1

1 *Soft introduction to Hadoop*

Our goal here is to present an intro to Hadoop so simple that any programmer who reads it will be able to write simple Hadoop solutions and run them on a Hadoop cluster.

First, however, let us have the two basic definitions - what is Hadoop and what is MapReduce?

MapReduce is a programming framework, published by Google in 2004. Much like other frameworks, such as Spring, Struts, or MFC, the MapReduce framework does some things for you, and provides the place for you to fill in the blanks. What MapReduce does for you is to organize your multiple computers in a cluster in order to perform the calculations you need. It takes care of distributing the work between computers and of putting together the results of each computer's computation. Just as important, it takes care of hardware and network failures, so that they do not affect the flow of your computation. You, in turn, have to break your problem into separate pieces which can be processed in parallel by multiple machines, and you provide the code to do the actual calculation.

Hadoop is an open-source implementation of the MapReduce framework. For Google everything is MapReduce: both the way to write their applications, and the actual implementation are called MapReduce. For the rest of the world, the framework, the theory, and the way you spell out the computations are all called MapReduce. However, the actual software that implements this is called Hadoop.

Let us agree for simplicity that MapReduce and Hadoop are just two names for the same things. Note, however, that when Google teaches college students the ideas of MapReduce programming, they, too, use Hadoop, because their actual

implementation of MapReduce is proprietary code. To emphasize the difference, we can note that the Hadoop engineers at Yahoo like to challenge the engineers at Google to sorting competitions between Hadoop and MapReduce.

1.1 Why Hadoop?

We have already mentioned that the MapReduce framework is used at Google, Yahoo, Facebook. It sees big uptake in finance, retail, telecom, and the government. It is making inroads into life sciences. Why is this?

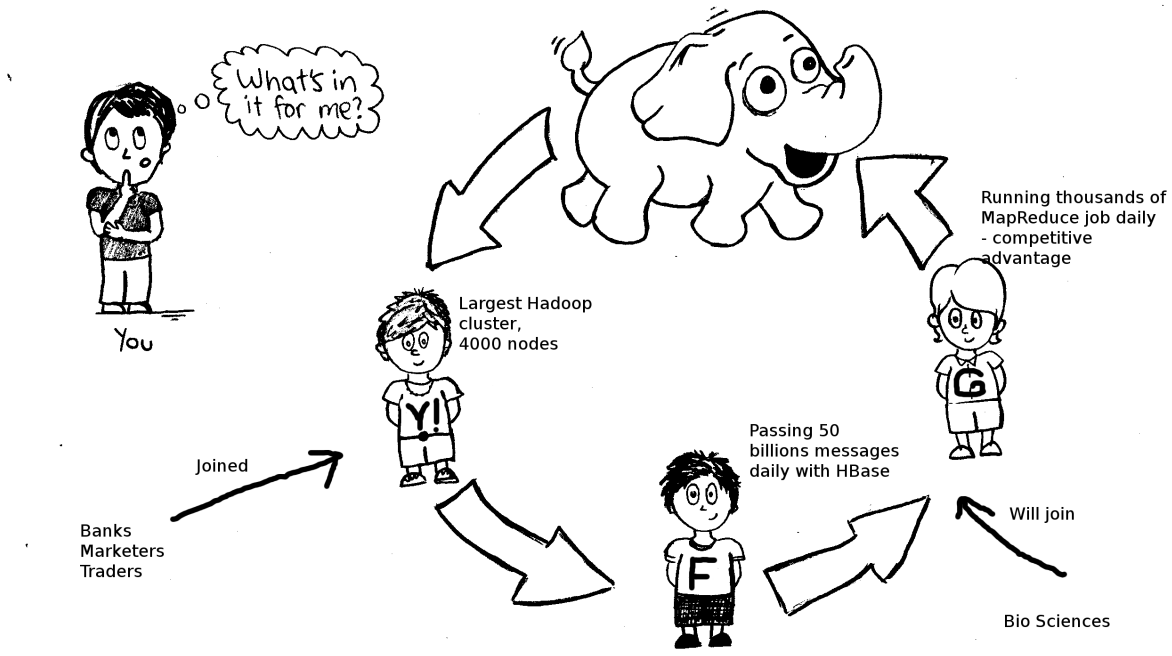

Figure 1.1 Will you join the Hadoop dance?

The short answer is that it simplifies dealing with Big Data. This answer immediately resonates with people, it is clear and succinct, but it is not complete. The Hadoop framework has built-in power and flexibility to do what you could not do before. In fact, Cloudera presentations at the latest O'Reilly Strata conference mentioned that MapReduce was initially used at Google and Facebook not primarily for its scalability, but for what it allowed do with the data.

In 2010, the average size of the Cloudera's customers' cluster was 30 machines. In 2011 it was 70. When people start using Hadoop, they do it for many reasons, all concentrated around the new ways of dealing with the data. What gives them the security to go ahead is the knowledge that Hadoop solutions are massively scalable, as has been proved by the Hadoop running in the world's largest computer centers and at largest companies.

As you will discover, the Hadoop framework organizes the data and the computations, and then runs your code. At times, it makes sense to run your solution, expressed in MapReduce paradigm, even on a single machine.

But of course, Hadoop really shines when you have not one, but rather tens, hundreds, or thousands of computers. If your data or computations are significant enough (and whose aren't these days?), then you need more than one machine to do the number crunching. If you try to organize the work yourself, you will soon discover that you have to coordinate the work of many computers, handle failures, retries, and collect the results together, and so on. Enter Hadoop to solve all these problems for you. Now that you have a hammer, everything becomes a nail: people will often reformulate their problem in MapReduce terms, rather than create a new custom computation platform.

No less important than Hadoop itself are its many friends. Hadoop Distributed File System (HDFS) provides unlimited file space available from any Hadoop node. HBase is a high-performance unlimited-size database working on top of Hadoop. If you need the power of familiar SQL over your large datasets, Pig provides you with an answer. While Hadoop can be used by programmers and taught to students as an introduction to Big Data, its companion projects (including the ZooKeeper, about which we will hear later on) will make possible and simplify, by providing tried-and-proven frameworks, every aspect of dealing with large datasets.

As you learn the concepts, and perfect your skills with the techniques described in this book, you will discover that there are many cases where Hadoop storage, Hadoop computation, or Hadoop's friends can help you. Let's look at some of these situations.

- Do you find yourself often cleaning the limited hard drives in your company? Do you need to transfer data from one drive to another, as a backup? Many people are so used to this necessity, that they consider it an unpleasant but unavoidable part of life. Hadoop distrubuted file system, HDFS, grows by adding servers. To you it looks like one hard drive. It is self-replicating (you set the replication factor) and thus provides redundancy as a software alternative to RAID.
- Do your computations take an unacceptably long time? Are you forced to give up on projects because you don’t know how to easily distribute the computations between multiple computers? MapReduce helps you solve these problems. What if you don’t have the hardware to run the cluster? - Amazon EC2 can run MapReduce jobs for you, and you pay only for the time that it runs - the cluster is automatically formed for you and then disbanded.
- But say you are lucky, and instead of maintaining legacy software, you are charged with building new, progressive software for your company's workflow. Of course, you want to

have unlimited storage, solving this problem once and for all, so as to concentrate on what's really important. The answer is: you can mount HDFS as FUSE file system, and you have your unlimited storage. In our cases studies we look at the successful use of the HDFS as as grid storage for the Large Hadron Collider.

