNode.startDataNode (Configuration ,AbstractList ,SecureDataNodeStarter$SecureResources)
BlockPoolTokenSecretManager .BlockPoolTokenSecretManager()
DataNodeMetrics.create (Configuration,String)
DataNode.initDataXceiver (Configuration)
DataNode$BlockPoolManager .DataNode$BlockPoolManager(DataNode,Configuration)
DataNode.startInfoServer (Configuration)
DataStorage.DataStorage()

This method initialize IPCServer,DataXceiver which is the thread for processing incoming/outgoing data stream, create data node metrics instance.

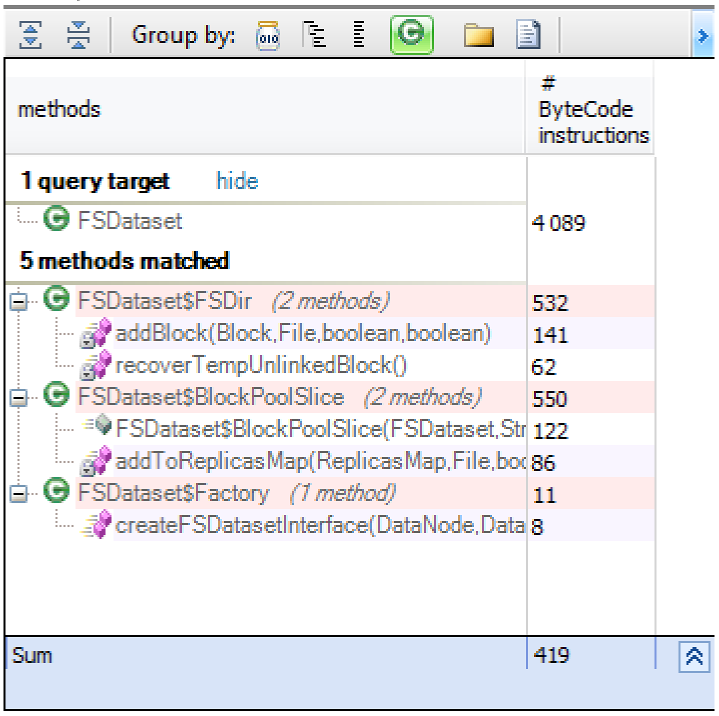
### 1.4.3 How data is managed?

The DataNode class has an attribute named data of type FSDatasetinterface. FSDatasetinterface is an interface for the underlying storage that stores blocks for a data node. Let's search which implementations are available in Hadoop. from t in Types where t.Implement ("org.apache.hadoop.hdfs.server.datanode.FSDatasetInterface") select new {t, t.NbBCInstructions}



| type | # ByteCode instructions |
|---|---|
| **1 query target** hide | |
| hadoop-hdfs-0.23.6 *(1 type)* | 183 369 |
| FSDatasetInterface | N/A |
| **1 type matched** | |
| hadoop-hdfs-0.23.6 *(1 type)* | 183 369 |
| FSDataset | 4 089 |
| Sum | 4 089 |

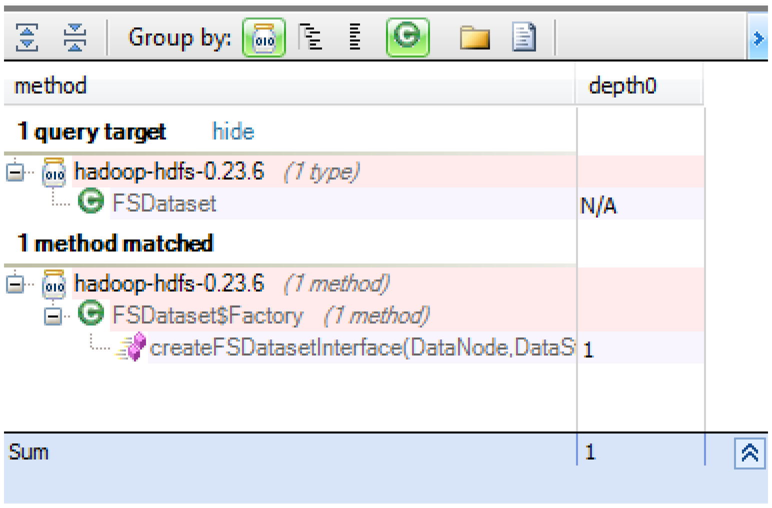
Hadoop provides FSDataset which manages a set of data blocks and store them on dirs. Using interfaces enforce low coupling and makes the design very flexible, however if the implementation is used instead of the interface we lose this advantage, and to check if interfaceDataSet is used anywhere to represent the data, let's search for all methods using FSDataSet. from m in Methods where m.IsUsing ("org.apache.hadoop.hdfs.server.datanode.FSDataset") select new {m, m.NbBCInstructions}



Group by:

| methods | # ByteCode instructions |
|---|---|
| **1 query target** hide | |
| FSDataset | 4 089 |
| **5 methods matched** | |
| FSDataset$FSDir *(2 methods)* | 532 |
| addBlock(Block,File,boolean,boolean) | 141 |
| recoverTempUnlinkedBlock() | 62 |
| FSDataset$BlockPoolSlice *(2 methods)* | 550 |
| FSDataset$BlockPoolSlice(FSDataset,Str | 122 |
| addToReplicasMap(ReplicasMap,File,boo | 86 |
| FSDataset$Factory *(1 method)* | 11 |
| createFSDatasetInterface(DataNode,Data | 8 |
| Sum | 419 |

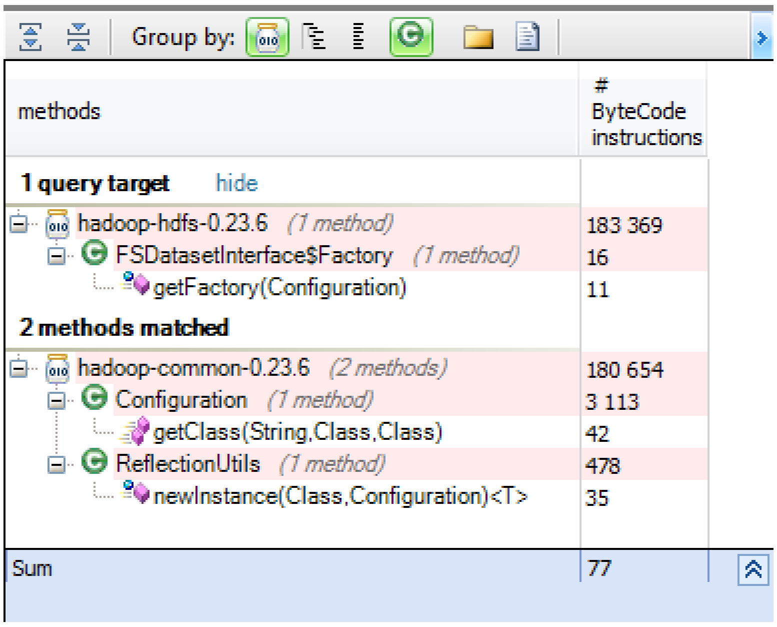
Only FSDataSet inner classes use it directly, and for all the other places the interfaceDataSet is used instead, what makes the possibility to change the dataset kind very easy. But how can I change the interfaceDataSet and give my own implementation? For that let's search where the FSDataSet is created.
from m in Methods let depth0 = m.DepthOfCreateA ("org.apache.hadoop.hdfs.server.datanode.FSDataset") where depth0 == 1 select new {m, depth0}



