Industrial Training Project Report

Ambedkar Institute of Advanced Communication Technologies and Research, Geeta Colony, Delhi-110031

2016

Submitted by:	Anunay Sinha
Branch:	C.S.E.
Semester:	5th
Roll Number:	03010102714

Industrial Training Report

Big Data Analysis using Hadoop

CANDIDATE'S DECLARATION

I, hereby declare that the work which is being presented in this project report entitled "**Big Data Analysis using Hadoop**" submitted, as a part of curriculum is an authentic record of my original work carried out under the guidance of Mr. Rakesh Kumar, IBM-KVCH, Sector-2, Noida.

I am highly obliged to IBM-KVCH, Sector-2, Noida.

Date: ______ Anunay Sinha

Place: ______

This is to certify that the above statement made by the candidate is correct to the best of my knowledge.

Rakesh Kumar
IBM-KVCH
Sector-2,NOIDA.

ACKNOWLEDGEMENT

It was a dream come true working with eminent developers of the esteemed organization IBM-KVCH. It’s a place where continuous stream of knowledge flows in the form of precious words, latest technology that can just be felt.

I am honored and grateful to Mr. Rakesh Kumar for introducing me to the vast world of “Big Data and Hadoop” and for his kind and continual support and constructive suggestions given during the course of this project.

CONTENTS

Chapter 1 What is Big Data?

In pioneer days they used oxen for heavy pulling, and when one ox couldn't budge a log, they didn't try to grow alarger ox. We shouldn't be trying for bigger computers, but for more systems of computers.
— Grace Hopper

Today we have finally arrived in the digital age. Data traditionally meant Documents, financial transactions, stock records and personal data but today it comprises of photos, audio, text, simulations, videos, location data and anything that exists in 1s and 0s.
According to a study, 90% of the digital data was generated in the last 2 years!! It came from smartphones, social media, trading platforms, Health Care, Scientific and other platforms. Today there are more than 2 billion internet users and 4.6 billion mobile phone users. The size of the "Digital Universe" was 4.4 zettabytes in 2013 and estimated to grow 10 folds by 2020. This huge amount of data is coming from various sources like

- 7TB of data is processed by Twitter every day.
- 10TB of data is processed by Facebook daily.
- The New York Stock Exchange generates about 4-5 terabytes of data per day.
- The Large Hadron Collider in Geneva produces about 30 petabytes of data per year.

So there is a lot of data out there and all of this data is what we call 'Big Data'. The trend is that the digital footprint of every individual is growing along with the data generated from machines around us which is leading to a growing mountain of data. It is said 'more data usually beats better algorithms' which is to say that the more data one has the more accurately can he provide results. The good news is that big data is the way to go but the bad news is that we are struggling to store and analyze it.
Bug Data spans over 3 Dimensions, namely

- Volume: Scale from terabytes to zettabytes. Enterprises are awash with ever-growing data of all types, easily amassing petabytes of information.
- Velocity: Analyze streaming data and large volumes of persistent data. Sometimes 2-minutes are too late. For time-sensitive processes such as catching fraud, big data must be used as it streams into the enterprise in order to maximize its value.
- Variety: Manage and benefit from diverse data types and data structures. Big data is any type-structured or unstructured like text, audio, sensor data, video, click streams, log files and more. New insights are found in analyzing these data types together.

So although the storage capabilities of hard drives have increased over the years, access speeds- the rate at which the data can be read from drives have not kept up. Back in 1990 1370 MB of data could be read from a drive in 5 minutes at a speed of 4.4 MB/s. Over 20 years later 1-terabyte of data requires more than two and a half hours to read all the data at around 100MB/s of speed. This is a long time to read all the data on a single drive and writing is even slower.
The obvious way to reduce the time is to read from multiple disk at once, spreading the data over multiple drives and reading the data on them simultaneously and processing the data in a way that they don't interfere each other means shorter analysis times.
There's more to being able to read and write data in parallel from multiple disks though.
The first problem is to solve hardware failure: as soon as we start using many pieces of hardware, the chance that they will fail increases considerably. A common practice to avoid this is data replication-keeping multiple copies of the data spread over multiple systems so that in case one copy fails another is available for processing.
The second problem is that most analysis tasks need to be able to combine the data in some way as data from one disk may need to be combined with data from some other disk spread over the various systems.

Various distribution systems allow data to be combined from multiple sources, but having a high accuracy is the challenge here.
This is where Hadoop comes in. Hadoop provides a reliable, scalable platform for storage and analysis. Hadoop runs on commodity hardware and is open source, in short Hadoop is affordable.

Chapter 2 A brief history of Hadoop

Hadoop was created by Doug Cutting, the creator of Apache Lucene, the widely used textsearch library. Hadoop has its origins in Apache Nutch, an open source web search engine, itself a part of the Lucene project.
The name Hadoop is not an acronym; it's a made-up name. The project's creator, Doug Cutting, explains how thename came about:The name my kid gave a stuffed yellow elephant. Short, relatively easy to spell and pronounce, meaningless, andnot used elsewhere: those are my naming criteria. Kids are good at generating such. Googol is a kid's term.
Projects in the Hadoop ecosystem also tend to have names that are unrelated to their function, often with an elephantor other animal theme ("Pig," for example). Smaller components are given more descriptive (and therefore moremundane) names. This is a good principle, as it means you can generally work out what something does from its name.
Building a web search engine from scratch was an ambitious goal, for not only is thesoftware required to crawl and index websites complex to write, but it is also a challengeto run without a dedicated operations team, since there are so many moving parts. It'sexpensive, too: Mike Cafarella and Doug Cutting estimated a system supporting a one-billion page index would cost around $500,000 in hardware, with a monthly running costof $30,000.Nevertheless, they believed it was a worthy goal, as it would open up andultimately democratize search engine algorithms.
Nutch was started in 2002, and a working crawler and search system quickly emerged.However, its creators realized that their architecture wouldn't scale to the billions of pageson the Web. Help was at hand with the publication of a paper in 2003 that described thearchitecture of Google's distributed filesystem, called GFS, which was being used inproduction at Google. GFS, or something like it, would solve their storage needs for the very large files generated as a part of the web crawl and indexing process. Inparticular, GFS would free up time being spent on administrative tasks such as managingstorage nodes. In 2004, Nutch's developers set about writing an open sourceimplementation, the Nutch Distributed Filesystem (NDFS).
In 2004, Google published the paper that introduced MapReduce to the world. Early in 2005, the Nutch developers had a working MapReduce implementation in Nutch, and bythe middle of that year all the major Nutch algorithms had been ported to run using MapReduce and NDFS.
NDFS and the MapReduce implementation in Nutch were applicable beyond the realm ofsearch, and in February 2006 they moved out of Nutch to form an independent subproject of Lucene called Hadoop. At around the same time, Doug Cutting joined Yahoo!, whichprovided a dedicated team and the resources to turn Hadoop into a system that ran at web scale (see the following sidebar). This was demonstrated in February 2008 when Yahoo! announced that its production search index was being generated by a 10,000-core Hadoop cluster.
In January 2008, Hadoop was made its own top-level project at Apache, confirming its success and its diverse, active community. By this time, Hadoop was being used by many other companies besides Yahoo!, such as Last.fm, Facebook, and the *New York Times*.
In one well-publicized feat, the *New York Times* used Amazon's EC2 compute cloud to crunch through 4 terabytes of scanned archives from the paper, converting them to PDFs for the Web. The processing took less than 24 hours to run using 100 machines, and the project probably wouldn't have been embarked upon without the combination ofAmazon's pay-by-the-hour model (which allowed the *NYT* to access a large number ofmachines for a short period) and Hadoop's easy-to-use parallel programming model.
In April 2008, Hadoop broke a world record to become the fastest system to sort an entire terabyte of data. Running on a 910-node cluster, Hadoop sorted 1 terabyte in 209 seconds (just under 3.5 minutes), beating the previous year's winner of 297 seconds.InNovember of the same year, Google reported that its MapReduce implementation sorted 1terabyte in 68 seconds. Then, in April 2009, it was announced that a team at Yahoo!had used Hadoop to sort 1 terabyte in 62 seconds.
The trend since then has been to sort even larger volumes of data at ever faster rates. In the

2014 competition, a team from Databricks were joint winners of the Gray Sort benchmark.
They used a 207-node Spark cluster to sort 100 terabytes of data in 1,406 seconds, a rate of 4.27 terabytes per minute.