- Imagine you have multiple clients using your online resources, computations, or data. Each single use is saved in a log, and you need to generate a summary of use of resources for each client by day or by hour. From this you will do your invoices, so it IS important. But the data set is large. You can write a quick MapReduce job for that. Better yet, you can use Hive, a data warehouse infrastructure built on top of Hadoop, with its ETL capabilities, to generate your invoices in no time. We'll talk about Hive later, but we hope that you already see that you can use Hadoop and friends for fun and profit.

Once you start thinking without the usual limitations, you can improve on what you already do and come up with new and useful projects. In fact, this book partially came about by asking people how they used Hadoop in their work. You, the reader, are invited to submit your applications that became possible with Hadoop, and we, the authors, get to do the second edition.

1.2 Meet the Hadoop Zoo

(Epigraph)

QUINCE

Is all our company here?

BOTTOM

You were best to call them generally, man by man, according to the scrip.

Shakespeare, Midsummer Night's Dream

There are a number of animals in the Hadoop zoo, and each deals with a certain aspect of Big Data. Let us illustrate this with a picture, and then introduce them one by one.

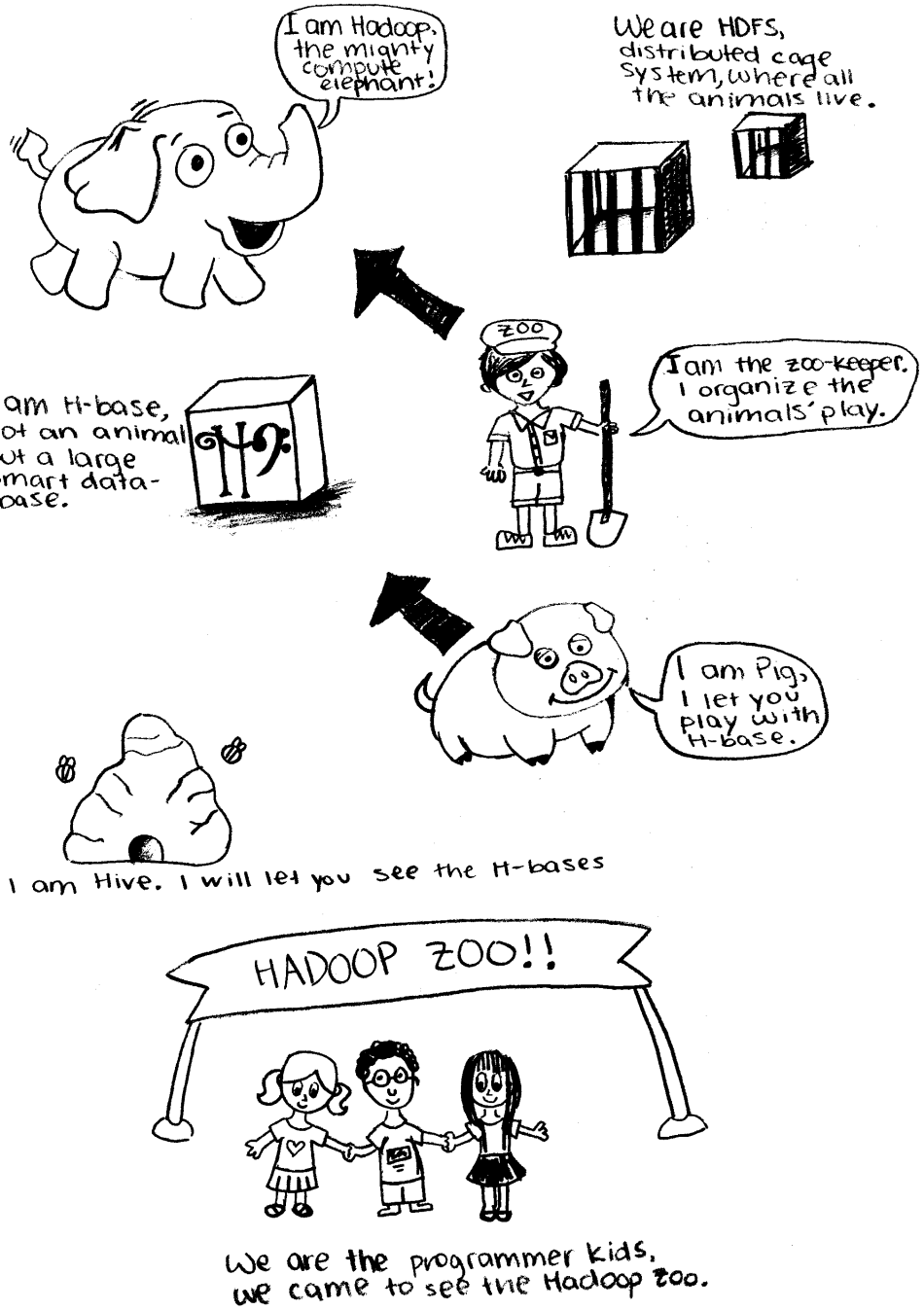

Figure 1.2 The Hadoop Zoo

1.2.1 HDFS - Hadoop Distributed File System

HDFS, or Hadoop Distributed File System, gives the programmer unlimited storage. Unlimited storage is a programmers's dream, so you can skip the rest. However, here are the additional advantages of HDFS.

- Horizontal scalability. Thousands of servers holding petabytes of data. When you need more more storage, you don't switch to more expensive solutions, but add servers to it.
- Commodity hardware. The HDFS is designed with relatively cheap commodity hardware in mind. HDFS is self-healing and replicating.
- Fault tolerance. Every member of the Hadoop zoo knows how to deal with hardware failures. If you have 10 thousand servers, then you will see one server fail every day, on the average. HDFS foresees that by replicating the data, by default three times, on different data node servers. Thus, if one data node fails, the other two can be used to restore the third one in a different place.

The HDFS implementation is modeled after GFS, Google Distributed File system, thus you can read the first paper on this, to be found here: http://labs.google.com/papers/gfs.html.

1.2.2 Hadoop, the little elephant

Hadoop organizes your computations. It reads the data, usually from HDFS, in blocks. However, it can read the data from other places too, including mounted local file systems, web, and databases. It divides the computations between different computers (servers, or nodes). It is also fault-tolerant.

If some of your nodes fail, Hadoop knows how to continue with the computation, by re-assigning the incomplete work to another node and cleaning up after that node that could not complete its task. It also knows how to combine the results of the computation in one place.