Group by:

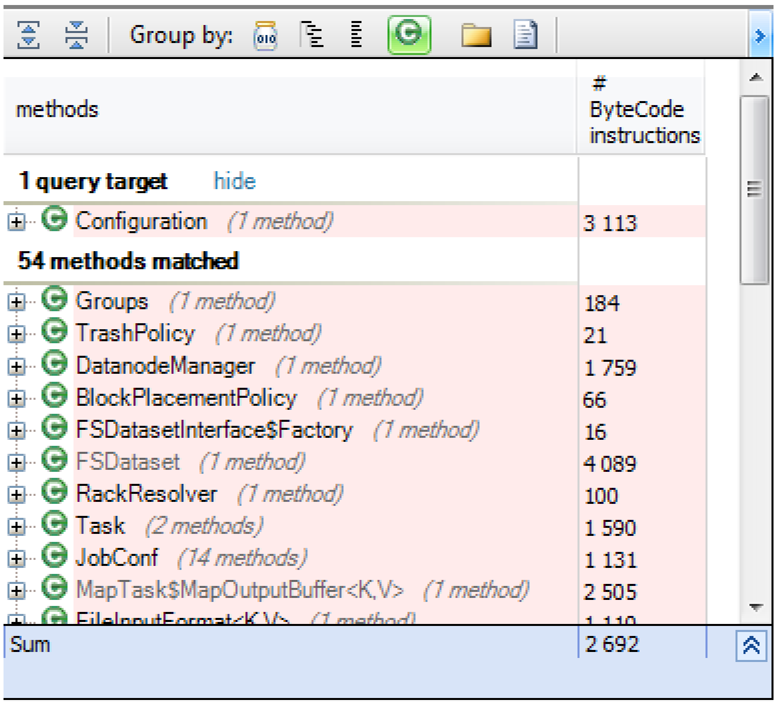
| method | depth0 |
|---|---|
| **1 query target** hide | |
| hadoop-hdfs-0.23.6 *(1 type)* | |
| FSDataset | N/A |
| **1 method matched** | |
| hadoop-hdfs-0.23.6 *(1 method)* | |
| FSDataset$Factory *(1 method)* | |
| createFSDatasetInterface(DataNode,DataS | 1 |
| Sum | 1 |

The factory pattern is used to create the instance; the problem is if this factory creates the implementation directly inside getFactory method, we have to change the Hadoop code to give it our custom DataSet manager. Let's discover which methods are used by the getFactory method. from m in Methods where m.IsUsedBy ("org.apache.hadoop.hdfs.server.datanode.FSDatasetInterface $Factory.getFactory(Configuration)") select new {m, m.NbBCInstructions}



| methods | # ByteCode instructions |
|---|---|
| **1 query target** hide | |
| hadoop-hdfs-0.23.6 *(1 method)* | 183 369 |
| FSDatasetInterface$Factory *(1 method)* | 16 |
| getFactory(Configuration) | 11 |
| **2 methods matched** | |
| hadoop-common-0.23.6 *(2 methods)* | 180 654 |
| Configuration *(1 method)* | 3 113 |
| getClass(String,Class,Class) | 42 |
| ReflectionUtils *(1 method)* | 478 |
| newInstance(Class,Configuration)<T> | 35 |
| Sum | 77 |

The good news is that the factory uses the Configuration to get the class implementation, so we can only by configuration gives our custom DataSet, we can also search for all classes that can be given by configuration.
from m in Methods where m.IsUsing ("org.apache.hadoop.conf.Configuration.getClass(String,Class,Class)") select new {m, m.NbBCInstructions}

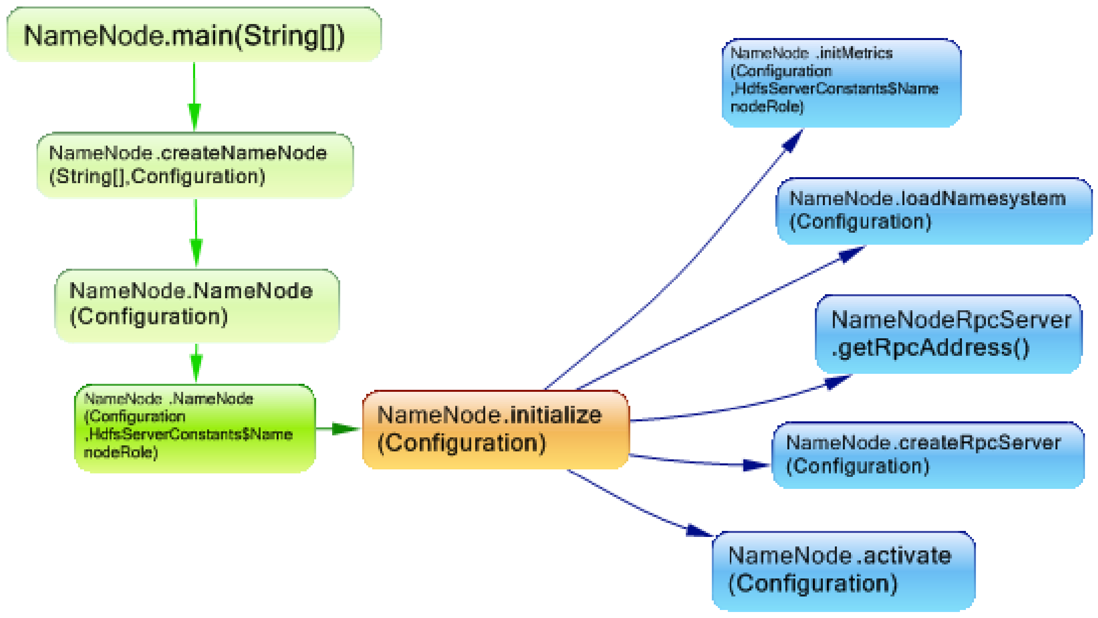


| methods | # ByteCode instructions |
|---|---|
| **1 query target** hide | |
| Configuration *(1 method)* | 3 113 |
| **54 methods matched** | |
| Groups *(1 method)* | 184 |
| TrashPolicy *(1 method)* | 21 |
| DatanodeManager *(1 method)* | 1 759 |
| BlockPlacementPolicy *(1 method)* | 66 |
| FSDatasetInterface$Factory *(1 method)* | 16 |
| FSDataset *(1 method)* | 4 089 |
| RackResolver *(1 method)* | 100 |
| Task *(2 methods)* | 1 590 |
| JobConf *(14 methods)* | 1 131 |
| MapTask$MapOutputBuffer<K,V> *(1 method)* | 2 505 |
| FileInputFormat<K,V> *(1 method)* | 1 110 |
| Sum | 2 692 |

Many classes could be injected inside the Hadoop framework without changing its source code, what makes it very flexible.