Today, Hadoop is widely used in mainstream enterprises. Hadoop’s role as a generalpurpose storage and analysis platform for big data has been recognized by the industry,and this fact is reflected in the number of products that use or incorporate Hadoop in someway. Commercial Hadoop support is available from large, established enterprise vendors,including EMC, IBM, Microsoft, and Oracle, as well as from specialist Hadoopcompanies such as Cloudera, Hortonworks, and MapR.

Chapter 3 About Hadoop

Initially Hadoop consisted of the distributed file system, HDFS(Hadoop File Distribution System) and the processing algorithm MapReduce which was basically a batch query processor with the ability to run ad hoc query against the whole dataset and get the results in a reasonable time which was transformative. It gave people the opportunity to innovate with data. Questions that took too long to get answered before could now be answered, which in turn led to new questions and new insights.
Though for all its strengths, MapReduce is not suitable for interactive analysis. Queries typically take minutes or more, so it's best for offline use.
However, since its original incarnation, Hadoop has evolved beyond batch processing. The term 'Hadoop' is now used to refer to a larger ecosystem of projects, not just HDFS and MapReduce, that come under infrastructure for distributed computing and large-scale data processing. Many of these are hosted by the Apache Software Foundation, which provides support for a community of open source software projects.
The real enabler for new processing models in Hadoop was the introduction of YARN(which stands for *Yet Another Resource Negotiator*) in Hadoop 2. YARN is a clusterresource management system, which allows any distributed program (not justMapReduce) to run on data in a Hadoop cluster.
In the last few years, there has been a flowering of different processing patterns that workwith Hadoop. Here is a sample:

Interactive SQL
By dispensing with MapReduce and using a distributed query engine that uses dedicated "always on" daemons (like Impala) or container reuse (like Hive on Tez), it'spossible to achieve low-latency responses for SQL queries on Hadoop while still scalingup to large dataset sizes.

Iterative processing
Many algorithms — such as those in machine learning — are iterative in nature, so it's much more efficient to hold each intermediate working set in memory, compared toloading from disk on each iteration. The architecture of MapReduce does not allow this,but it's straightforward with Spark, for example, and it enables a highly exploratorystyle of working with datasets.

Stream processing
Streaming systems like Storm, Spark Streaming, or Samza make it possible to run real-time, distributed computations on unbounded streams of data and emit results to Hadoop storage or external systems.

Search
The Solr search platform can run on a Hadoop cluster, indexing documents as they are added to HDFS, and serving search queries from indexes stored in HDFS.

Despite the emergence of different processing frameworks on Hadoop, MapReduce stillhas a place for batch processing, and it is useful to understand how it works since itintroduces several concepts that apply more generally (like the idea of input formats, orhow a dataset is split into pieces).

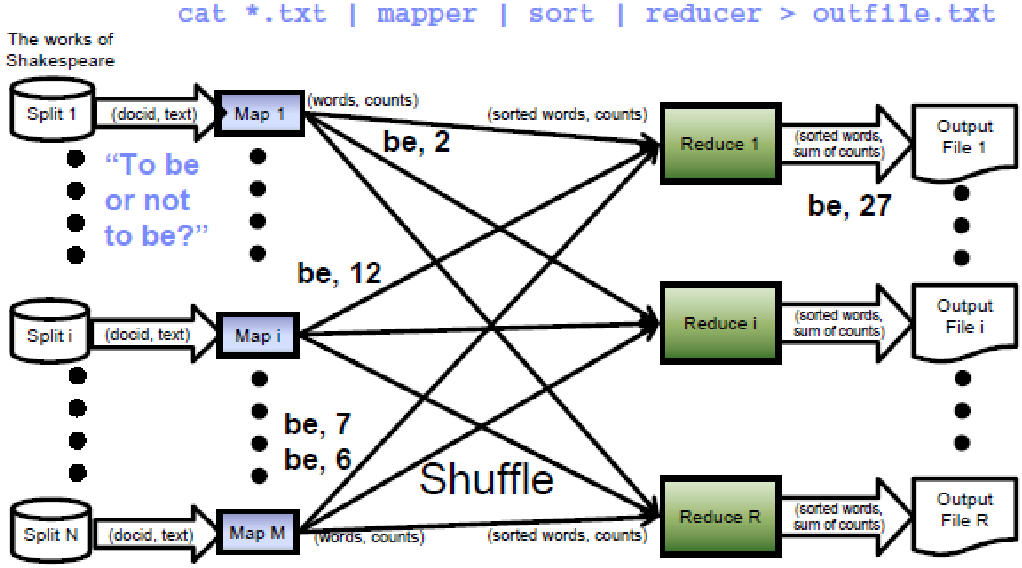
Chapter 4 MapReduce

MapReduce is designed to process huge datasets for certain kind of distributable problems using a large number of nodes. MapReduce works by breaking the processing into two phases: the map phase and the reduce phase. Each phase has key-value pairs as input and output. These phases are specified by two functions: the mapper function and the reducer function. In the Map phase the Master node partitions the input into smaller sub-problems and distributes the sub-problems to the worker nodes. All the Worker nodes generally do the same process. During the Reduce phase the Master node takes the answers to all the sub-problems and combines them in a way specified by the Reducer Class to get the Output.
Here we would perform a MapReduce Job on a .txt file to find the Word Count. In this we want to count the occurrences of words in the document. The Map function reads in a line of text and for each word encountered in the text emits an output pair that contains the word(as the key) and a value of 1(as the value).
What follows is the Sort phase which is invoked automatically by the MapReduce framework. Output of the key-value pairs from the Map function is sorted on the key value. After the sort, records are sent to the Reduce function nodes where Records with the same key value from all the Map function nodes are sent to the same Reduce function node. These Records then become the input to the Reducer function.
REDUCE is a family of higher-order functions that iterates an arbitrary function over a data structure in some order and builds up a return value. Typically a Reduce deals with two things- a combining function and a list of elements of some data structure. The output of Reduce is stored in HDFS. In this example, the key-value pair from the Map function is sorted(merged) on the key. The Reducer function reads all the key-value pairs for a particular word and totals the occurrences. It then outputs another key-value pair that contains the word(as the key) and the total of the word occurrences(as the value).
Sometimes another function is defined known as the Combiner Function. It is optional and lessens the data transferred between Map and Reduce tasks. For each Map function it takes the output and combines it into a lesser number of records which then becomes the input to a Reduce function.

