AN INTRODUCTION TO HDFS

HDFS namenodes and datanodes

Hadoop includes two main pieces: a distributed architecture for running MapReduce jobs, which are Java and other programs used to convert data from one format to another, and a distributed file system (HDFS) for storing data in a distributed architecture. Here we discuss HDFS.

In a regular file on a regular PC or server, the computer stores files in contiguous disk space called blocks. A central mechanism keeps track of which blocks are stored on which disk location. On a PC or server that is called the FAT (File Allocation Table) or NTFS (New Technology File System). Plus there are other structures that keep track of what data is written where.

On a magnetic disk drive the data is written to disk using a moving device called a disk controller. In HDFS writes are pushed across the network using Hadoop network protocols to individual servers which then use a Hadoop data note operations together with FAT or NTFS or something similar to write to the host machine's data blocks. This is usually locally attached storage, to drive down the cost, instead of a storage array, like a storage rack.(Low operating cost is one of the selling points of Hadoop.)

Hadoop removes the physical limitations of fixed disk sizes by storing file metadata (i.e., the location of blocks) in a namenode process on the master server and the data on any infinite number of datanodes.

Datanode processes running on each node in the cluster to write data to disk blocks there.

Unlike a regular file system, the HDFS can grow without limit as the architecture and administrator can add nodes at will. This abstraction of a single file across multiple computers lets files grow without limits as to their size.

But it also serves as a handy bucket notation where users can conventionally store files without having to give any thought to what directory they are mounted on, etc. For example they can copy any kind of file to hdfs://(server name):port and can retrieve that from any computer on the network.

Many other big data products like Spark, Storm, Cassandra Hbase, Hive, etc. use at least part of HDFS for their applications.

(This article is part of our Hadoop Guide. Use the right-hand menu to navigate.)

HDFS only writes data, does not update

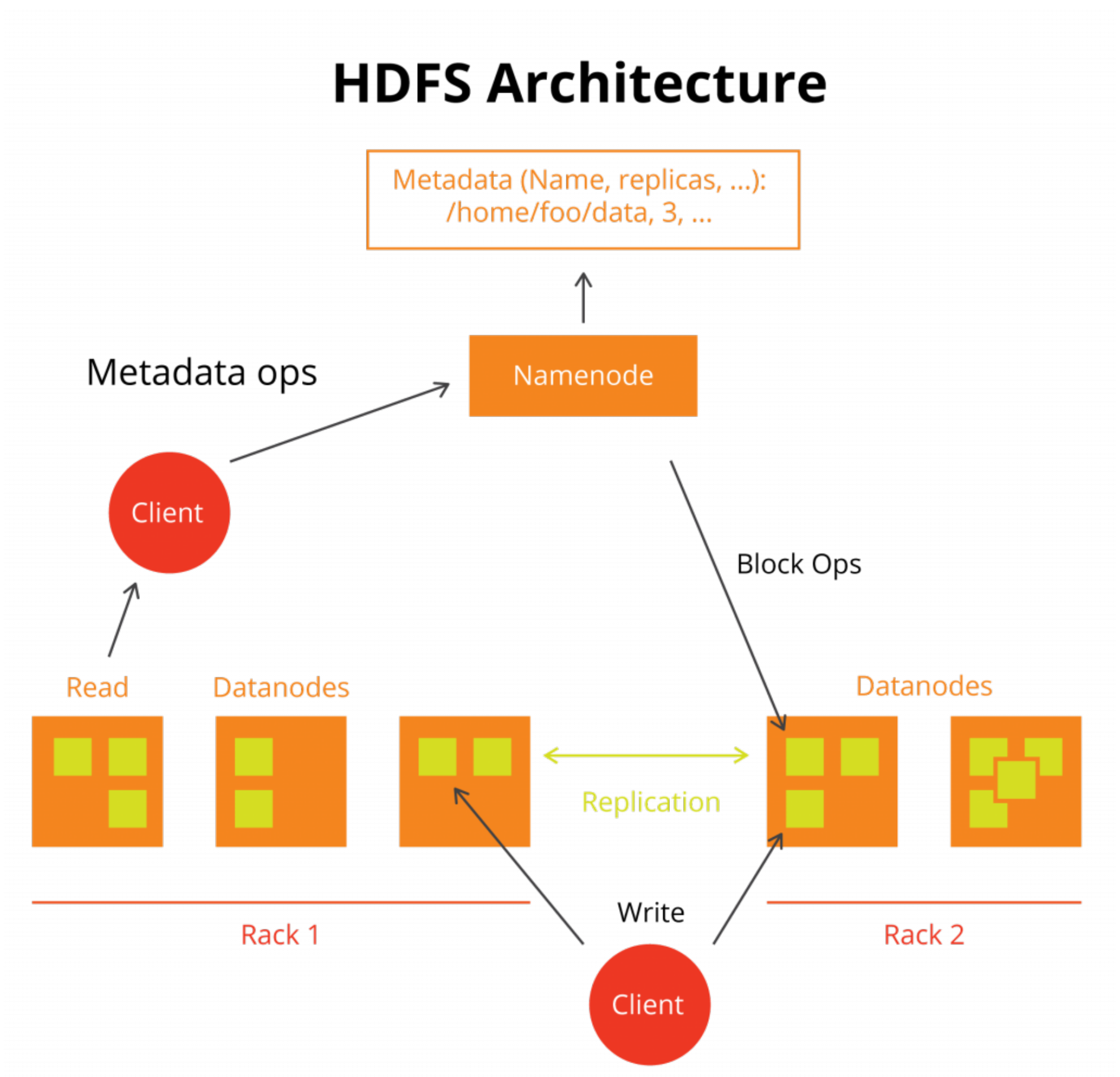
In Hadoop you can only write and delete files. You cannot update them.