### 1.4.4 NameNode

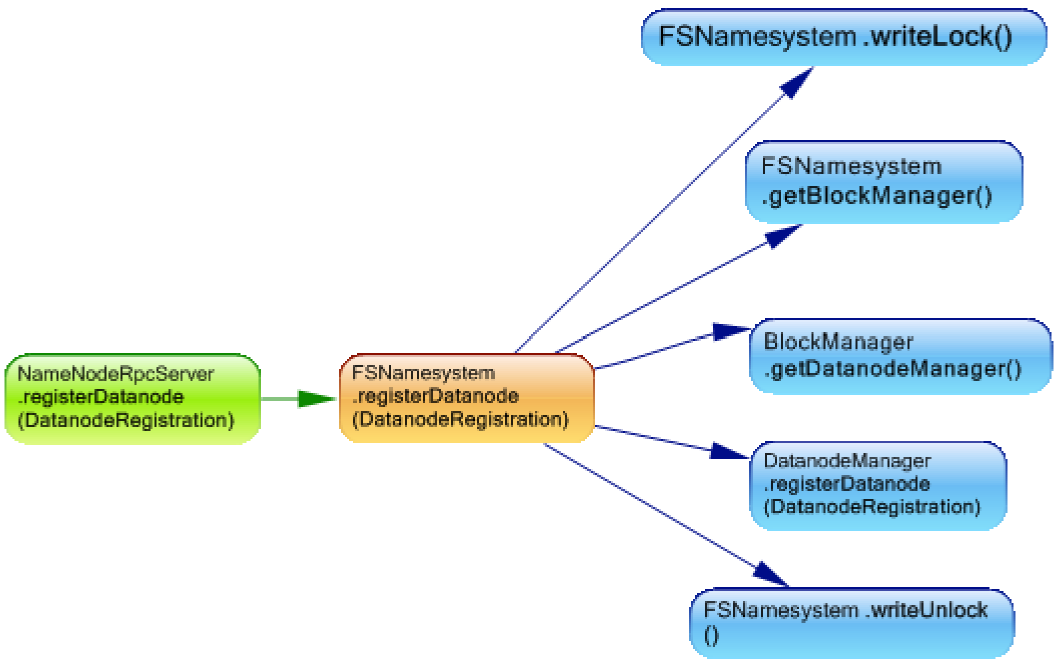
The NameNode is the arbitrator and repository for all HDFS metadata. The system is designed in such a way that user data never flows through the NameNode. Here are some methods invoked when the name node is launched.


NameNode.main(String[])
NameNode.createNameNode (String[],Configuration)
NameNode.NameNode (Configuration)
NameNode .NameNode (Configuration ,HdfsServerConstants$NamenodeRole)
NameNode.initialize (Configuration)
NameNode .initMetrics (Configuration ,HdfsServerConstants$NamenodeRole)
NameNode.loadNamesystem (Configuration)
NameNodeRpcServer .getRpcAddress()
NameNode.createRpcServer (Configuration)
NameNode.activate (Configuration)

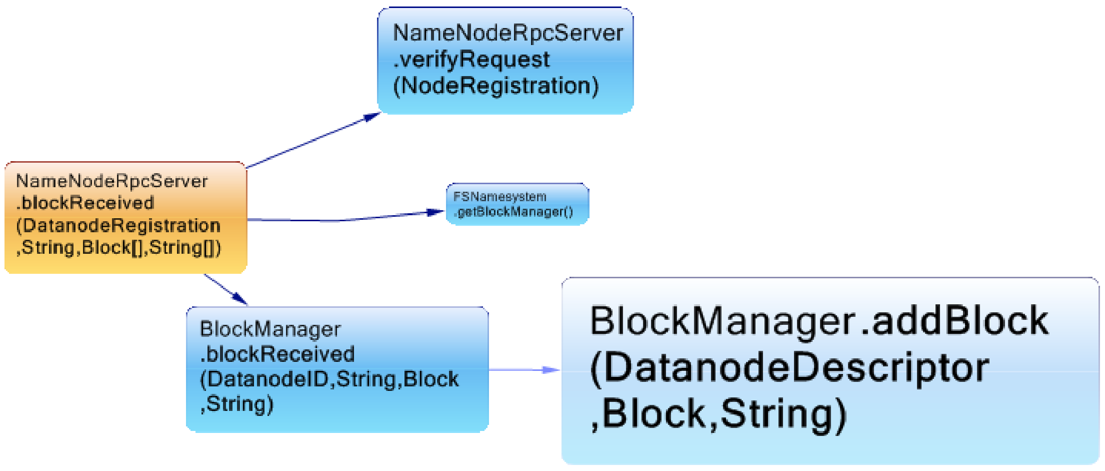
The RPC Server is created, and the fsnamesystem is loaded, here's a quick look to these two components:

### 1.4.5 NameNodeRpcServer

NameNodeRpcServer is responsible for handling all of the RPC calls to the NameNode. For example, when a data node is launched, it must register itself with the NameNode, the rpc server receive this request and forward it to fsnamesystem, which redirect it to dataNodeManager.


FSNamesystem .writeLock()
FSNamesystem .getBlockManager()
NameNodeRpcServer .registerDatanode (DatanodeRegistration)
FSNamesystem .registerDatanode (DatanodeRegistration)
BlockManager .getDatanodeManager()
DatanodeManager .registerDatanode (DatanodeRegistration)
FSNamesystem .writeUnlock ()

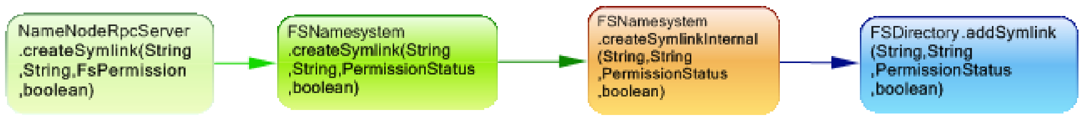
Another example is when a block of data is received. from m in Methods where m.IsUsedBy ("org.apache.hadoop.hdfs.server.namenode.NameNodeRpcServer.blockReceived( DatanodeRegistration,String,Block[],String[])") select new {m, m.NbBCInstructions}


NameNodeRpcServer .verifyRequest (NodeRegistration)
NameNodeRpcServer .blockReceived (DatanodeRegistration ,String,Block[],String[])
FSNamesystem .getBlockManager()
BlockManager .blockReceived (DatanodeID,String,Block ,String)
BlockManager.addBlock (DatanodeDescriptor ,Block,String)

Each rectangle in the graph is proportional to the number of bytes of code instructions, and we can observe the BlockManager.addBlock do the most of the job. What's interesting with Hadoop is that each class has a specific responsibility, and any request is redirected to the corresponding manager.