MapReduce example: wordcount

Conceptually:

`cat *.txt | mapper | sort | reducer > outfile.txt`



Map(in_key, in_value) => list of (out_key, intermediate_value) **Reduce**(out_key, list of intermediate_values) => out_value(s)

This is the skeleton structure for the *Mapper* class, the *Reducer* class and the *main* class

Basic map code

```
public class MyMapper
      extends Mapper<LongWritable, Text, Text, IntWritable> {
  . . .
  @Override
  public void map(LongWritable key, Text value, Context context)
      throws IOException, InterruptedException {
   . . .
   context.write(new Text(…), new IntWritable(…));
   }
  }
}
```

Basic reduce code

```
public class MyReducer
      extends Reducer<Text, IntWritable, Text, IntWritable> {
  @Override
  public void reduce(Text key,
                     Iterable<IntWritable> values,
                     Context context)
      throws IOException, InterruptedException {
   . . .
   for (IntWritable value : values) {
      . . .
   }
   context.write(key, new IntWritable(…));
  }
}
```


main()

```
import org.apache.hadoop.fs.Path;
import org.apache.hadoop.io.IntWritable;
import org.apache.hadoop.io.Text;
import org.apache.hadoop.mapreduce.Job;
import org.apache.hadoop.mapreduce.lib.input.FileInputFormat;
import org.apache.hadoop.mapreduce.lib.output.FileOutputFormat;

pubic class MyMapReduce {

   public static void main(String[] args) throws Exception {
      . . .
      Job job = new Job();
      job.setJarByClass(MyMapReduce.class);
      job.setJobName("Some Name");
      FileInputFormat.addInputPath(job, new Path(args[0]));
      FileOutputFormat.setOutputForamt.setOutputPath(job,
                                                     new Path(args[1]));
      job.setMapperClass(MyMapper.class);
      job.setCombinerClass(MyReducer.class);
      job.setReducerClass(MyReducer.class);
      job.setOutputKeyClass(Text.class);
      job.setOutputValueClass(IntWritable.class);
      System.exit(job.waitForCompletion(true) ? 0 : 1);
   }
}
```

Chapter 5 HDFS – Hadoop Distributed File System

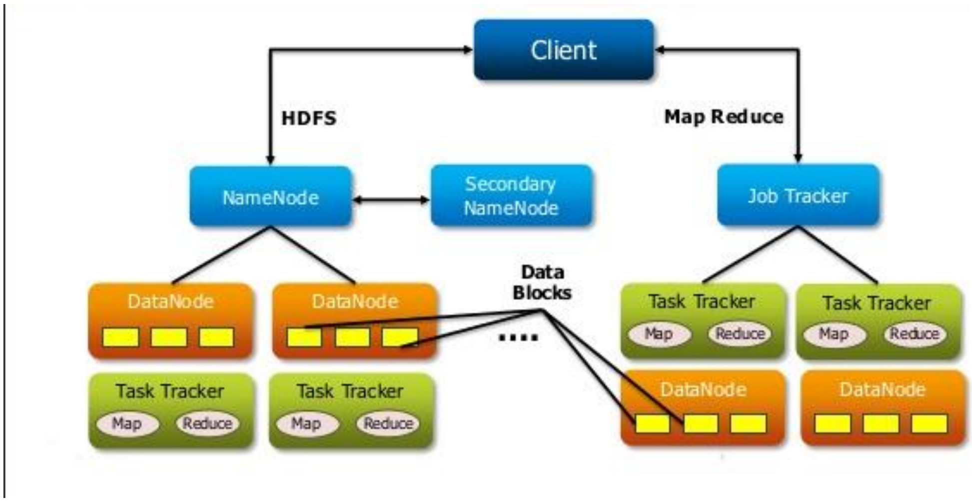
The HDFS was invented by Doug Cutting while he was working at Yahoo! The HDFS is tolerant to high component failure rate even when the HDFS works on community hardware. HDFS is designed to carry on working without a noticeable interruption to the user in the face of hardware failures. HDFS provides a new way of storing and processing the data. It lets the system handle most of the issues automatically like failures, scalability while using relatively inexpensive hardware.

It brings the processing to the data. It is well suited for dealing with hundreds of megabytes of data. The clusters running today store petabytes of data. But it is not well suited for small files because the namenode holds filesystem metadata in memory, the limit to the number of files in a filesystem is governed by the amount of memory on the namenode.

HDFS was built around the idea that the most efficient data processing pattern is a write-once, read-many-times pattern. The dataset is typically generated or copied from source and then various analyses are performed on the dataset over time. But at the same time it is not suited for low-latency access to the data. HDFS is optimized for delivering a high throughput of data and this may be at the expense of latency.

The Hadoop system consists of the following components,

- NameNode: An HDFS cluster has two types of nodes operating in a master-slave pattern, namely NameNode(the master) and a number of datanodes(workers). The namenode manages the filesystem namespace. It maintains the filesystem tree and the metadata for all the files and directories in the tree. The namenode also knows the datanodes on which all the blocks for a given file are located. However it doesn't store the block locations persistently as this information is reconstructed from datanodes when the system starts. There is only a single NameNode per cluster.
- DataNode: These are the workhorses of the filesystem. They store and retrieve blocks when they are told to(by clients or the namenode), and they report back to the namenode periodically with the list of blocks that they are storing. The files are stored in blocks of 128 MB each. Different blocks from the same file are stored on different DataNodes. The replication factor of these blocks is configurable as well as the size of each block is also configurable.
- Secondary NameNode: It is of importance to use reliable hardware for namenode as it is the single point of failure for HDFS. Also as a safeguard the Secondary namenode is used, which despite its name does not act as a namenode. Its main role is to periodically merge the namespace image with the edit log to prevent the edit log from becoming too large. The secondary namenode generally runs on a separate physical machine because it requires plenty of CPU and as much memory as the namenode to perform the merge. It keeps a copy of the merged namespace image, which can be used in the event of namenode failure. However, the state of secondary namenode lags that of the primary, so in the event of total failure of the primary, data loss is almost certain.
- JobTracker and TaskTracker: Only one JobTracker can exist per cluster. It manages the jobs in a Hadoop cluster. TaskTracker manages the jobs on datanodes and only one TaskTracker can exist per datanode. The JobTracker determines which TaskTrackers are to handle which tasks. It attempts to direct a task to the datanode where the data exists. The TaskTrackers run the MapReduce tasks in JVMs which have a set number of slots to run tasks. The TaskTrackers communicate with the JobTracker via heartbeat messages.


Client
HDFS
Map Reduce
NameNode
Secondary NameNode
Job Tracker
Data Blocks
DataNode
DataNode
Task Tracker
Map
Reduce
Task Tracker
Map
Reduce
....
Task Tracker
Map
Reduce
Task Tracker
Map
Reduce
DataNode
DataNode

Chapter 6 Hadoop Input/output

To access the data present in a Hadoop cluster or input data into a Hadoop cluster, following are few useful commands:

• **cat**
– Usage: hadoop fs -cat URI [URI …]
- Copies source paths to *stdout*.
- Example:

– hadoop fs -cat hdfs:/mydir/test_file1 hdfs:/mydir/test_file2
– hadoop fs -cat file:///file3 /user/hadoop/file4

• **chgrp**
– Usage: hadoop fs -chgrp [-R] GROUP URI [URI …]
- Change group association of files
- With -R, make the change recursively through the directory structure
-

• **chmod**
– Usage: hadoop fs -chmod [-R] <MODE[,MODE]... | OCTALMODE>
URI [URI …]
- Change the permissions of files.
- With -R, make the change recursively through the directory structure

• **chown**
– Usage: hadoop fs -chown [-R] [OWNER][:[GROUP]] URI [URI]
- Change the owner of files.
- With -R, make the change recursively through the directory structure.

