Hadoop is written in Java,

Job’s setJarByClass() method is used to set the class, Hadoop will use to locate the relevant JAR file by looking for the JAR file containing this class.

An input path is specified by calling the static addInputPath() method on FileInputFormat, and it can be a single file, a directory (in which case, the input forms all the files in that directory), or a file pattern. As the name suggests, addInputPath() can be called more than once to use input from multiple paths.

The directory shouldn’t exist before running the job because Hadoop will complain and not run the job. This precaution is to prevent data loss accidentally

The tasks are scheduled using YARN and run on nodes in the cluster. If a task fails, it will be automatically rescheduled to run on a different node.

a good split size tends to be the size of an HDFS block, which is 128 MB by default

if you have multiple reducers, you will end up getting as many output part files like

-rw-r--r-- 1 pro supergroup 0 2019-05-17 13:12 hadoopMongoTestResult/\_SUCCESS

-rw-r--r-- 1 pro supergroup 26 2019-05-17 13:12 hadoopMongoTestResult/**part-r-00000**

-rw-r--r-- 1 pro supergroup 0 2019-05-17 13:12 hadoopMongoTestResult/**part-r-00001**

HDFS blocks are large compared to disk blocks, and the reason is to minimize the cost

of seeks. If the block is large enough, the time it takes to transfer the data from the disk

can be significantly longer than the time to seek to the start of the block. Thus, transferring a large file made of multiple blocks operates at the disk transfer rate.

There is a single namenode (the master) and a number of datanodes (workers).

It is also possible to run a secondary namenode, which despite its name does not act as

a namenode.

You can read HDFS file system from java

**public class URLCat** {

**static** {

URL.setURLStreamHandlerFactory(**new** FsUrlStreamHandlerFactory());

}

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

InputStream in = **null**;

**try** {

in = **new** URL(args[0]).openStream();

in.read();……

} **finally** {

IOUtils.closeStream(in);

}

}

}

Likewise, using Hadoop classes, it is also possible to write to HDFS using java code

Hadoop maintains Data Integrity while file transfers using CheckSum

In Hadoop, interprocess communication between nodes in the system is implemented using remote procedure calls (RPCs). The RPC protocol uses serialization

The Writable interface defines two methods—one for writing its state to a DataOutput binary stream and one for reading its state from a DataInput binary stream

ArrayWritable, BooleanWritable, IntWritable, DoubleWritable, FloatWritable are all subclasses as Writable.

Since Hadoop also sorts things, there is a generic RawComparator which compares 2 byte[]

Hadoop Text is Writable equivalent of java String

You can create your own Writable by implementing WritableComparable

Hadoop does not use java serialization. Why? Java Serialization doesn’t meet compact, fast, extensible, and interoperable criterias.

You can use xml files for application properties, of the format

<?xml version="1.0"?>

**<configuration>**

**<property>**

**<name>**color**</name>**

**<value>**yellow**</value>**

**<description>**Color**</description>**

**</property>**

**<property>**

**<name>**size**</name>**

**<value>**10**</value>**

**<description>**Size**</description>**

**</property>**

**</configuration>**

How to test Hadoop? Writing a Unit Test with MRUnit