### 1.4.6 FSnamesystem

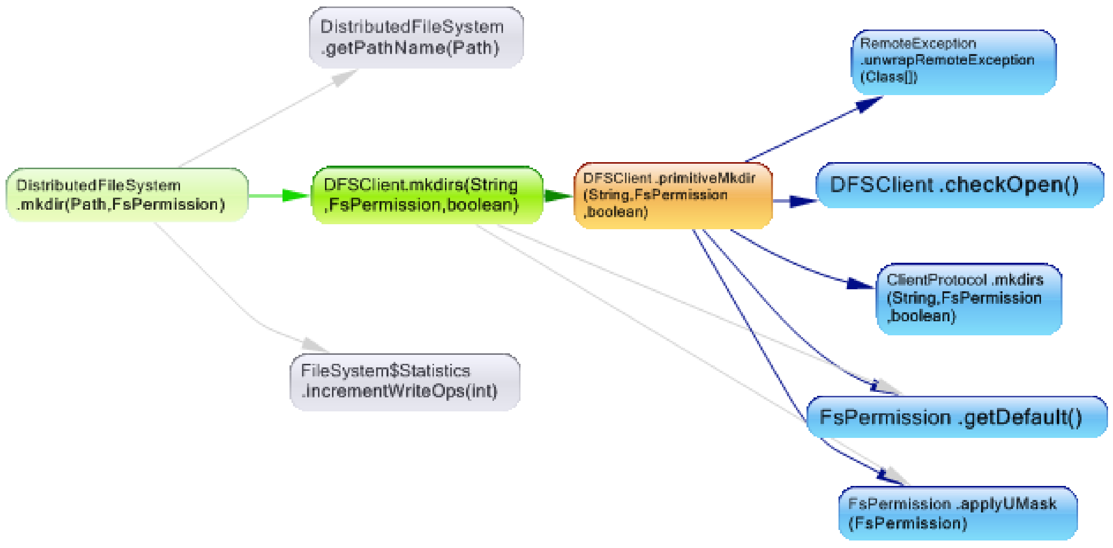
HDFS supports a traditional hierarchical file organization. A user or an application can create directories and store files inside these directories. The file system namespace hierarchy is similar to most other existing file systems; one can create and remove files, move a file from one directory to another, or rename a file. For example, here's a dependency graph concerning the creation of a symbolic link.


NameNodeRpcServer .createSymlink(String ,String,FsPermission ,boolean)
FSNamesystem .createSymlink(String ,String,PermissionStatus ,boolean)
FSNamesystem .createSymlinkInternal (String,String ,PermissionStatus ,boolean)
FSDirectory.addSymlink (String,String ,PermissionStatus ,boolean)

## 1.4.7 HDFS Client

DFSClient can connect to a Hadoop Filesystem and perform basic file tasks. It uses the ClientProtocol to communicate with a NameNode daemon, and connects directly to DataNodes to read/write block data.

Hadoop DFS users should obtain an instance of DistributedFileSystem, which uses DFSClient to handle filesystem tasks. DistributedFileSystem act as facade and redirect requests to the DFSClient class, here's the dependency graph concerning the creation of a directory request.

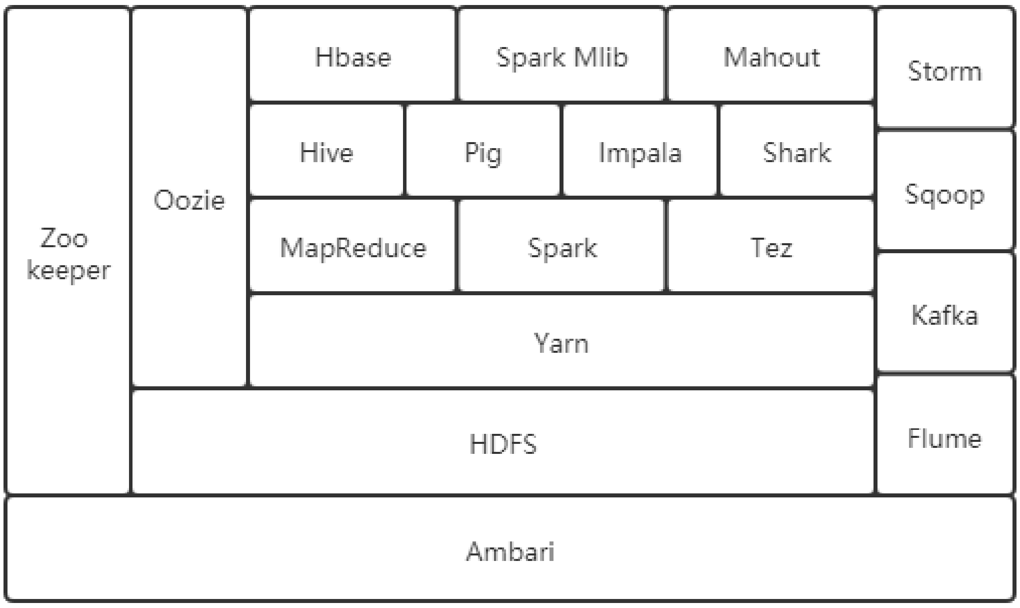

DistributedFileSystem .getPathName(Path)
RemoteException .unwrapRemoteException (Class[])
DistributedFileSystem .mkdir(Path,FsPermission)
DFSClient.mkdirs(String ,FsPermission,boolean)
DFSClient .primitiveMkdir (String,FsPermission ,boolean)
DFSClient .checkOpen()
ClientProtocol .mkdirs (String,FsPermission ,boolean)
FileSystem$Statistics .incrementWriteOps(int)
FsPermission .getDefault()
FsPermission .applyUMask (FsPermission)

## 1.4.8 Conclusion

Using frameworks as user is very interesting, but going inside this framework could give us more info suitable to understand it better, and adapt it to our needs easily. Hadoop is a powerful framework used by many companies, and most of them need to customize it, fortunately Hadoop is very flexible and permit us to change the behavior without changing the source code.

## 1.5 Ecosystem of related tools

In a broad sense, Hadoop refers to an ecosystem, which generally refers to open-source components or products related to big data technology, such as HBase, Hive, Spark, Pig, ZooKeeper, Kafka, Flume, Phoenix, SQOOP, etc. These components or products in the ecosystem may be dependent on each other, but they are independent. HBase and Kafka rely on ZooKeeper, and Hive relies on MapReduce. And the Hadoop ecosystem technology is constantly developing, there will be new components constantly, some old components may be replaced by new components [10].



（1） Hive. Hive is a Hadoop-based data warehouse, open source by Facebook, originally designed to solve the problem of massive structured log data statistics.