• **copyFromLocal**
– Usage: hadoop fs -copyFromLocal <localsrc> URI:

• **copyToLocal**
– Usage: hadoop fs -copyToLocal [-ignoreCrc] [-crc] URI <localdst>

• **count**
– Usage: hadoop fs -count [-q] <paths>
- Count the number of directories, files and bytes under the paths that match the

specified file pattern.
- The output columns are:

DIR_COUNT, FILE_COUNT, CONTENT_SIZE FILE_NAME.
- The output columns with -q are:

QUOTA, REMAINING_QUATA, SPACE_QUOTA, REMAINING_SPACE_QUOTA, DIR_COUNT, FILE_COUNT, CONTENT_SIZE, FILE_NAME.
– Example:
- hadoop fs -count hdfs:/mydir/test_file1 hdfs:/mydir/test_file2
- hadoop fs -count -q hdfs:/mydir/test_file1

• **cp**
– Usage: hadoop fs -cp URI [URI …] <dest>
- Copy files from source to destination.
- This command allows multiple sources as well in which case the

destination must be a directory.
• Example:
– hadoop fs -cp hdfs:/mydir/test_file file:///home/hdpadmin/foo
– hadoop fs -cp file:///home/hdpadmin/foo file:///home/hdpadmin/boo hdfs:/mydir

• **du**
– Usage: hadoop fs -du URI [URI …]
• Displays aggregate length of files contained in the directory or the length of a file in case it's just a file.
• Example:
– hadoop fs -du file:///home/hdpadmin/test_file hdfs:/mydir

• **dus**
– Usage: hadoop fs -dus <args>
• Displays a summary of file lengths.

• **expunge**
– Usage: hadoop fs –expunge
• Empty the Trash

• **get**
– Usage: hadoop fs -get [-ignoreCrc] [-crc] <src><localdst>
• Copy files to the local file system.
– Files that fail the CRC check may be copied with the -ignoreCrc option.
– Files and CRCs may be copied using the -crc option.
• Example:
– hadoop fs -get hdfs:/mydir/file file:///home/hdpadmin/localfile

• **getmerge**
– Usage: hadoop fs -getmerge <src><localdst> [addnl]
• Takes a source directory and a destination file as input and concatenates files in the source into the destination local file.
• An additional option can be set to enable adding a newline character at the end of each file.

• **ls**
– Usage: hadoop fs -ls <args>
• For a file returns statistics on the file with the following format:
– permissions number_of_replicas userid groupid filesize modification_date modification_time Filename
• For a directory it returns list of its direct children as in UNIX .A directory is listed as:
– permissions userid groupid modification_date modification_time dirname
• Example:
– hadoop fs -ls hdfs:/mydir/test_file

• **lsr**
– Usage: hadoop fs -lsr <args>
• Recursive version of ls. Similar to Unix ls -R.
• Example:

– hadoop fs -lsr hdfs:/mydir

• **mkdir**
– Usage: hadoop fs -mkdir <paths>
• Takes path uri's as argument and creates directories. The behavior is much like unix mkdir -p creating parent directories along the path.
– Example:
• hadoop fs -mkdir hdfs:/mydir/foodir hdfs:/mydir/boodir

• **mv**
– Usage: hadoop fs -mv URI [URI …] <dest>
• Moves files from source to destination.
– This command allows multiple sources
> In which case the destination needs to be a directory.
• Moving files across file systems is not permitted.
•Example:
– hadoop fs -mv file:///home/hdpadmin/test_file file:///home/hdpadmin/test_file1
– hadoop fs –mv hdfs:/mydir/file1 hdfs:/mydir/file2 hdfs:/mydir2

• **put**
– Usage: hadoop fs -put <localsrc> ... <dst>
• Copy a single source, or multiple sources from local file system to the destination file system
• Reads input from stdin and writes to destination file system
– Notice that it can be the same file system.
• Examples:
– hadoop fs -put file:///home/hdpadmin/test_file hdfs:/mydir
– hadoop fs -put localfile1 localfile2 hdfs:/mydir
– hadoop fs -put - hdfs://mydir/input_file1 (reads input from stdin)

• **rm**
– Usage: hadoop fs -rm [-skipTrash] URI [URI …]
• Delete files specified as args
– Only deletes non empty directory and files.
• Example:
– hadoop fs -rm hdfs:/home/hdpadmin/test_file file:///home/hdpadmin/test_file

• **rmr**
– Usage: hadoop fs -rmr [-skipTrash] URI [URI …]
• Recursive version of delete Example:
– hadoop fs -rmr file:///home/hdpadmin/mydir
– hadoop fs -rmr –skipTrash hdfs:/mydir

• **setrep**
– Usage: hadoop fs –setrep [-w] [-R] <path>
• Changes the replication factor of a file
• Example:
– hadoop fs -setrep -w 5 -R hdfs:/user/hadoop/dir1

• **stat**

– Usage: hadoop fs -stat URI [URI …]

- Returns the statistical information on the path.
- Example:

– hadoop fs –stat hdfs:/mydir/test_file

• **tail**

– Usage: hadoop fs -tail [-f] URI

- Displays last kilobyte of the file to stdout. -f option can be used as in Unix.
- Example:

– hadoop fs -tail hdfs:/mydir/test_file

• **test**

– Usage: hadoop fs -test -[ezd] URI

• Options:

-e check to see if the file exists. Return 0 if true.

-z check to see if the file is zero length. Return 0 if true.

-d check to see if the path is directory. Return 0 if true.

•Example:

– hadoop fs -test –e hdfs:/mydir/test_file

• **text**

– Usage: hadoop fs -text <src>

- Takes a source file and outputs the file in text format.

• **touchz**

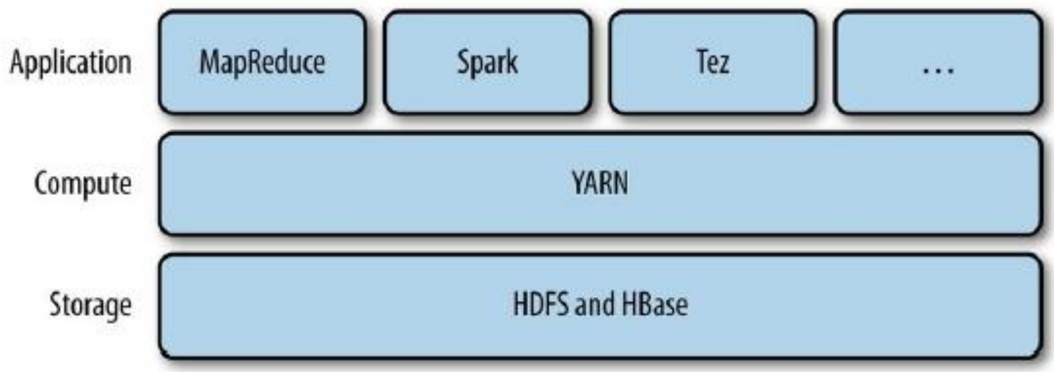
– Usage: hadoop fs -touchz URI [URI]