1.2.3 HBase, the database for Big Data

Not properly an animal, HBase is nevertheless very powerful. It is currently denoted by the letter H with a base clef. If you think this is not so great, you are right, and the HBase people are thinking of changing the logo. HBase is a database for Big Data, up to millions of columns and billions of rows.

Another feature of HBase is that it is a key-value database, not a relational database. We will get into the pros and cons of these two approaches to databases later, but for now let's just note that key-value databases are considered as more fitting for Big Data.

1.2.4 ZooKeeper

Every zoo has ZooKeepper, and the Hadoop zoo is no exception. When all the Hadoop animals want to do something together, it is the ZooKeeper who helps them do it. They all know him and listen and obey his commands. Thus, the ZooKeeper is a centralized service for maintaining configuration information, naming, providing distributed synchronization, and providing group services.

ZooKeeper is also fault-tolerant. In your environment development, you can put the zookeeper on one node, but in production you usually run it on an odd number of servers, such as 3 or 5.

1.2.5 Hive - data warehousing

Hive is a way for you to get all the honey, and to leave all the work to the bees. You can do a lot of data analysis with Hadoop, but you will also have to write MapReduce tasks. Hive tasks it upon itself. Hive defines a simple SQL-like query language, called QL, that enables users familiar with SQL to query the data.

At the same time, if your Hive program does almost what you need, but not quite, you can call on your MapReduce skill. Hive allows you to write custom mappers and reducers to extend the QL capabilities.

1.2.6 Pig - Big Data manipulation

Pig is called this way not because it eats a lot, although you can imagine a pig pushing around and consuming big volumes of information. Rather, it is called this way because it speaks Pig Latin. Others who also speak this language are the kids - or the programmers, who visit the Hadoop zoo.

So what is Pig Latin that Apache Pig speaks? As a rough analogy, if Hive is the SQL of Big Data, then Pig Latin is the language of the stored procedures of Big Data. It allows you to manipulate large volume of information, analyze them, and create new derivative data sets. Internally it creates a sequence of MapReduce jobs, and thus you, the programmer-kid, can use this simple language to solve pretty sophisticated large-scale problems.

1.3 Hadoop alternatives

Now that we have met the Hadoop zoo, we are ready to start our excursion. Only one thing stops are at this point - and that is, a gnawing doubt, are we in the right zoo? Let us look at some alternatives to dealing with Big Data. Granted, our concentration here is Hadoop, and we may not give justice to all the other approaches. But we will try.

1.3.1 Large data storage alternatives

HDFS is not the only, and in fact, not the earliest or the latest distributed file system. CEPH claims to be more flexible and to remove the limit on the number of files. HDFS stored all of its file information in the memory of the server which is called the NameNode. This is its strong point - speed - but it is also its Achilles' heel. CEPH, on the other hand, makes the function of the NameNode completely distributed.

Another possible contender is ZFS, an open-source file system from SUN, and currently Oracle. Intended as a complete redesign of file system thinking, ZFS holds a strong promise of unlimited size, robustness, encryption, and many other desirable qualities built into the low-level file system. After all, Hadoop and its model GFS both build on a conventional file system, creating their improvement on top of it, and the premise of ZFS is that the underlying file system should be redesigned to address the core issues.

I have seen production architectures built on ZFS, where the data storage requirements were very clear and well-defined and where storing data from multiple field sensors was considered better done with ZFS. The pros for ZFS in this case were: built-in replication, low overhead, and - given the right structured of records when written - build-in indexing for searching. Obviously, this was a very specific, though very fitting solution.

While other file system start out with the goal of improving on HDFS/GFS design, the advantage of HDFS is that it is very widely used. I think that in evaluating other file systems, the reader can be guided by the same considerations that led to the design of GFS: the designer of it analyzed prevalent file usage in the majority of their applications, and created a file system that optimized reliability for that particular type of usage. The reader may be well advised to compare the assumption of GFS designers with his or her case, and decide if HDFS fits the purpose, or if something else should be used in its place.

We should also note here that we compare Hadoop to other open-source storage solutions. There are proprietary and commercial solutions, but such comparison goes beyond the scope of this introduction.

1.3.2 Large database alternatives

The closest to HBase is Cassandra. While HBase is a near-clone of Google's BigTable, Cassandra purports to being a "BigTable/Dynamo hybrid". It can be said that while Cassandra's "writes-never-fail" emphasis has its advantages, HBase is the more robust database for a majority of use-cases. HBase being more prevalent in use, Cassandra faces an uphill battle - but it may be just what the reader needs.

Hypertable is another database close to Google's BigTable in features, and it claims to run 10 times faster than HBase. There is an ongoing discussion between HBase and Hypertable proponents, and the authors do not want to take sides in it, leaving the comparison to the reader. Like Cassandra, Hypertable has less users than HBase, and here too, the reader needs to evaluate the speed of Hypertable for his application, and weight it with other factors.

MongoDB (from "humongous") is a scalable, high-performance, open source, document-oriented database. Written in C++, MongoDB features document-oriented storage, full index on any attribute, replication and high availability, rich, document-based queries, and it works with MapReduce. If you are specifically processing documents and not arbitrary data, it is worth a look.

Other open-source and commercial databases that may be given consideration include Vertica with its SQL support and visualization, Cloudran for OLTP, and Spire.

In the end, before embarking on a development project, you will need to compare alternatives. Below is an example of such comparison. Please keep in mind that this is just possible point of view, and that the specifics of your project and of your view will be different. Therefore, the table below is mainly to encourage the reader to do a similar evaluation for his own needs.

Table 1.1 Comparison of Big Data databases

DB Pros/Cons	Hadoop/Hbase	Vertica	CloudTran	HyperTable
Pros	Key-based NoSQL, active user community, Cloudera support	Closed-source, SQL-standard, easy to use, visualization tools, complex queries	Closed-source optimized online transaction processing	Drop-in replacement for HBase, open-source
Cons	Steeper learning curve, less tools, simpler queries	Vendor lock-in, price, RDMS/BI - may not fit every application	Vendor lock-in, price, transaction-optimized, may not fit every application, needs wider adoption	New, needs user adoption and more testing
Notes	Good for new, long-term development	Good for existing SQL-based applications that needs fast scaling	Arguably the best OLTP	To be kept in mind as a possible alternative

1.3.3 Alternatives for distributed massive computations

Here too, depending upon the type of application that the reader needs, other approaches make prove more useful or more fitting to the purpose.

The first such example is the JavaSpaces paradigm. JavaSpaces is a giant hash map container. It provides the framework for building large-scale systems with multiple cooperating computational nodes. The framework is thread-safe and fault-tolerant. Many computer working on the same problem can store their data in a JavaSpaces container. When a node wants to do some work, it finds the data in the container, check its out, works on it, and then returns it back. The framework provides for atomicity. While the node is working on the data, other nodes cannot see it. If it fails, its lease on the data expires, and the data is returned back to the pool of data for processing.