（2） The Spark. Spark is a distributed and parallel computing framework based on memory. Unlike MapReduce, the intermediate output of Job can be stored in memory, so there is no need to read and write HDFS. Therefore, Spark is better suited for iterative MapReduce algorithms such as data mining and machine learning.

（3） Flink. Flink is a memory-based distributed parallel processing framework similar to Spark, but with some major design differences.

（4） ZooKeeper. Solve data management problems in distributed environment: unified naming, state synchronization, cluster management, configuration synchronization, etc. Many components of Hadoop rely on ZooKeeper, which runs on a cluster of computers to manage Hadoop operations.

（5） Hive. Hive defines an SQL-like query language (HQL) that converts SQL into MapReduce tasks that are executed on Hadoop.

（6） Impala. Impala is an MPP (massively parallel processing) SQL query engine for processing large amounts of data stored in a Hadoop cluster.

（7） HBase. HBase is a column-oriented, scalable, highly reliable, high-performance, distributed, and column-oriented dynamic schema database built on top of HDFS for structured data.
（8） The Flume. Flume is a mass log collection system that is scalable and suitable for complex environments.
（9） Kafka. Kafka is a high-throughput distributed publish-and-subscribe messaging system that can handle all action streams of data in consumer-scale websites. Subject, partition and its queue patterns, and producer and consumer architecture patterns are implemented.
（10） Oozie. Oozie is an extensible workspace integrated into the Hadoop stack to coordinate the execution of multiple MapReduce jobs. It can manage a complex system and execute based on external events, including the timing and occurrence of data [11].

[1] Jiye Xu, Jiehua Zhu, Haibin Wang, eds., Meteorological Big Data, Shanghai Science and Technology Press,2018.09, p. 46
[2] Wenbi Rao, Ed., Hadoop Core Technology and Experiment, Wuhan University Press,2017.04, page 1
[3] Xu Yang, Haijing Tang, Gangyi Ding, Introduction to Data Science, 2nd edition, Beijing Institute of Technology Press,2017.01, page 168
[4] Xiaohua Li, Yi Zhou, Ed.; Liu Xiaohui, deputy Ed., Technology and Application of Database of Hospital Information Sun Yat-sen University Press,2015.10, pp. 431 ~ 433
[5] Ting Qin, Changhua Zhang, Ed.; Bojue Wang, Hailong Shen, Shanguo Ji, deputy Ed., Cloud Computing Technology Project Tutorial, Intellectual Property Publishing House,2016.12, page 124
[6] https://hadoop.apache.org/docs/r1.0.4/cn/mapred_tutorial.html
[7] https://baike.baidu.com/item/yarn
[8] Graphic source network.
[9] https://www.javacodegeeks.com/2013/04/how-hadoop-works-hdfs-case-study.html
[10] https://www.jianshu.com/p/9918fb395d1e
[11] https://www.cnblogs.com/KdeS/p/13563300.html

# 2 Installation

## 2.1 local mode

### 2.1.1Configuration environment

1. Install the virtual machine, IP address 192.168.10.100, host name hadoop0, memory 4G, hard disk 50G.
2. hadoop0 virtual machine configuration requirements are as follows (all Linux systems

in this paper take Centos-7.5-x86-1804 as an example)

（1）Test the virtual machine networking situation, because the use of yum installation requires the virtual machine can be normally connected to the Internet.

```
# ping www.google.com
```

（2）Install the epel-release

```
# yum install -y epel-release
```

（3）If you have a minimal Linux installation, you also need to install the following tools; If you install the Linux desktop standard edition, do not need to perform the following operations.

```
# yum install -y net-tools
# yum install -y vim
```

3. Close the firewall, close the firewall boot bootup

```
# systemctl stop firewalld
# systemctl disable firewalld.service
```

4. Create a user named hadoop and change the password for the hadoop user

```
# useradd hadoop
# passwd password
```

5. The user is configured to have root permissions, so that it is convenient to add sudo to execute commands with root permissions later

```
# vim /etc/sudoers
```

Modify the /etc/sudoers file and add a line under %wheel:

```
%wheel        ALL=(ALL)     ALL
hadoop  ALL=(ALL)    NOPASSWD:ALL
```

6. Create a folder in the /opt directory and change the owner and group

（1）

```
# mkdir /opt/module
# mkdir /opt/software
```

（2）Modify the owner and group of the module and software folder to be hadoop users

```
# chown hadoop: hadoop/opt/module
# chown hadoop: hadoop/opt/software
```

（3）Check that the modification was successful

7. Uninstall the JDK that comes with the virtual machine

Note: You do not need to perform this step if your virtual machine is minimally installed.

```
# rpm -qa | grep -i java | xargs -n1 rpm -e –nodeps
```

8. Restart the virtual machine

```
# reboot
```

### 2.1.2 Install the JDK

9. Be sure to uninstall the existing JDK.

10. Download the JDK installation zip file to /opt/software. You can use the Xshell transfer tool or download it directly from the official website. Download link:

https://www.oracle.com/java/technologies/javase/javase-jdk8-downloads.html

11. Unzip the JDK to the /opt/module directory, for example

```
$ tar -zxvf jdk-8u291-linux-x64.tar.gz -C /opt/module/
```

12. Configuration of environment variables

（1）Edit the JDK environment variables

```
$ sudo vim /etc/profile.d/my_env.sh
```

Add the following:

```
#JAVA_HOME
export JAVA_HOME=/opt/module/jdk1.8.0_291
export PATH=$PATH:$JAVA_HOME/bin
```

（2）Save and exit

：wq

（3）Let the new environment variable PATH take effect

```
$ source /etc/profile
```

13. Test that the JDK was installed successfully

```
$ java -version
```

If you see the following results, the Java installation is successful.

```
java version "1.8.0_291"
```

restart.

```
$ sudo reboot
```

### 2.1.3 Install the Hadoop

14. Download the Hadoop installation zip to /opt/software. You can use the Xshell transfer tool or download it directly from the official website. Download link: https://archive.apache.org/dist/hadoop/common/hadoop-3.2.2/

15. Unzip the installation file to /opt/module

```
$ tar -zxvf hadoop-3.2.2.tar.gz -C /opt/module/
```

16. Configure the Hadoop environment variables

（1）Open the /etc/profile.d/my_env.sh file

```
$ sudo vim /etc/profile.d/my_env.sh
```

（2）Add the following to the end of the my_env.sh file:

```
#HADOOP_HOME
export HADOOP_HOME=/opt/module/hadoop-3.2.2
export PATH=$PATH:$HADOOP_HOME/bin
export PATH=$PATH:$HADOOP_HOME/sbin
```

save and exit:

:wq

（3）Let the modified file take effect

```
$ source /etc/profile
```

17. Test for successful installation

```
$ hadoop version
Hadoop 3.2.2
```

18. Restart.

```
$ sudo reboot
```

19. Run WordCount program in local run mode to check that Hadoop was successfully installed. See WordCount Test in Chapter 3 for details.