- Creates a file of zero length.
- Example:

– hadoop fs -touchz hdfs:/mydir/test_file

Chapter 7 YARN

Apache YARN (Yet Another Resource Negotiator) is Hadoop's cluster resourcemanagement system. YARN was introduced in Hadoop 2 to improve the MapReduceimplementation, but it is general enough to support other distributed computing paradigmsas well.
YARN provides APIs for requesting and working with cluster resources, but these APIs are not typically used directly by user code. Instead, users write to higher-level APIsprovided by distributed computing frameworks, which themselves are built on YARN andhide the resource management details from the user. YARN provides its core services via two types of long-running daemon: a *resourcemanager* (one per cluster) to manage the use of resources across the cluster, and *nodemanagers* running on all the nodes in the cluster to launch and monitor *containers*. Acontainer executes an application-specific process with a constrained set of resources(memory, CPU, and so on). Depending on how YARN is configured, a container may be aUnix process or a Linux cgroup. To run an application on YARN, a client contacts the resource manager and asks it to runan *application master* process. The resource manager then finds anode manager that can launch the application master in a container.
Precisely what the application master does once it is running depends on the application. It could simply run a computation in the container it is running in and return the result to the client. Or it could request morecontainers from the resource managers, and usethem to run a distributed computation.
YARN has a flexible model for making resource requests. A request for a set of containerscan express the amount of computer resources required for each container (memory and CPU), as well as locality constraints for the containers in that request.
Locality is critical in ensuring that distributed data processing algorithms use the cluster bandwidth efficiently, so YARN allows an application to specify locality constraints for the containers it is requesting. Locality constraints can be used to request a container on aspecific node or rack, or anywhere on the cluster (off-rack).
Sometimes the locality constraint cannot be met, in which case either no allocation is made or, optionally, the constraint can be loosened. For example, if a specific node wasrequested but it is not possible to start a container on it (because other containers arerunning on it), then YARN will try to start a container on a node in the same rack, or, ifthat's not possible, on any node in the cluster.
In the common case of launching a container to process an HDFS block (to run a map task in MapReduce, say), the application will request a container on one of the nodes hosting the block's three replicas, or on a node in one of the racks hosting the replicas, or, failingthat, on any node in the cluster.
A YARN application can make resource requests at any time while it is running. Forexample, an application can make all of its requests up front, or it can take a more dynamic approach whereby it requests more resources dynamically to meet the changingneeds of the application.
Spark takes the first approach, starting a fixed number of executors on the cluster. MapReduce, on the other hand, has two phases: the map task containersare requested up front, but the reduce task containers are not started until later. Also, if anytasks fail, additional containers will be requested so the failed tasks can be rerun.


Application
MapReduce
Spark
Tez
...
Compute
YARN
Storage
HDFS and HBase

Chapter 8 Cluster Setup and Installation

This section describes how to install and configure a basic Hadoop cluster from scratch using the Apache Hadoop distribution on a Unix operating system. It provides backgroundinformation on the things you need to think about when setting up Hadoop. For a production installation, most users and operators should consider one of the Hadoopcluster management tools listed at the beginning of this chapter.

Installing Java

Hadoop runs on both Unix and Windows operating systems, and requires Java to beinstalled. For a production installation, you should select a combination of operatingsystem, Java, and Hadoop that has been certified by the vendor of the Hadoop distributionyou are using. There is also a page on the Hadoop wikithat lists combinations thatcommunity members have run with success.

Creating Unix User Accounts

It's good practice to create dedicated Unix user accounts to separate the Hadoop processes from each other, and from other services running on the same machine. The HDFS, MapReduce, and YARN services are usually run as separate users, named hdfs, mapred, and yarn, respectively. They all belong to the same hadoop group.

Installing Hadoop

Download Hadoop from the Apache Hadoop releases page, and unpack the contents of thedistribution in a sensible location, such as */usr/local* (*/opt* is another standard choice; note that Hadoop should not be installed in a user's home directory, as that may be an NFSmounteddirectory):

% **cd /usr/local**

% **sudo tar xzf hadoop-*x.y.z*.tar.gz**

You also need to change the owner of the Hadoop files to be the hadoop user and group:

% **sudo chown -R hadoop:hadoop hadoop-*x.y.z***

It's convenient to put the Hadoop binaries on the shell path too:

% **export HADOOP_HOME=/usr/local/hadoop-*x.y.z***

% **export PATH=$PATH:$HADOOP_HOME/bin:$HADOOP_HOME/sbin**

Configuring SSH

The Hadoop control scripts (but not the daemons) rely on SSH to perform cluster-wide operations. For example, there is a script for stopping and starting all the daemons in the cluster. Note that the control scripts are optional — cluster-wide operations can be performed by other mechanisms, too, such as a distributed shell or dedicated Hadoopmanagement applications.

To work seamlessly, SSH needs to be set up to allow passwordless login for the hdfs andyarn users from machines in the cluster. The simplest way to achieve this is to generatea public/private key pair and place it in an NFS location that is shared across the cluster.

First, generate an RSA key pair by typing the following. You need to do this twice, once as the hdfs user and once as the yarn user:

% **ssh-keygen -t rsa -f ~/.ssh/id_rsa**

Even though we want passwordless logins, keys without passphrases are not considered good practice (it's OK to have an empty passphrase when running a local pseudodistributed cluster), so we specify a passphrase whenprompted for one. We use *ssh-agent* to avoid the need to enter a password for eachconnection.

The private key is in the file specified by the -f option, *~/.ssh/id_rsa*, and the public key is stored in a file with the same name but with *.pub* appended, *~/.ssh/id_rsa.pub*.

Next, we need to make sure that the public key is in the *~/.ssh/authorized_keys* file on all the machines in the cluster that we want to connect to. If the users' home directories arestored on an NFS filesystem, the keys can be shared across the cluster by typing thefollowing (first as hdfs and then as yarn):

% **cat ~/.ssh/id_rsa.pub >> ~/.ssh/authorized_keys**

If the home directory is not shared using NFS, the public keys will need to be shared bysome other means (such as *ssh-copy-id*).

Test that you can SSH from the master to a worker machine by making sure *ssh-agent* isrunning, and then run *ssh-add* to store your passphrase. You should be able to SSH to a worker without entering the passphrase again.

Configuring Hadoop

Hadoop must have its configuration set appropriately to run in distributed mode on a cluster. The important configuration settings to achieve this are discussed in the following table.