The champion of JavaSpaces is a commercial company called GigaSpaces. The license for a JavaSpaces container from GigaSpaces is free - provided that you can

fit into the memory of one computer. Beyond that, GigaSpaces has implemented unlimited JavaSpaces container where multiple servers combine their memories into a shared pool. GigaSpaces has created a big sets of additional functionality for building large distributed systems. So again, everything depends on the reader's particular situation.

GridGain is another Hadoop alternative. The proponents of GridGain claim that while Hadoop is a compute grid and a data grid, GridGain is just a compute grid, so if your data requirements are not huge, why bother? They also say that seems to be enormously simpler to use. Study of the tools and prototyping with them can give one a good feel for the most fitting answer.

Terracotta is a commercial open source company, and in the open source realm it provides Java big cache and a number of other components for building large distributed systems. One of its advantages is that it allows to scale existing applications without a significant rewrite. By now we have gotten pretty far away from Hadoop, which proves that we have achieved our goal - give a reader a quick overview of various alternatives for building large distributed systems. Success in whichever way the reader chooses to go!

1.3.4 Arguments for Hadoop

We have given the pro arguments for the Hadoop alternatives, but now we can put in a word for the little elephant and its zoo. It boast wide adoption, has an active community, and has been in production use in many large companies. I think that before embarking on an exciting journey of building large distributed systems, the reader will do well to view the presentation by Jeff Dean, a Google Fellow, on the "Design, Lessons, and Advice from Building Large Distributed Systems" found on SlideShare: http://www.slideshare.net/yarapavan/largescale-distributed-systems-at-google-current-sy Google has built multiple applications on GFS, MapReduce, and BigTable, which are all implemented as open-source projects in the Hadoop zoo. According to Jeff, the plan is continue with 1,000,000 to 10,000,000 machines spread at 100s to 1000s locations around the world, and as arguments go, that is pretty big.

1.4 Say "Hi!" to Hadoop

Enough words, let’s look at some code! First, however, let us explain how MapReduce works in human terms.

1.4.1 A dialog between you and Hadoop

Imagine you want to count word frequencies in a text. It may be a book or a document, and word frequencies may tell you something about its subject. Or it may be a web access log, and you may be looking for suspicious activity. It may be a log of any customer activity, and you might be looking for most frequent customers, and so on.

In a straightforward approach, you would create an array or a hash table of words, and start counting the word's occurrences. You may run out of memory, or the process may be slow. If you try to use multiple computers all accessing shared memory, the system immediately gets complicated, with multi-threading, and we have not even thought of hardware failures. Anyone who has gone through similar exercises know how quickly a simple task can become a nightmare.

Enter Hadoop and offers its services. The following dialog ensues.

Hadoop : How can I help?

You : I need to count words in a document.

Hadoop : I will read the words and give them to you, one at a time, can you count that?

You : Yes, I will assign a count of 1 to each and give them back to you.

Hadoop : Very good. I will sort them, and will give them back to you in groups, grouping the same words together. Can you count that?

You : Yes, I will go through them and give you back each word and its count.

Hadoop : I will record each word with its count, and we'll be done.

I am not pulling your leg: it is that simple. That is the essence of a MapReduce job. Hadoop uses the cluster of computers (nodes), where each node reads words in parallel with all others (Map), then the nodes collect the results (Reduce) and write them back. Notice that there is a sort step, which is essential to the solution, and is provided for you - regardless of the size of the data. It may take place all in memory, or it may spill to disk. If any of the computers go bad, their tasks are assigned to the remaining healthy ones.

How does this dialog look in the code?

1.4.2 Geek talk

Hadoop : How can I help?

Listing 1.1 Hadoop

```
public class WordCount extends Configured implements Tool {
```

```
  public int run(String[] args) throws Exception {
```

You : I need to count words in a document.

Listing 1.2 You

```
Job job = new Job(getConf());
job.setJarByClass(WordCount.class);
job.setJobName("wordcount");
```

Hadoop : I will read the words and give them to you, one at a time, can you count that?

Listing 1.3 Hadoop

```
public static class Map extends Mapper <LongWritable, Text, Text, IntWritable> {
  public void map(LongWritable key, Text value, Context context)
    throws IOException, InterruptedException {
    String line = value.toString();
```

You : Yes, I will assign a count of 1 to each and give them back to you, Hadoop.

Listing 1.4 You

```
String [] tokens = line.split(" ,");
for (String token: tokens) {
  Text word = new Text();
  word.set(token);
  context.write(word, new IntWritable(1));
}
```

You have done more than you promised - you can process multiple words on the same line, if Hadoop choses to give them to you. This follows the principles of defensive programming. Then you immediately realize that each input line can be as long as it wants. In fact, Hadoop is optimized to have the best overall throughput on large datasets. Therefore, each input can be a complete document, and you are counting word frequencies in documents. If the documents come from the Web, for example, you already have the scale of computations needed for such tasks.

Hadoop : Very good. I will sort them, and will give them back to you in groups,

in which all words are the same. Can you count that?

Listing 1.5 Hadoop

```
public static class Reduce extends Reducer<Text, IntWritable, Text, IntWritable> {
  @Override public void reduce(Text key, Iterable<IntWritable> values, Context conte
    throws IOException, InterruptedException {
```

You : Yes, I will go through them and give you back each word and its count

Listing 1.6 You

```
int sum = 0;
for (IntWritable val : values) {
  sum += val.get();
}
context.write(key, new IntWritable(sum));
```

Hadoop : I will record each word with its count, and we'll be done.

Listing 1.7 Hadoop

```
// This is invisible - no code
// but you can trust that he does it
```

1.4.3 Let me see, who is Map and who is Reduce

MapReduce (or MR, if you want to impress your friends) has mappers and reducers, as can be seen by their names. What are they?

Mapper is the code that you supply to process one entry. This entry can be a line from a book or from a log, a temperature or financial record, etc. In our example it was counting to 1. In a different use case, it may be a name of an archive file, which the Mapper will unpack and process.

When the Mapper is done processing, it returns the results to the framework. The return takes the form of a Map, which consists of a key and a value. In our case, the key was the word. It can be a hash of a file, or any other important characteristic of your value. The keys that are the same will be combined, so you select them in such a way that elements that you want to be processed together will have the same key.

Finally, your Mapper code gives the Map to the framework. This is called emitting the map. It is expressed by the following code line

Listing 1.8 Emitting the map

```
context.write(key, value);
```

Now the framework sorts your maps. Sorting is an interesting process and it occurs a lot in computer processing, so we will talk in more detail about it in the next chapter. Having sorted the maps, the framework gives them back to you in groups. You supply the code which tells it how to process each group. This is the Reducer. The key that you gave to the framework is the same key that it returns to your Reducer. In our case, it was the word found in a document.

In the code, this is how we went through a group of values:

Listing 1.9 Going through values in the reducer

```
int sum = 0;
for (IntWritable val : values) {
  sum += val.get();
}
```

While the key was now the word, the value was count - which, as you may remember, we have set to 1 for each word. These ones are being summed up.

Finally, you return the reduced result to the framework, and it outputs results to a file.