## 2.2 Fully distributed mode

### 2.2.1 Configure the cluster file

20. Cluster deployment planning

NameNode and SecondaryNameNode should not be installed on the same server. Do not configure a ResourceManager on the same machine as NameNode and SecondaryNameNode. So, the deployment plan for each client is as follows:

| | hadoop0 | hadoop1 | hadoop2 |
|---|---|---|---|
| HDFS | NameNode<br>DataNode | DataNode | SecondaryNameNode<br>DataNode |
| YARN | NodeManager | ResourceManager<br>NodeManager | NodeManager |

21. Modify the cluster configuration file

（1）core-site.xml

```
$ cd $HADOOP_HOME/etc/hadoop
$ vim core-site.xml
```

Content:

```
<?xml version="1.0" encoding="UTF-8"?>
<?xml-stylesheet type="text/xsl" href="configuration.xsl"?>

<configuration>
    <property>
        <name>fs.defaultFS</name>
        <value>hdfs://hadoop102:8020</value>
    </property>

    <property>
        <name>hadoop.tmp.dir</name>
        <value>/opt/module/hadoop-3.2.2/data</value>
    </property>

    <property>
    <name>hadoop.http.staticuser.user</name>
    <value> hadoop</value>
    </property>
</configuration>
```

（2）hdfs-site.xml

```
$ vim hdfs-site.xml
```

Content:
```
<?xml version="1.0" encoding="UTF-8"?>
<?xml-stylesheet type="text/xsl" href="configuration.xsl"?>

<configuration>
    <property>
        <name>dfs.namenode.http-address</name>
        <value>hadoop0:9870</value>
    </property>

    <property>
        <name>dfs.namenode.secondary.http-address</name>
        <value>hadoop2:9868</value>
    </property>
</configuration>
```
（3）yarn-site.xml
```
$ vim yarn-site.xml
```
Content:
```
<?xml version="1.0" encoding="UTF-8"?>
<?xml-stylesheet type="text/xsl" href="configuration.xsl"?>

<configuration>
    <property>
        <name>yarn.nodemanager.aux-services</name>
        <value>mapreduce_shuffle</value>
    </property>

    <property>
        <name>yarn.resourcemanager.hostname</name>
        <value>hadoop1</value>
    </property>

    <property>
        <name>yarn.nodemanager.env-whitelist</name>
        <value>JAVA_HOME,HADOOP_COMMON_HOME,HADOOP_HDFS_H
        OME,HADOOP_CONF_DIR,CLASSPATH_PREPEND_DISTCACHE,HADO
        OP_YARN_HOME,HADOOP_MAPRED_HOME</value>
    </property>
</configuration>
```
（4）mapred-site.xml
```
$ vim mapred-site.xml
```
Content:
```
<?xml version="1.0" encoding="UTF-8"?>
<?xml-stylesheet type="text/xsl" href="configuration.xsl"?>
```

```
<configuration>
    <property>
        <name>mapreduce.framework.name</name>
        <value>yarn</value>
    </property>

    <property>
        <name>yarn.app.mapreduce.am.env</name>
        <value>HADOOP_MAPRED_HOME=$HADOOP_HOME</value>
    </property>

    <property>
        <name>mapreduce.map.env</name>
        <value>HADOOP_MAPRED_HOME=$HADOOP_HOME</value>
    </property>

    <property>
        <name>mapreduce.reduce.env</name>
        <value>HADOOP_MAPRED_HOME=$HADOOP_HOME</value>
    </property>
</configuration>
```

22. Configuration of workers

```
$ vim /opt/module/hadoop-3.2.2/etc/hadoop/workers
```

Add the following to the file:

```
hadoop0
hadoop1
hadoop2
```

Note: No Spaces are allowed at the end of any content added to this file, and no blank lines are allowed in the file.

23. Close the hadoop0 client machine.

## 2.2.2 Clone virtual machines

24. Clone two virtual machines using template machine hadoop0: hadoop1, hadoop2

25. Modify the static IP of two clones. For example, hadoop2:

（1）Modify the static IP of the clone virtual machine

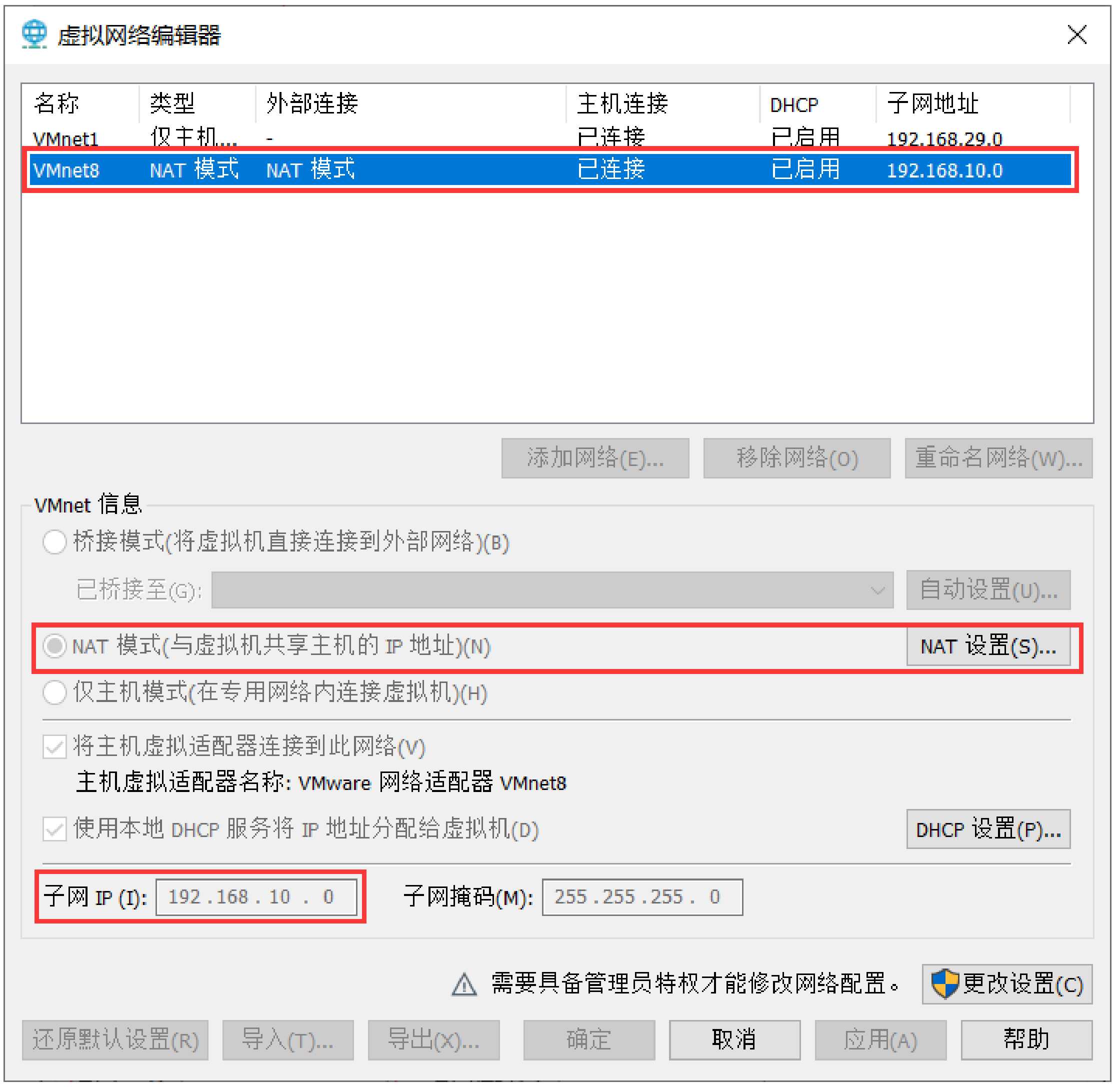
```
# vim /etc/sysconfig/network-scripts/ifcfgens33
```