The system is made to be resilient and fail proof because when each datanode writes its memory to disk data blocks, it also writes that memory to another server using replication. The number of replicas is set by the user. And Datanodes can be made rack aware. This is necessary because redundancy does not work when you write data to two disk drives in the same rack, that share the same electric power and network connections.

Data blocks on a regular PC are something like 4K or 64K in size. But in Hadoop they are often 64MB.

Hadoop is a batch system so it is not designed to read data fast. Instead it is designed to write large amounts of data and retrieve that using batch MapReduce jobs.

Below we show a diagram of the basic architecture.



Here we see a mount point/home/foo/data. This is called a Namespace. This folder does not exist on any disk, it is just an abstraction across the cluster. The namespace exists across all datanodes.

The namenode tells the datanodes where to write data. The datanodes report back into which block and drive they have written data to the namenode, so that one central repository can know that. The namenode duplicates that information into a secondary name node.

The datanodes write data to local storage in block format, just like a regular PC. So in the case of a failed disk drive the namenode can reconstruct the data on another data note. (In a large data center disk drives fail daily.) The namenode use the Editlog to keep track of all the write operations. The edit log is written to the namenode server on regular disk local storage. This file is called the FSImage. Hadoop truncates the Editlog as transactions is the FSImage metadata file are written to the datanodes.

HDFS CLI

You work with Hadoop files either from a client program using the API, like the Java program shown below, or the command line. The commands are almost exactly the same as regular Linux commands: ls, mkdir, chown, etc. So they are very easy to use. The only difference is you write "hadoop fs -(command". For example hadoop fs -ls lists files in a directory (namespace).

The full list of CLI commands are:

```
appendToFile cat checksum chgrp chmod chown copyFromLocal copyToLocal count
cp createSnapshot deleteSnapshot df du dus expunge find get getfacl getfattr
getmerge help ls lsr mkdir moveFromLocal moveToLocal mv put renameSnapshot rm
rmdir rmr setfacl setfattr setrep stat tail test text touchz truncate usage
```

The is command output looks just like regular Linux command:

```
hadoop fs -ls /data
-rw-r--r--   1 root supergroup    1081141 2017-03-31 15:37 /data/ssh.log
```

HDFS configuration

The HDFS parameters list is very long and is best checked in the Hadoop manual. In brief there are only a few parameters that you need to set to get Hadoop running in local or node mode. The most commons ones related to storage are shown below:

hdfs-site.xml

```
      dfs.namenode.name.dir
      file:///usr/local/hadoop

      dfs.datanode.name.dir
      file:///usr/local/hadoop
```

core-site.xml

```
      fs.defaultFS
      hdfs://hadoop-master:9000/

Turns on or off permissions checking.

      dfs.permissions
      false
```