Listing 1.10 Reducer emits the final map

```
context.write(key, new IntWritable(sum));
```

1.4.4 All of the code

Now let's look at the complete code of your first MapReduce program. Although it is the "Hello World" type of program, you can take it, adjust your Mapper and Reducer to do real processing, and you have a real application. This is especially true since we have prepared all the code that goes with the book as projects, both for Eclipse and for NetBeans. The reader is thus free to download the projects for his or her favorite IDE, and use it as a template. We have followed this approach with every example in the book.

Listing 1.11 Complete WordCount program

```
package com.shmsoft.hadoopinpractice;

import java.io.IOException;
import org.apache.hadoop.conf.Configured;
import org.apache.hadoop.fs.Path;
import org.apache.hadoop.io.IntWritable;
import org.apache.hadoop.io.LongWritable;
import org.apache.hadoop.io.Text;
import org.apache.hadoop.mapreduce.Job;
import org.apache.hadoop.mapreduce.Mapper;
import org.apache.hadoop.mapreduce.Reducer;
import org.apache.hadoop.mapreduce.lib.input.FileInputFormat;
import org.apache.hadoop.mapreduce.lib.input.TextInputFormat;
import org.apache.hadoop.mapreduce.lib.output.FileOutputFormat;
import org.apache.hadoop.mapreduce.lib.output.TextOutputFormat;
import org.apache.hadoop.util.Tool;
import org.apache.hadoop.util.ToolRunner;

public class WordCountExample extends Configured implements Tool {

  private static IntWritable ONE = new IntWritable(1);

  @Override
  public int run(String[] args) throws Exception {
    String testData = "test-data/moby-dick.txt";
    String outputPath = "test-output";
    Job job = new Job(getConf());
    job.setJarByClass(WordCountExample.class);
    job.setJobName("WordCountExample");

    job.setOutputKeyClass(Text.class);
    job.setOutputValueClass(IntWritable.class);

    job.setMapperClass(Map.class);
    job.setCombinerClass(Reduce.class);
    job.setReducerClass(Reduce.class);

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

1 **Best practice - avoid unnecessary object creation in a loop**

2 **The jar that contains your mapper and reduce code**

```
    FileInputFormat.setInputPaths(job, new Path(testData));
    FileOutputFormat.setOutputPath(job, new Path(outputPath));

    boolean success = job.waitForCompletion(true);
    return success ? 0 : 1;
  }

  public static void main(String[] args) throws Exception {
    int ret = ToolRunner.run(new WordCountExample(), args);
    System.exit(ret);
  }

  public static class Map extends Mapper<LongWritable, Text, Text, IntWritable> {

    @Override
    public void map(LongWritable key, Text value, Context context)
            throws IOException, InterruptedException {
      String line = value.toString();
      String[] words = line.split("\W");
      for (String word : words) {
        if (word.trim().length() > 0) {
          Text text = new Text();
          text.set(word);
          context.write(text, ONE);
        }
      }
    }
  }

  public static class Reduce extends Reducer<Text, IntWritable, Text, IntWritable> {

    @Override
    public void reduce(Text key, Iterable<IntWritable> values, Context context)
            throws IOException, InterruptedException {
      int sum = 0;
      for (IntWritable val : values) {
        sum += val.get();
      }
      context.write(key, new IntWritable(sum));
    }
  }
}
```

3 **Break the line of text into words**

4 **Send this word with the count of 1 to the reducer**

5 **Count all occurrences of the same word**

6 **Record the resulting count**

After you run the program, and apply it to the complete text of Moby Dick's - that is what we have used for testing and included in our sample project - your output will look something like this segment:

```
A 168
ABOUT 1
ACCOUNT 2
ACTUAL 1
ADDITIONAL 1
ADVANCING 2
```

```
ADVENTURES 1
AFFGHANISTAN 1
AFRICA 1
AFTER 1
AGAINST 1
AGREE 2
AGREEMENT 1
AHAB 10
AK 1
ALFRED 1
ALGERINE 1
```

Running this example on your computer is the first lab at the end of the chapter, where we will also give you practical advice on the setup.

1.5 How Hadoop works inside

Having successfully gone through the first example, and perhaps even having compiled and run it, we can take a breather and look at some theory. Early on, programmers had little use or respect for theory, but that approach can only take you so far. For example, if a multi-threaded application is written without knowledge and a good understanding of concurrency issues, the creation will fail at random, unpredictable times, most often under load and on the verge of success. This is why we should welcome theory, and here it is.

1.5.1 Masters and Workers

Hadoop uses a simple Master-Worker organization to distribute work. The Master is that computer which assigns work to the slaves. There is only one Master, and if it fails, the job has to be repeated. However, a single computer failure is rare, so this risk is acceptable.

The Workers are those machines that do the actual work. The Master picks out idle Workers and assigns a portion of work to them. Should a Worker computer fail, the Master will take its unfinished work and assign it to another Worker. This is done by the Master periodically pinging all Workers. If a Worker fails to respond, it is marked as having failed, and its work portion is returned back to the pool of available work, and later picked up by another Worker.

1.5.2 No data changed in place

Now you begin to appreciate that the data is read in one place and output to the other, but is never overwritten. Had the Worker been writing the data back, it would be impossible to restore it to the state that it was in before the work has started. Then in case of failure, we would have no way to pick up where the failed Worker left off.

Instead, no data is ever changed in place. This idea is called *functional programming* , and to us it simply means that the input and the output data reside in different places.

1.5.3 Splits

Hadoop divides all input data into chunks, also called splits, or more precisely, input splits. The input data comes to Hadoop in files, and these files usually reside in HDFS, which is able to hold any amount of input data, and more importantly, to make that data equally available to any of the computing nodes.

The files themselves can be anything: line-based log files, multi-line input records, or binary files. So how do you tell Hadoop what format is your input data in? You use the InputFormat class, like this:

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

In the example above, we have used the `TextInputFormat`, predefined in the Hadoop API, which tells Hadoop to split the input line by line - and, by the way, to output the results also line by line. When this format is used, the key that your Mapper gets is the byte offset of the line, and the value is the line contents.

But what if your lines already contain the key and the value. For example, your lines can be in the form

Time-of-the-measurement Tab Data-on-resource-utilization

and your task may be to collect the data by each hour and group it by client, so that you can prepare invoices for your multiple clients. In that case, you may want to use KeyValueInputFormat. This format will take each of your input lines and treat its first field, marked by a tab, as the key, and will give you the rest of the line as the value. That's exactly what you want in this case. Your Mapper will then have

to convert the input key into the hour, and parse and prepare resource utilization data for the future use by your Reducer. Incidentally, there is a better way to solve this problem with Hive, and we'll talk about it later.

The default size of a split chunk is 64MB - which is the same as the default size of the HDFS block. However, you can control this either in the Hadoop configuration file, or on the job-by-job level. Let us say you want to read only 50 characters from a line at a time. In both cases you do this with `Configuration.setInt` call, but the exact shape of the call depends on the Hadoop version.