Content:

```
DEVICE=ens33
TYPE=Ethernet
ONBOOT=yes
BOOTPROTO=static
NAME="ens33"
```

```
IPADDR=192.168.10.102
PREFIX=24
GATEWAY=192.168.10.2
DNS1=192.168.10.2
```

（2）Check out the Virtual Network Editor for the Linux Virtual Machine, edit-> virtual network editor->VMnet8



虚拟网络编辑器

| 名称 | 类型 | 外部连接 | 主机连接 | DHCP | 子网地址 |
|---|---|---|---|---|---|
| VMnet1 | 仅主机... | - | 已连接 | 已启用 | 192.168.29.0 |
| VMnet8 | NAT 模式 | NAT 模式 | 已连接 | 已启用 | 192.168.10.0 |

VMnet 信息

○ 桥接模式(将虚拟机直接连接到外部网络)(B)

已桥接至(G):

● NAT 模式(与虚拟机共享主机的 IP 地址)(N)

○ 仅主机模式(在专用网络内连接虚拟机)(H)

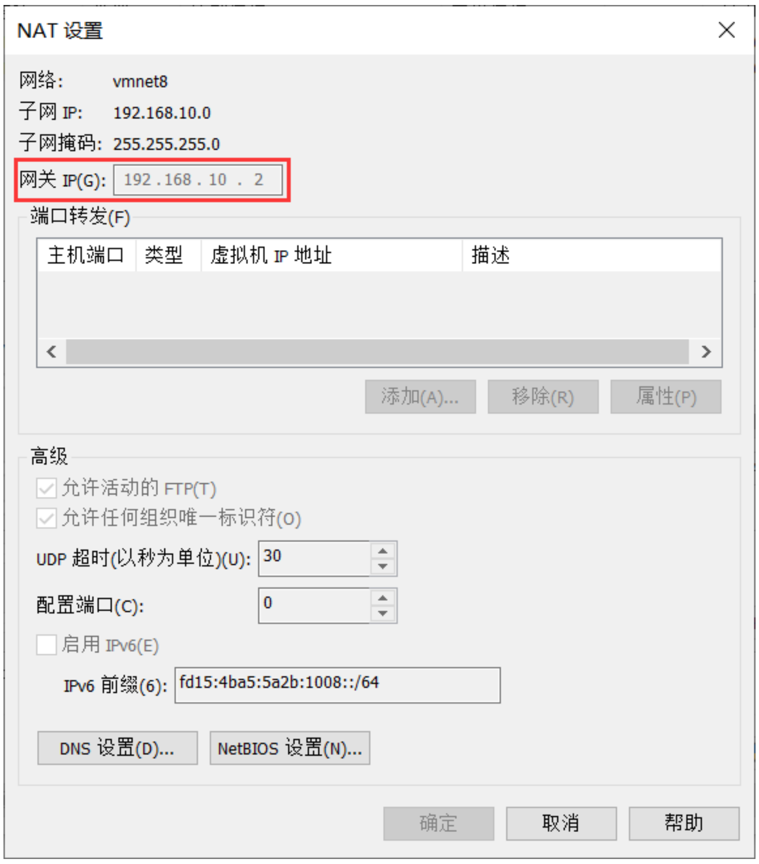
☑ 将主机虚拟适配器连接到此网络(V)

主机虚拟适配器名称: VMware 网络适配器 VMnet8

☑ 使用本地 DHCP 服务将 IP 地址分配给虚拟机(D)

子网 IP (I): 192 .168 . 10 . 0    子网掩码(M): 255 .255 .255 . 0

需要具备管理员特权才能修改网络配置。


NAT 设置

网络: vmnet8
子网 IP: 192.168.10.0
子网掩码: 255.255.255.0
网关 IP(G): 192 . 168 . 10 . 2

端口转发(F)

| 主机端口 | 类型 | 虚拟机 IP 地址 | 描述 |
|---|---|---|---|

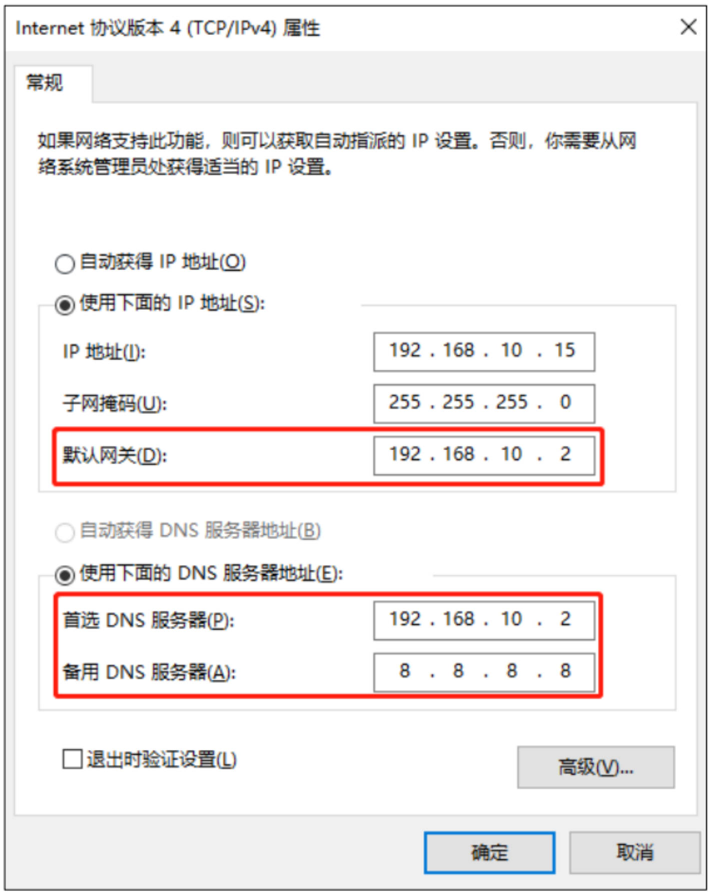
添加(A)... 移除(R) 属性(P)

高级
☑ 允许活动的 FTP(T)
☑ 允许任何组织唯一标识符(O)
UDP 超时(以秒为单位)(U): 30
配置端口(C): 0
☐ 启用 IPv6(E)
IPv6 前缀(6): fd15:4ba5:5a2b:1008::/64

DNS 设置(D)... NetBIOS 设置(N)...