Filename	Format	Description
hadoop-env.sh	Bash script	Environment variables that are used in the scripts to run Hadoop
mapred-env.sh	Bash script	Environment variables that are used in the scripts to run MapReduce (overrides variables set in *hadoop-env.sh*)
yarn-env.sh	Bash script	Environment variables that are used in the scripts to run YARN (overrides variables set in *hadoop-env.sh*)
core-site.xml	Hadoop configuration XML	Configuration settings for Hadoop Core, such as I/O settings that are common to HDFS, MapReduce, and YARN
hdfs-site.xml	Hadoop configuration XML	Configuration settings for HDFS daemons: the namenode, the secondary namenode, and the datanodes
mapred-site.xml	Hadoop configuration XML	Configuration settings for MapReduce daemons: the job history server
yarn-site.xml	Hadoop configuration XML	Configuration settings for YARN daemons: the resource manager, the web app proxy server, and the node managers
slaves	Plain text	A list of machines (one per line) that each run a datanode and a node manager
hadoop-metrics2 .properties	Java properties	Properties for controlling how metrics are published in Hadoop (see Metrics and JMX)
log4j.properties	Java properties	Properties for system logfiles, the namenode audit log, and the task log for the task JVM process (Hadoop Logs)
hadoop-policy.xml	Hadoop configuration XML	Configuration settings for access control lists when running Hadoop in secure mode

Formatting the HDFS Filesystem

Before it can be used, a brand-new HDFS installation needs to be formatted. The formatting process creates an empty filesystem by creating the storage directories and theinitial versions of the namenode's persistent data structures. Datanodes are not involved inthe initial formatting process, since the namenode manages all of the filesystem'smetadata, and datanodes can join or leave the cluster dynamically. For the same reason,you don't need to say how large a filesystem to create, since this is determined by thenumber of datanodes in the cluster, which can be increased as needed, long after thefilesystem is formatted. Formatting HDFS is a fast operation. Run the following command as the hdfs user:
% **hdfs namenode -format**

Starting and Stopping the Daemons
Hadoop comes with scripts for running commands and starting and stopping daemons across the whole cluster. To use these scripts (which can be found in the *sbin* directory),you need to tell Hadoop which machines are in the cluster. There is a file for this purpose,called *slaves*, which contains a list of the machine hostnames or IP addresses, one per line.The *slaves* file lists the machines that the datanodes and node managers should run on. Itresides in Hadoop's configuration directory, although it may be placed elsewhere (andgiven another name) by changing the HADOOP_SLAVES setting in *hadoop-env.sh*. Also, this file does not need to be distributed to worker nodes, since they are used only by thecontrol scripts running on the namenode or resource manager.
The HDFS daemons are started by running the following command as the hdfs user:
% **start-dfs.sh**
The machine (or machines) that the namenode and secondary namenode run on is determined by interrogating the Hadoop configuration for their hostnames. For example,the script finds the namenode's hostname by executing the following:
% **hdfs getconf -namenodes**
By default, this finds the namenode's hostname from fs.defaultFS. In slightly more detail, the *start-dfs.sh* script does the following:

- Starts a namenode on each machine returned by executing hdfs getconf -namenodes
- Starts a datanode on each machine listed in the *slaves* file
- Starts a secondary namenode on each machine returned by executing hdfs getconf -secondarynamenodes

The YARN daemons are started in a similar way, by running the following command as the yarn user on the machine hosting the resource manager:
% **start-yarn.sh**
In this case, the resource manager is always run on the machine from which the *startyarn.sh* script was run. More specifically, the script:

- Starts a resource manager on the local machine
- Starts a node manager on each machine listed in the *slaves* file

Also provided are *stop-dfs.sh* and *stop-yarn.sh* scripts to stop the daemons started by the corresponding start scripts.
These scripts start and stop Hadoop daemons using the *hadoop-daemon.sh* script (or the *yarn-daemon.sh* script, in the case of YARN). If you use the aforementioned scripts, youshouldn't call *hadoop-daemon.sh* directly. But if you need to control Hadoop daemonsfrom another system or from your own scripts, the *hadoop-daemon.sh* script is a goodintegration point. Likewise, *hadoop-daemons.sh* (with an "s") is handy for starting thesame daemon on a set of hosts.
Finally, there is only one MapReduce daemon — the job history server, which is started asfollows, as the mapred user:
% **mr-jobhistory-daemon.sh start historyserver**

Creating User Directories
Once you have a Hadoop cluster up and running, you need to give users access to it. This involves creating a home directory for each user and setting ownership permissions on it:
% **hadoop fs -mkdir /user/*username***
% **hadoop fs -chown *username*:*username* /user/*username***
This is a good time to set space limits on the directory. The following sets a 1 TB limit on the given user directory:
% **hdfs dfsadmin -setSpaceQuota 1t /user/*username***

Chapter 9 Pig

Apache Pig raises the level of abstraction for processing large datasets. MapReduce allowsyou, as the programmer, to specify a map function followed by a reduce function, but working out how to fit your data processing into this pattern, which often requires multiple
MapReduce stages can be a challenge. With Pig, the data structures are much richer, typically being multivalued and nested, and the transformations you can apply to the dataare much more powerful. They include joins, for example, which are not for the faint ofheart in MapReduce.

Pig is made up of two pieces:

- The language used to express data flows, called *Pig Latin*.
- The execution environment to run Pig Latin programs. There are currently twoenvironments: local execution in a single JVM and distributed execution on a Hadoopcluster.

A Pig Latin program is made up of a series of operations, or transformations, that are applied to the input data to produce output. Taken as a whole, the operations describe adata flow, which the Pig execution environment translates into an executablerepresentation and then runs. Under the covers, Pig turns the transformations into a seriesof MapReduce jobs, but as a programmer you are mostly unaware of this, which allowsyou to focus on the data rather than the nature of the execution.
Pig is a scripting language for exploring large datasets. One criticism of MapReduce isthat the development cycle is very long. Writing the mappers and reducers, compiling andpackaging the code, submitting the job(s), and retrieving the results is a time-consuming business, and even with Streaming, which removes the compile and package step, the experience is still involved. Pig's sweet spot is its ability to process terabytes of data in response to a half-dozen lines of Pig Latin issued from the console. Indeed, it was created at Yahoo! to make it easier for researchers and engineers to mine the huge datasets there.
Pig is very supportive of a programmer writing a query, since it provides severalcommands for introspecting the data structures in your program as it is written. Even moreuseful, it can perform a sample run on a representative subset of your input data, so youcan see whether there are errors in the processing before unleashing it on the full dataset.
Pig was designed to be extensible. Virtually all parts of the processing path arecustomizable: loading, storing, filtering, grouping, and joining can all be altered by userdefinedfunctions (UDFs). These functions operate on Pig's nested data model, so they canintegrate very deeply with Pig's operators. As another benefit, UDFs tend to be morereusable than the libraries developed for writing MapReduce programs.
In some cases, Pig doesn't perform as well as programs written in MapReduce. However,the gap is narrowing with each release, as the Pig team implements sophisticatedalgorithms for applying Pig's relational operators. It's fair to say that unless you arewilling to invest a lot of effort optimizing Java MapReduce code, writing queries in PigLatin will save you time.

Installing and Running Pig

Pig runs as a client-side application. Even if you want to run Pig on a Hadoop cluster, there is nothing extra to install on the cluster: Pig launches jobs and interacts with HDFS(or other Hadoop filesystems) from your workstation.
Installation is straightforward. Download a stable release from:
http://pig.apache.org/releases.html, and unpack the tarball in a suitable place on your workstation:
% **tar xzf pig-*x.y.z*.tar.gz**
It's convenient to add Pig's binary directory to your command-line path. For example:
% **export PIG_HOME=~/sw/pig-*x.y.z***

% **export PATH=$PATH:$PIG_HOME/bin**
You also need to set the JAVA_HOME environment variable to point to a suitable Java installation.
Try typing pig -help to get usage instructions.

Local mode
In local mode, Pig runs in a single JVM and accesses the local filesystem. This mode is suitable only for small datasets and when trying out Pig.
The execution type is set using the -x or -exectype option. To run in local mode, set the option to local:
% **pig -x local**
grunt>
This starts Grunt, the Pig interactive shell.