Java

Hadoop is written in Java. Apache Pig creates Java MapReduce programs based upon SQL commands typed into the console. So it is good to know Java if you want to work with HDFS.
Here is an example program Hdfs.java to copy a String to a file in HDFS.
First source the environment:

```
export HADOOP_CLASSPATH=$($HADOOP_HOME/bin/hadoop classpath)
```

Compile the program like this:

```
javac -classpath ${HADOOP_CLASSPATH} Hdfs.java
```

Then run it like this:

```
java -cp .:$HADOOP_CLASSPATH Hdfs
```

Here is the source of Hdfs.java.

```
import java.io.BufferedInputStream;
import java.io.BufferedOutputStream;
import java.io.File;
import java.io.FileInputStream;
import java.io.FileOutputStream;
import java.io.IOException;
import java.io.InputStream;
import java.io.OutputStream;

import org.apache.hadoop.conf.Configuration;
import org.apache.hadoop.fs.FSDataInputStream;
import org.apache.hadoop.fs.FSDataOutputStream;
import org.apache.hadoop.fs.FileSystem;
import org.apache.hadoop.fs.Path;

public class Hdfs {


public static void main(String[] args) throws IOException {

Configuration conf = new Configuration ();
FileSystem fileSystem = FileSystem.get(conf);

        Path path = new Path("/data/silly.txt");

  FSDataOutputStream out = fileSystem.create(path);


String helloWorld = "Hello Hadoop World.";

out.writeBytes(helloWorld);

out.close();
fileSystem.close();

}

}
```

When we first run this we get this error:

```
Permission denied: user=walker, access=WRITE,
inode="/data":root:supergroup:drwxr-xr-x
```

So change the folder permission just as you would with the Linux chown command:

```
hadoop ds -chown walker /data
```

Then cat the file's contents to show that it workd:

```
hadoop fs -cat /data/silly.txt

Hello Hadoop World.
```

Hadoop file names

Hadoop files created by Map and Reduce are stored as a varying number of files in folders.
This is because such jobs are split into parallel processes and run across the cluster.
So when you run a MapReduce job and you set the output folder as:

```
FileOutputFormat.setOutputPath(job, "/data.txt);
```

The output is split into a files called "success" and "partn" in the folder /data.txt/ where n ranges from 1 to how every many partitions this step was divided into.

Hadoop file storage

Hadoop uses several file storage formats, including Avro, Parquet, Sequence, and Text.

Avro

Avro serialization for Hadoop writes data in JSON byte format so it can be consumed by programs written in any language. Avro is built into Hadoop. (Serialization means writing data to storage, like a field or web session or file.) With Avro you can represent complex structures (Arrays, customClass) in addition to primitives (.e.g., int, float, boolean).

Parquet

Parquet is a row and column storage format added to Hadoop by Cloudera and Twitter. It is like Hbase which is a storage mechanism on Hadoop to store columns of data, positioned close to each other for fast retrieval in a row-column storage design. HBase is the open source implementation for Google's Big Table database. Parquet is designed to serialize complex structures in bulk, like the classes you define in Java, Python, and other languages, together with their member fields and functions. Parquet supports APache Hive, Pig, and Spark storage as well.

Sequence Files

Is a JSON text file, but stored in binary (key-value= format. So (a=>b) would be stored as (01100001->01100010). There are 3 types of sequence files, the difference being whether the keys and values are compressed or not. Those uses these codecs for compression: BZip2Codec, DefaultCodec, GzipCodec. Without explaining how those work observe that a string of 6 bits of 1 could be written as something like "6 1's" thus taking up less space than 6 actual 1s.

Plain Text (csv and txt)

Needs no explanation.In Java it is designated:

```
job.setOutputFormatClass(TextOutputFormat.class);
```

Hadoop in the cloud

Different vendors have implemented their own Hadoop abstraction, for example Windows Azure and Amazon EC2 and S3. S3 in fact is already a URL-type file system, like Hadoop, replacing directory names like //server/directory/file with URLs http://. Cloudera has made Hadoop the bulk of its business.