确定 取消 帮助

（3）Look at the IP address of the Windows system Adapter: VMware Network Adapter VMNet8


Internet 协议版本 4 (TCP/IPv4) 属性

常规

如果网络支持此功能，则可以获取自动指派的 IP 设置。否则，你需要从网络系统管理员处获得适当的 IP 设置。

○ 自动获得 IP 地址(O)
● 使用下面的 IP 地址(S):
IP 地址(I): 192 . 168 . 10 . 15
子网掩码(U): 255 . 255 . 255 . 0
默认网关(D): 192 . 168 . 10 . 2

○ 自动获得 DNS 服务器地址(B)
● 使用下面的 DNS 服务器地址(E):
首选 DNS 服务器(P): 192 . 168 . 10 . 2
备用 DNS 服务器(A): 8 . 8 . 8 . 8

☐ 退出时验证设置(L)

高级(V)...

确定 取消

（4）Make sure that the IP address and virtual network editor address in the ifcfg-ens33 file on Linux are the same as the VM8 network IP address on Windows.

26. Change the hostname of two clone machines, for example, hadoop2:

（1）Modify host name

```
# vim /etc/hostname
hadoop2
```

（2）Configure the Linux clone machine host name mapping hosts file and open /etc/hosts

```
# vim /etc/hosts
```

Content:

```
192.168.10.100 hadoop0
192.168.10.101 hadoop1
192.168.10.102 hadoop2
```

27. restart.

```
# reboot
```

28. Modify the Windows host mapping file (hosts file)

（1）Enter the C:\Windows\System32\ Drivers \etc path

（2）Copy the hosts file to the desktop

（3）Open the desktop hosts file and add the following

```
192.168.10.100 hadoop0
192.168.10.101 hadoop1
192.168.10.102 hadoop2
```

（4）Overwrite the desktop hosts file to the C:\Windows\System32\ Drivers \etc path hosts file

### 2.2.3 Start to cluster

29. Keyless configuration

（1）Go to /home/hadoop/.ssh. Note that this folder will only be generated after the virtual machine has used the SSH directive.

（2）Generate public and private keys

```
$ ssh-keygen -t rsa
```

After three hits, two files will be generated: id_rsa, id_rsa.pub

（3）Copy the public key to the target machine to be logged in without secrecy, such as:

```
$ ssh-copy-id hadoop1
$ ssh-copy-id hadoop2
```

（4）Do this for each virtual machine that requires a key-free login.

30. Start the cluster

（1）If the cluster is started for the first time, the NameNode needs to be formatted in hadoop0

```
$ hdfs namenode -format
```

（2）start HDFS in hadoop0

```
$ sbin/start-dfs.sh
```

（3）Start YARN in hadoop1

```
$ sbin/start-yarn.sh
```

（4）view the startup situation:

```
$ jpsall
```

Hadoop0 displays: NameNode、DataNode、NodeManager、Jps
Hadoop1 displays: DataNode、ResourceManager、NodeManager、Jps
Hadoop2 displays: SecondaryNameNode、DataNode、NodeManager、Jps
The cluster started successfully.

（5）View the NameNode for HDFS on the Web side
（a）Type in the browser: http://hadoop0:9870
（b）View the data information stored on HDFS

（6）View the ResourceManager for YARN on the Web side
（a）Type in the browser: http://hadoop1:8088
（b）View the Job information running in YARN

31. Cluster start/stop method

（1）Each module starts/stops separately
Overall start/stop HDFS
start-dfs.sh/stop-dfs.sh
Overall start/stop YARN
start-yarn.sh/stop-yarn.sh

（2）Each service component starts/stops one by one
Start/stop HDFS components separately
hdfs –daemon start/stop namenode/datanode/secondarynamenode
Start/stop YARN components separately
yarn –daemon start/stop resourcemanager/nodemanager

# 3 Wordcount test

In local mode, run the first Hadoop program: hadoop-mapreduce-examples.

1. Create a wcinput folder under the hadoop-3.2.2 file

```
$ cd /opt/module/hadoop-3.2.2
$ mkdir wcinput
```

2. Create and edit the Word.txt file under the wcinput file

```
$ vim wcinput/word.txt
```

Enter arbitrary content in a file. Such as:

```
Hello World Bye World
Hello Hadoop Goodbye Hadoop
```

3. run hadoop-mapreduce-examples program

```
$ hadoop jar share/hadoop/mapreduce/hadoop-mapreduce-examples-3.2.2.jar
wordcount wcinput wcoutput
```

4. View Results

```
$ cat wcoutput/part-r-00000
```

Result:

```
Bye 1
Goodbye 1
Hadoop 2
Hello 2
World 2
```

# 4 Tutorial

Get a feel for how clustering works in fully distributed operation mode, HDFS and MapReduce.

1. Suppose you have HDFS and YARN started on three clients.
2. On hadoop0, upload the files to the cluster

   （1）Upload the word.txt created in Chapter 3

   ```
   $ hadoop fs -mkdir /input
   $ hadoop fs -put $HADOOP_HOME/wcinput/word.txt /input
   ```

   （2）Upload large files, JDK installs compressed files

   ```
   $ hadoop fs -put /opt/software/jdk-8u291- linux-x64.tar.gz /
   ```

3. After uploading the file, view the just uploaded file on HDFS.

   And on hadoop1, look at the HDFS file storage path

   ```
   $ cd /opt/module/hadoop-3.2.2/data/dfs/data/current/BP-1436128598-
   192.168.10.102- 1610603650062/current/finalized/subdir0/subdir0
   ```

   View HDFS to store file contents on disk

   ```
   $ cat blk_1073741825
   Hello World Bye World
   Hello Hadoop Goodbye Hadoop
   ```

4. On hadoop1, splicing

   ```
   $ cat blk_1073741826>>tmp.tar.gz
   $ cat blk_1073741827>>tmp.tar.gz
   $ tar -zxvf tmp.tar.gz
   ```

   You can observe that you are unpacking a JDK installation zip file.

5. On hadoop1, download it

   ```
   $ hadoop fs -get /jdk-8u291-linux-x64.tar.gz ./
   ```

6. On hadoop2, execute the wordcount program

   ```
   $ cd /opt/module/hadoop-3.2.2/
   $ hadoop jar share/hadoop/mapreduce/hadoop-mapreduce-examples-3.2.2.jar
   wordcount /input /output
   ```

   Job information can be viewed in Yarn while the program is running. After running, check the results under the /output path on HDFS. The results are stored in the PART-R-00000 file.