Running Pig Programs
There are three ways of executing Pig programs, all of which work in both local and MapReduce mode:

- Script
  Pig can run a script file that contains Pig commands. For example, pig script.pig runsthe commands in the local file *script.pig*. Alternatively, for very short scripts, you canuse the -e option to run a script specified as a string on the command line.
- Grunt
  Grunt is an interactive shell for running Pig commands. Grunt is started when no file isspecified for Pig to run and the -e option is not used. It is also possible to run Pigscripts from within Grunt using run and exec.
- Embedded
  You can run Pig programs from Java using the PigServer class, much like you can useJDBC to run SQL programs from Java. For programmatic access to Grunt, usePigRunner.

Grunt
Grunt has line-editing facilities like those found in GNU Readline (used in the bash shelland many other command-line applications). For instance, the Ctrl-E key combination will move the cursor to the end of the line. Grunt remembers command history, too, and youcan recall lines in the history buffer using Ctrl-P or Ctrl-N (for previous and next), orequivalently, the up or down cursor keys.
Another handy feature is Grunt's completion mechanism, which will try to complete PigLatin keywords and functions when you press the Tab key. For example, consider the following incomplete line:
grunt>**a = foreach b ge**
If you press the Tab key at this point, ge will expand to generate, a Pig Latin keyword:
grunt>**a = foreach b generate**
You can customize the completion tokens by creating a file named *autocomplete* and placing it on Pig's classpath (such as in the *conf* directory in Pig's *install* directory) or inthe directory you invoked Grunt from. The file should have one token per line, and tokensmust not contain any whitespace. Matching is case sensitive. It can be very handy to addcommonly used file paths (especially because Pig does not perform filename completion)or the names of any user-defined functions you have created.
You can get a list of commands using the help command. When you've finished your Grunt session, you can exit with the quit command, or the equivalent shortcut \q.

Pig Latin Editors
There are Pig Latin syntax highlighters available for a variety of editors, includingEclipse, IntelliJ IDEA, Vim, Emacs, and TextMate. Details are available on the Pig wiki. Many Hadoop distributions come with the Hue web interface, which has a Pig script editorand launcher.

An Example

Let's look at a simple example by writing the program to calculate the maximum recorded temperature by year for the weather dataset in Pig Latin. The complete program is only a few lines long:

```
-- max_temp.pig: Finds the maximum temperature by year
records = LOAD 'input/ncdc/micro-tab/sample.txt'
AS (year:chararray, temperature:int, quality:int);
filtered_records = FILTER records BY temperature != 9999 AND
quality IN (0, 1, 4, 5, 9);
grouped_records = GROUP filtered_records BY year;
max_temp = FOREACH grouped_records GENERATE group,
MAX(filtered_records.temperature);
DUMP max_temp;
```

To explore what's going on, we'll use Pig's Grunt interpreter, which allows us to enter lines and interact with the program to understand what it's doing. Start up Grunt in localmode, and then enter the first line of the Pig script:

```
grunt>records = LOAD 'input/ncdc/micro-tab/sample.txt'
>>AS (year:chararray, temperature:int, quality:int);
```

For simplicity, the program assumes that the input is tab-delimited text, with each linehaving just year, temperature, and quality fields. (Pig actually has more flexibility thanthis with regard to the input formats it accepts, as we'll see later.) This line describes theinput data we want to process. The year:chararray notation describes the field's nameand type; chararray is like a Java String, and an int is like a Java int. The LOADoperator takes a URI argument; here we are just using a local file, but we could refer to anHDFS URI. The AS clause (which is optional) gives the fields names to make it convenientto refer to them in subsequent statements.

The result of the LOAD operator, and indeed any operator in Pig Latin, is a *relation*, whichis just a set of tuples. A *tuple* is just like a row of data in a database table, with multiplefields in a particular order. In this example, the LOAD function produces a set of (year,temperature, quality) tuples that are present in the input file. We write a relation with onetuple per line, where tuples are represented as comma-separated items in parentheses:

```
(1950,0,1)
(1950,22,1)
(1950,-11,1)
(1949,111,1)
```

Relations are given names, or *aliases*, so they can be referred to. This relation is given the records alias. We can examine the contents of an alias using the DUMP operator:

```
grunt>DUMP records;
(1950,0,1)
(1950,22,1)
(1950,-11,1)
(1949,111,1)
(1949,78,1)
```

We can also see the structure of a relation — the relation's *schema* — using the DESCRIBE operator on the relation's alias:

```
grunt>DESCRIBE records;
records: {year: chararray,temperature: int,quality: int}
```

This tells us that records has three fields, with aliases year, temperature, and quality, which are the names we gave them in the AS clause. The fields have the types given tothem in the AS clause, too. We examine types in Pig in more detail later.

The second statement removes records that have a missing temperature (indicated by avalue of 9999) or an unsatisfactory quality reading. For this small dataset, no records arefiltered out:

grunt>**filtered_records = FILTER records BY temperature != 9999 AND**

>>**quality IN (0, 1, 4, 5, 9);**

grunt>**DUMP filtered_records;**

(1950,0,1)

(1950,22,1)

(1950,-11,1)

(1949,111,1)

(1949,78,1)

The third statement uses the GROUP function to group the records relation by the year field. Let's use DUMP to see what it produces:

grunt>**grouped_records = GROUP filtered_records BY year;**

grunt>**DUMP grouped_records;**

(1949,{(1949,78,1),(1949,111,1)})

(1950,{(1950,-11,1),(1950,22,1),(1950,0,1)})

We now have two rows, or tuples: one for each year in the input data. The first field ineach tuple is the field being grouped by (the year), and the second field has a bag of tuplesfor that year. A *bag* is just an unordered collection of tuples, which in Pig Latin isrepresented using curly braces.

By grouping the data in this way, we have created a row per year, so now all that remainsis to find the maximum temperature for the tuples in each bag. Before we do this, let'sunderstand the structure of the grouped_records relation:

grunt>**DESCRIBE grouped_records;**

grouped_records: {group: chararray,filtered_records: {year: chararray,

temperature: int,quality: int}}

This tells us that the grouping field is given the alias group by Pig, and the second field is the same structure as the filtered_records relation that was being grouped. With thisinformation, we can try the fourth transformation:

grunt>**max_temp = FOREACH grouped_records GENERATE group,**

>>**MAX(filtered_records.temperature);**

FOREACH processes every row to generate a derived set of rows, using a GENERATEclause to define the fields in each derived row. In this example, the first field is group, which is just the year. The second field is a little more complex. Thefiltered_records.temperature reference is to the temperature field of thefiltered_records bag in the grouped_records relation. MAX is a built-in function forcalculating the maximum value of fields in a bag. In this case, it calculates the maximumtemperature for the fields in each filtered_records bag. Let's check the result:

grunt>**DUMP max_temp;**

(1949,111)

(1950,22)

We've successfully calculated the maximum temperature for each year.