```
In V 0.20: Configuration.setInt("mapred.linerecordreader.maxlength", 50)
In V 0.21: Configuration.setInt("mapreduce.input.linerecordreader.line.maxlength", 5
```

1.5.4 Shuffle and Sort

MapReduce promises you that when the maps (keys and values) arrive to your Reducer, each reducer sees them sorted by keys. As a programmer, that's all you need to know, and it will indeed come handy for some of the exercises, but as a software engineer, you may want to know a little more.

As an engineer, you are interested not only in the program correctness, but also in the overall performance. I know that this distinction between a programmer and an engineer is not great, but we need to denote the different levels of understanding somehow. The process which makes sure that all the maps are sorted by keys is called "shuffle". Your Mappers do some sorting, and the class responsible for this is called Combiner. Furthermore, Hadoop makes all maps come together and get sorted in the shuffle stage. If all maps can fit in memory - great, but if not, they will be "spilled" to the hard drives and Hadoop will sort them there. What is the practical outcome of this talk about shuffling? If you need to improve performance, or do some specialized sorting, you look into shuffling. We will look into this immediately in the next chapter, but for many applications the defaults will work just right.

The inner working of Hadoop is summarized in the figure below.

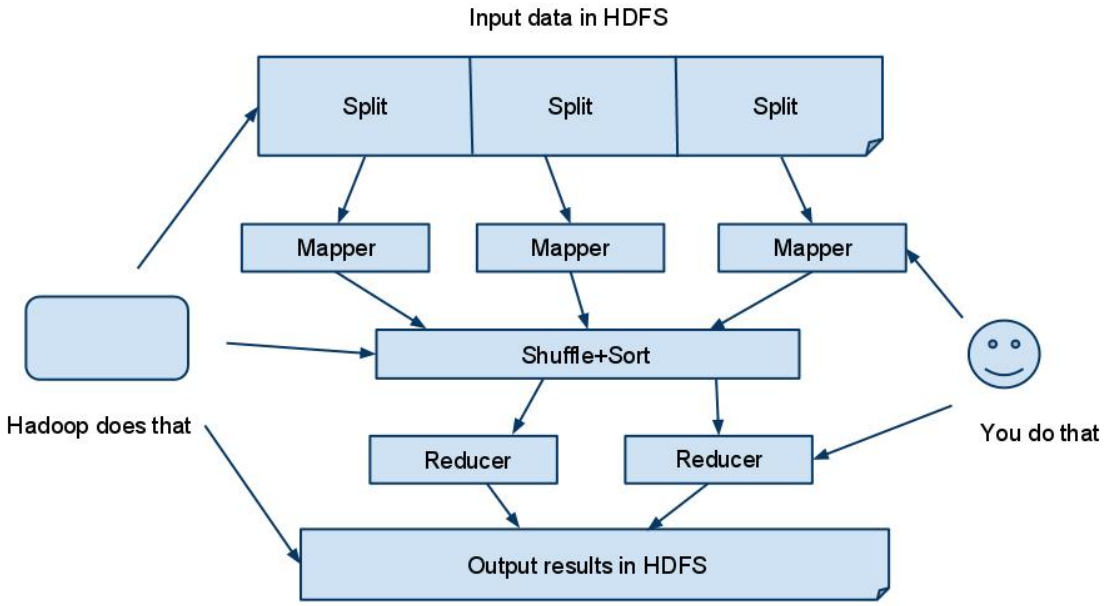

Figure 1.3 The inner working of Hadoop

1.6 Logging

Hadoop writes a number of logs, which includes those written by Hadoop daemon processes, HDFS, MapReduce tasks, and more. The structure of these logs for the Cloudera distribution is described in this blog post: `http://www.cloudera.com/blog/2009/09/apache-hadoop-log-file`

The logs most convenient for the code developer are the standard errors logs. They are created by each tasktracker (the part of Hadoop responsible for running MapReduce tasks) and are thus found on each node. They are place in this directory: `/var/log/hadoop/userlogs` and are named as follows:

`/var/log/hadoop/userlogs/attempt_job-id_map-or-reduce_att`

where `job-id` is the ID of the job that this attempt is doing work for, `map-or-reduce` is either "m" if the task attempt was a mapper, or “r” if the task attempt was a reducer, and `attempt-id` is the ID of the task attempt.

These logs are rotated according to the `mapred.userlog.retain.hours property`. You can clear these logs periodically without affecting Hadoop. You write to these logs by using `System.err.println("your`

`information")`.

Alternatively, you can write your debug information to standard out with `System.out.println("your information")`. These logs are very similar to the standard error logs.

You can also use log4j write, but these logs include some Hadoop internal diagnostic info. Log location may change with Hadoop and Cloudera releases, so check the documentation, but the above guidelines stay the same.

1.7 Rolling up your sleeves

With this additional information you are now sufficiently armed for doing your own development. One cannot overestimate the importance of the doing. What is more important, theory or practice? - Theory, because it leads to practice! How often, after reading on a subject and feeling that "all is clear," have you been stopped in your tracks by the question "what's the first line?" Ideally, you should to write the code in the techniques, or at least read it and run it.

A word of encouragement. Think of it as a weekend project. None of this is particularly hard. Don't sweat the small stuff though, search on the Internet, ask on the Hadoop discussion group, core-user@hadoop.apache.org, or on the book's forum.

TECHNIQUE 1: Setting up your development environment

There are many ways of setting up your environment, and if you set it up in the way that fits you best, you will be most productive.

Problem

Configure the Hadoop environment and run the first Hadoop project, WordCount, in your IDE, in stand-alone mode.

Solution

We have prepared the complete projects ready-to-run on GitHub (just find HadoopInPracticeBook) there, so it might seem trivial - but it may not be. The operating system, IDE version and Hadoop version may give you some trouble. Still, there is really no more basic task to start with than this.

1.7.1 For Windows programmers

We do realize that there are some programmers in the world who run Windows. You can still learn and use Hadoop. Here are a few possible roads that you can take: install Ubuntu alongside with Windows, install it as a double-boot, or install Fedora. Hadoop recommends Linux, and Cloudera provides packages for Debian and RedHat Linux. If you prefer to stay closer to Windows, use Cygwin, following the instructions on the Apache Hadoop site.

1.7.2 Running WordCount in your IDE

Discussion

Which Hadoop distribution should you use? The first two choices are Apache Hadoop and the Cloudera CDH. What are the pros and cons of each?

Apache Hadoop is the home of the Hadoop project. You can get any version there. Of course, if you a real programmer and don't have a production environment to support, you will choose the latest. That's fine - you can download and install the latest, usually in your local account, and accept the default settings and permission. That is the easiest way to get up and running with Apache Hadoop. If you want to install for more than yourself on that system, you would have to follow the best installation practices and learn about the preferred groups and permission.

You always deal with the latest and greatest code, and you can even try to build it yourself, if you adventurous enough you can make an improvement and suggest it to the committers, in the form of a HADOOP-PATH-XXX - you can learn about it here: http://wiki.apache.org/hadoop/HowToContribute

Your other choice is Cloudera CDH distribution. It offers the same code, but already packaged and tested, for many Linux flavors. If you go this route, you automatically get introduced to the best deployment practices, everything gets put into the right directories, and your Hadoop commands work from the command line out of the box, because they are put on the path by the install process. You don't even have to set the environmental variable yourself.

Cloudera CDH distribution offers other advantages: it puts in the patches that Cloudera tests, and it installs other Hadoop-related projects as packages. It may lag one version behind the current Apache Hadoop, so if you absolutely need the latest features, you will go with Apache.

Which choice is the better one? It almost does not matter, it will work either way. Most of my friends consultants tell me that they recommend Cloudera CDH

distribution to the clients, and I guess for two reasons: they will have less installation questions from the clients, and many commercial companies may prefer to know that there is commercial support for the open source Hadoop. Compare this to RedHat, and Cloudera is not shying away from this comparison either.

This covers the install issues. You need it before you are to go to the next exercise. Therefore, install Hadoop, download the code from GitHub, create the project in your IDE, and run it. You cannot do other labs before you do it.

Note that running from the IDE, you essentially ran your job with the following command:

```
java -jar dist/Chapter1.jar test-data/moby-dick.txt test-output
```

The jar that you give on the command line is the jar that contains your main function, used to submit your Hadoop job. The other jar, which you set with the call to `job.setJarByClass()`, is the one that contains your mapper and reducer code. It is this this second jar that Hadoop will copy to every node and run it there, following the rule of "move computations close to the data, not the data close to the computations." In this lab, one jar contains all the code.

Running inside of the IDE relies on the Hadoop jars being present. The command `java -jar` above worked, because the Hadoop jars were copied to `dist/lib`, and the Chapter1.jar pointed to them. In a more general case, you will need to be cognizant of packaging the required jars. This will be covered in a later chapter dealing with best practices.

1.7.3 TECHNIQUE 2: run WordCount outside of your IDE

Problem

After you were successful in running a MapReduce job from the IDE, it is time to to launch it on a Hadoop cluster, even if it is configured only in pseudo-distributed mode.

Solution

Now that sounds pretty simple. Build the jar in your IDE, then run it from the command line which will looks something like this:

```
hadoop jar your-jar parameters
```

When you were running your WordCountExample from the IDE, the code picked up the data from the local file system. This is very useful for debugging, but it will not work when running under hadoop, even in local mode on one machine,

because the data needs to reside in HDFS. Let's do it right from the beginning and copy the data to where it should be:

```
hadoop fs -mkdir /chapter1
hadoop fs -copyFromLocal moby-dick.txt /chapter1
See it here: http://localhost:50070/
```

Now we can run it with the following command:

Listing 1.12 run_wordcount_local.sh

```
hadoop jar ../dist/Chapter1.jar \
hdfs://localhost/chapter1/test-data/moby-dick.txt \
hdfs://localhost/chapter1/test-output
```

After the program runs, it is instructive and pleasant to view the output in the browser:

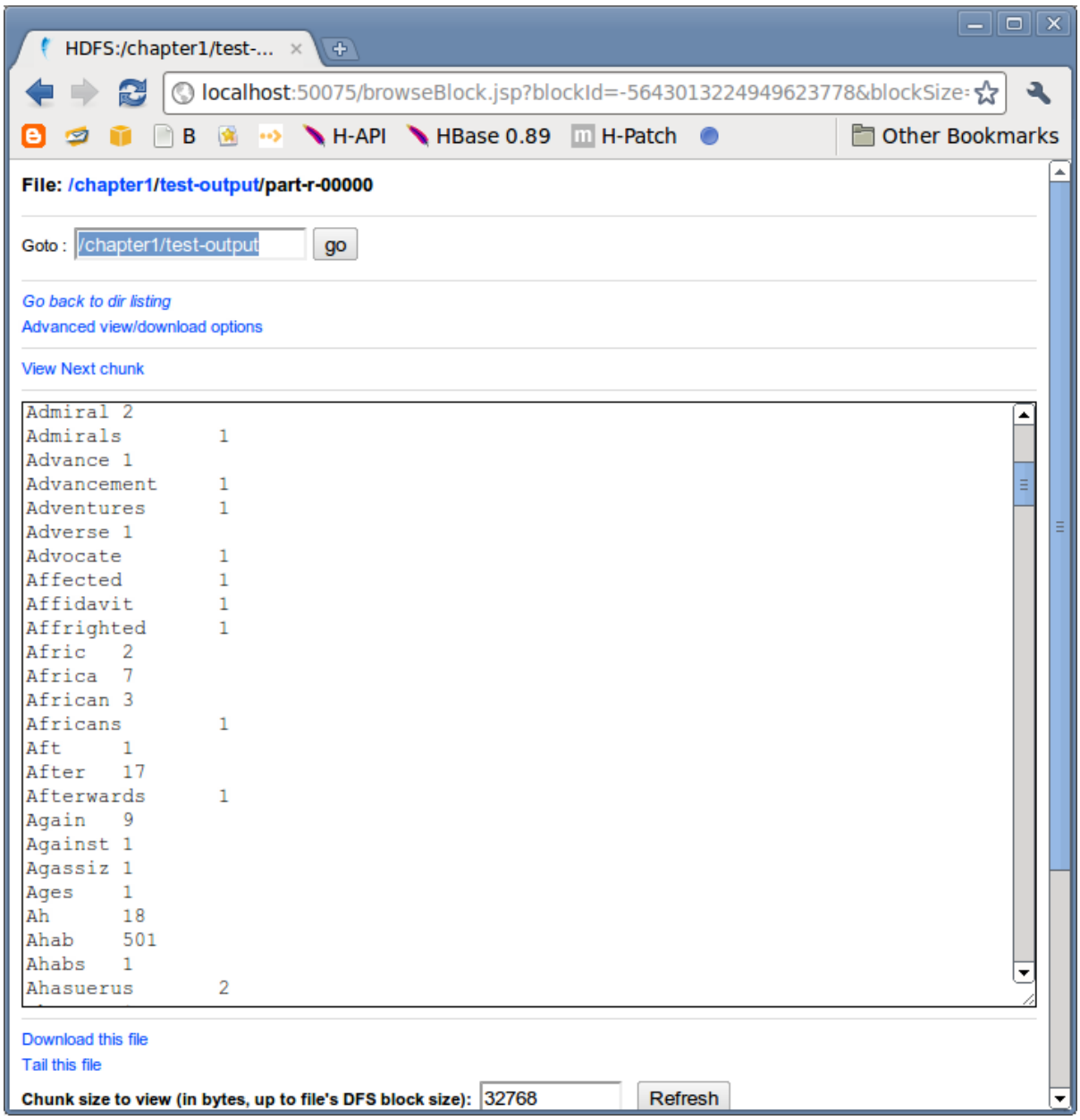

Figure 1.4 Output in HDFS viewed in the browser

Discussion

What happened to the Hadoop jars that we needed in Lab 1? Since we are now running relying on Hadoop, it takes care to provide your code with its library jars, and we do not need to care about them explicitely. This is true, provided that `HADOOP_CLASSPATH` configured correctly, and usually this is true if the install and run scripts are configured right. If not, you may need to adjust your installation.

1.7.4 Lab 3: configure distributed Hadoop cluster

Configure a minimal cluster of 2-3 nodes and run the WordCountExample there. Make sure that the tasks get distributed to different nodes. Verify this with Hadoop logging.

When you follow the instructions, using Cloudera or Apache Hadoop distribution, you should be able to see the HFDS in the browser, like in this figure for a 2-node cluster.

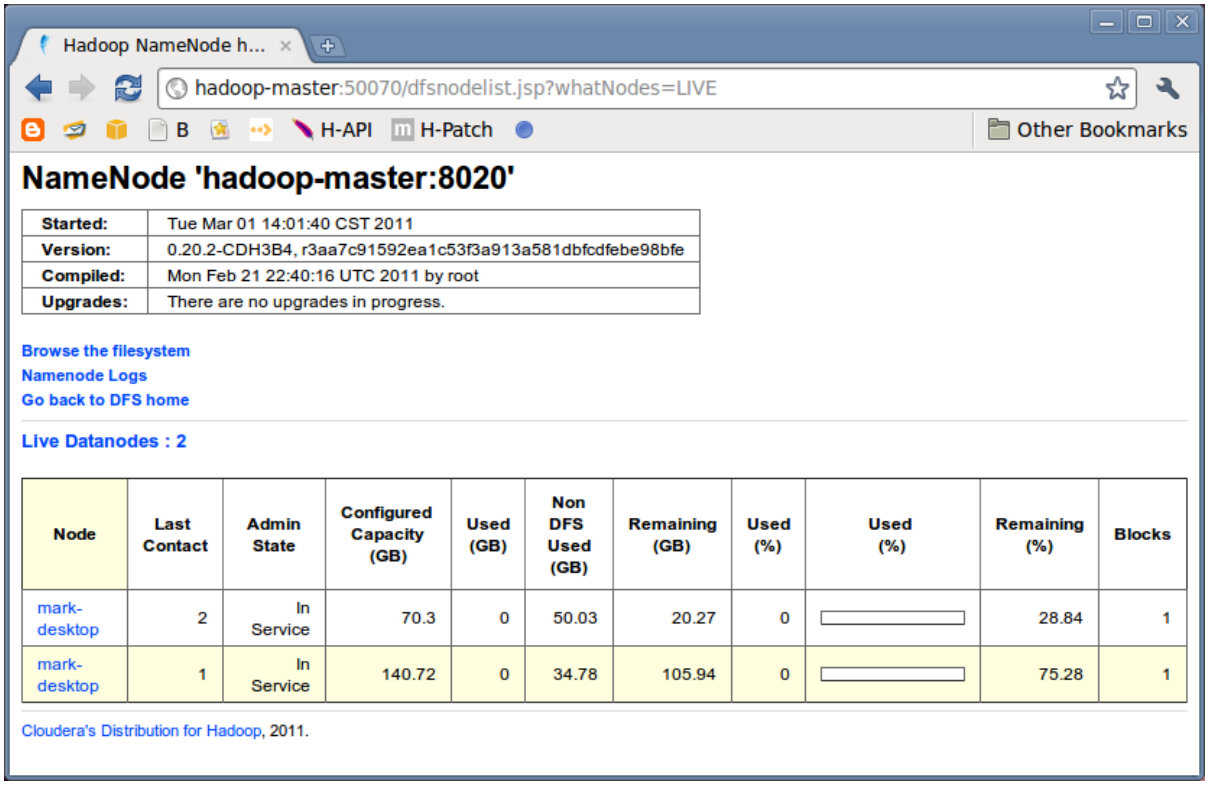

Figure 1.5 Browsing a 2-node HDFS

You would then run it with the following command:

Listing 1.13 run_wordcount_dist.sh