Chapter 10 Hive

In "Information Platforms and the Rise of the Data Scientist," Jeff Hammerbacher describes Information Platforms as "the locus of their organization's efforts to ingest, process, and generate information," and how they "serve to accelerate the process of learning from empirical data."
One of the biggest ingredients in the Information Platform built by Jeff's team at
Facebook was Apache Hive, a framework for data warehousing on top of Hadoop. Hivegrew from a need to manage and learn from the huge volumes of data that Facebook wasproducing every day from its burgeoning social network. After trying a few differentsystems, the team chose Hadoop for storage and processing, since it was cost effective andmet the scalability requirements.
Hive was created to make it possible for analysts with strong SQL skills (but meager Javaprogramming skills) to run queries on the huge volumes of data that Facebook stored in HDFS. Today, Hive is a successful Apache project used by many organizations as a general-purpose, scalable data processing platform.
Of course, SQL isn't ideal for every big data problem — it's not a good fit for buildingcomplex machine-learning algorithms, for example — but it's great for many analyses, and it has the huge advantage of being very well known in the industry. What's more, SQLis the lingua franca in business intelligence tools (ODBC is a common bridge, forexample), so Hive is well placed to integrate with these products.
This chapter is an introduction to using Hive. It assumes that you have working knowledge of SQL and general database architecture; as we go through Hive's features,we'll often compare them to the equivalent in a traditional RDBMS.

Installing Hive

In normal use, Hive runs on your workstation and converts your SQL query into a series of jobs for execution on a Hadoop cluster. Hive organizes data into tables, which provide ameans for attaching structure to data stored in HDFS. Metadata — such as table schemas — is stored in a database called the *metastore*.
When starting out with Hive, it is convenient to run the metastore on your local machine.
In this configuration, which is the default, the Hive table definitions that you create will belocal to your machine, so you can't share them with other users. We'll see how to configure a shared remote metastore, which is the norm in production environments, inThe Metastore.
Installation of Hive is straightforward. As a prerequisite, you need to have the sameversion of Hadoop installed locally that your cluster is running.Of course, you maychoose to run Hadoop locally, either in standalone or pseudodistributed mode, whilegetting started with Hive.

WHICH VERSIONS OF HADOOP DOES HIVE WORK WITH?

Any given release of Hive is designed to work with multiple versions of Hadoop. Generally, Hive works with the latest stable release of Hadoop, as well as supporting a number of older versions, listed in the release notes. You don'tneed to do anything special to tell Hive which version of Hadoop you are using, beyond making sure that the *Hadoop* executable is on the path or setting the HADOOP_HOME environment variable.
Download a release, and unpack the tarball in a suitable place on your workstation:
% **tar xzf apache-hive-*x.y.z*-bin.tar.gz**
It's handy to put Hive on your path to make it easy to launch:
% **export HIVE_HOME=~/sw/apache-hive-*x.y.z*-bin**

% **export PATH=$PATH:$HIVE_HOME/bin**
Now type hive to launch the Hive shell:
% **hive**
hive>

The Hive Shell
The shell is the primary way that we will interact with Hive, by issuing commands in *HiveQL*. HiveQL is Hive's query language, a dialect of SQL. It is heavily influenced by MySQL, so if you are familiar with MySQL, you should feel at home using Hive.
When starting Hive for the first time, we can check that it is working by listing its tables — there should be none. The command must be terminated with a semicolon to tell Hiveto execute it:
hive>**SHOW TABLES;**
OK
Time taken: 0.473 seconds
Like SQL, HiveQL is generally case insensitive (except for string comparisons), so show tables; works equally well here. The Tab key will autocomplete Hive keywords andfunctions.
For a fresh install, the command takes a few seconds to run as it lazily creates the metastore database on your machine. (The database stores its files in a directory called *metastore_db*, which is relative to the location from which you ran the hive command.)
You can also run the Hive shell in noninteractive mode. The -f option runs the commands in the specified file, which is *script.q* in this example:
% **hive -f script.q**
For short scripts, you can use the -e option to specify the commands inline, in which case the final semicolon is not required:
% **hive -e 'SELECT * FROM dummy'**
OK
X Time taken: 1.22 seconds, Fetched: 1 row(s)

NOTE
It's useful to have a small table of data to test queries against, such as trying out functions in SELECT expressions using literal data (see Operators and Functions). Here's one way of populating a single-row table:
% **echo 'X' > /tmp/dummy.txt**
% **hive -e "CREATE TABLE dummy (value STRING); **
**LOAD DATA LOCAL INPATH '/tmp/dummy.txt' **
OVERWRITE INTO TABLE dummy"
In both interactive and noninteractive mode, Hive will print information to standard error— such as the time taken to run a query — during the course of operation. You cansuppress these messages using the -S option at launch time, which has the effect ofshowing only the output result for queries:
% **hive -S -e 'SELECT * FROM dummy'**
X
Other useful Hive shell features include the ability to run commands on the host operating system by using a ! prefix to the command and the ability to access Hadoop filesystems using the dfs command.

An Example

Let's see how to use Hive to run a query on the weather dataset we explored in earlier chapters. The first step is to load the data into Hive's managed storage. Here we'll have
Hive use the local filesystem for storage; later we'll see how to store tables in HDFS.
Just like an RDBMS, Hive organizes its data into tables. We create a table to hold the weather data using the CREATE TABLE statement:

```
CREATE TABLE records (year STRING, temperature INT, quality INT)
ROW FORMAT DELIMITED
FIELDS TERMINATED BY '\t';
```

The first line declares a records table with three columns: year, temperature, andquality. The type of each column must be specified, too. Here the year is a string, whilethe other two columns are integers.
So far, the SQL is familiar. The ROW FORMAT clause, however, is particular to HiveQL. Thisdeclaration is saying that each row in the data file is tab-delimited text. Hive expects thereto be three fields in each row, corresponding to the table columns, with fields separated bytabs and rows by newlines.
Next, we can populate Hive with the data. This is just a small sample, for exploratory purposes:

```
LOAD DATA LOCAL INPATH 'input/ncdc/micro-tab/sample.txt'
OVERWRITE INTO TABLE records;
```

Running this command tells Hive to put the specified local file in its warehouse directory.
This is a simple filesystem operation. There is no attempt, for example, to parse the file and store it in an internal database format, because Hive does not mandate any particularfile format. Files are stored verbatim; they are not modified by Hive.
In this example, we are storing Hive tables on the local filesystem (fs.defaultFS is set toits default value of file:///). Tables are stored as directories under Hive's warehousedirectory, which is controlled by the hive.metastore.warehouse.dir property anddefaults to */user/hive/warehouse*.
Thus, the files for the records table are found in the */user/hive/warehouse/records*directory on the local filesystem:

```
% ls /user/hive/warehouse/records/
sample.txt
```

In this case, there is only one file, *sample.txt*, but in general there can be more, and Hivewill read all of them when querying the table.
The OVERWRITE keyword in the LOAD DATA statement tells Hive to delete any existing filesin the directory for the table. If it is omitted, the new files are simply added to the table's directory (unless they have the same names, in which case they replace the old files).
Now that the data is in Hive, we can run a query against it:

```
hive>SELECT year, MAX(temperature)
>FROM records
>WHERE temperature != 9999 AND quality IN (0, 1, 4, 5, 9)
>GROUP BY year;
1949 111
1950 22
```

This SQL query is unremarkable. It is a SELECT statement with a GROUP BY clause for grouping rows into years, which uses the MAX aggregate function to find the maximum temperature for each year group. The remarkable thing is that Hive transforms this query into a job, which it executes on our behalf, then prints the results to the console. There aresome nuances, such as the SQL constructs that Hive supports and the format of the datathat we can query — and we explore some of these in this chapter — but it is the ability toexecute SQL queries against our raw data that gives Hive its power.

References

1. IBM InfosphereBigInsights Training Manual
2. Hadoop the Definitive Guide 4th Edition, Tom White
3. www.wikipedia.org
4. “Introduction to Hadoop” course by IBM Information Management.
5. “Welcome to Hadoop” course by Big Data University.