```
hadoop jar ../dist/Chapter1.jar \
hdfs://hadoop-master/chapter1/test-data/moby-dick.txt \
hdfs://hadoop-master/chapter1/test-output
```

1.7.5 Lab 4: customer billing (advanced)

Each line of your input contains the timestamp for an instance of resource utilization, then a tab, and customer-related data: customer ID, resource ID, and resource unit price. Write a MapReduce job that will create, for each customer, a summary of resource utilization by the hour, and output the result into a text file.

Sample input format:

Wed Jan 5 11:07:00 CST 2011 (Tab) Cust89347281 Res382915 $0.0035

Generate test data for the exercise 4 above. In keeping with the general Hadoop philosophy, manual testing is not enough. Write an MR task to generate arbitrary amount of random test data from pre-defined small invoice, then run your answer to the exercise 4 and see if you get the results you started out with.

1.7.6 Lab 5: Deduplication (advanced)

Often input files contain records that are the same. This may happen in web crawling, when individual crawlers may come to the same URL. Write a MapReduce task that will "dedupe" the records, and output each of the same records only once. Hint: in the Map stage compute the MD5 or SHA-1 hash of the record and output this as a key for the Reducer. In the reduce stage (since the records are sorted by keys) output only one of the records that are given to the Reducer with the same key value.

1.7.7 Lab 6: Write a distributed grep.

This lab is taken straight out of Google initial MapReduce paper.

1.8 Exercises for chapter 1

This chapter, as well as all succeeding ones, contains exercises building on the material in the chapter. Doing the labs with the help of the instructions in this must have been useful, but going solo is a special event in the life of every pilot, and we, the programmers, should imitate the best. Therefore, here are suggested exercises for your own practice and enjoyment.

1.8.1 Exercise 1

Check out the Hadoop and HDFS project code from the Subversion repository on Apache. The process of doing so is described here: `http://wiki.apache.org/hadoop/HowToContribute`. Try to build the project, read the code, apply a patch, and build again. There are a few benefit in doing this. You will feel more comfortable with Hadoop by following the famous advice "Read the code, stupid!" You will also have a feeling of what would be involve if you want to submit a patch yourself.

1.8.2 Exercise 2

Modify Lab 4 to output results to a relational database. Hint: using RDBS directly may be problematic because of the load, and the possibility of node failures, so use the output to text files instead, and load the text files into the database on a separate post-processing step.

1.9 Summary of chapter 1

In this chapter we were introduced to the MapReduce/Hadoop framework and wrote our first Hadoop program, which can actually accomplish quite a lot. We got a first look at when Hadoop can be useful. If you did the labs and exercises, you can now safely state that you are an intermediate Hadoop programmer, which is no small thing.

In the next chapter we will go a little deeper into sorting. There are situations where more control over sorting is required. It will also give you a better feeling for Hadoop internals, so that after reading it you will feel closer to a seasoned veteran than to a novice. Still, we will try to make it a breeze, keeping in line with the motto of this book, "Who said Hadoop is hard?".