Quick answers to common problems

Hadoop MapReduce Cookbook

Recipes for analyzing large and complex datasets with Hadoop MapReduce

Srinath Perera
Thilina Gunarathne

[PACKT] PUBLISHING open source* community experience distilled

Hadoop MapReduce Cookbook

Recipes for analyzing large and complex datasets with Hadoop MapReduce

Srinath Perera

Thilina Gunarathne

[PACKT] PUBLISHING open source community experience distilled

BIRMINGHAM - MUMBAI

Hadoop MapReduce Cookbook

First published: February 2013

Production Reference: 2250113

Published by Packt Publishing Ltd.
Livery Place
35 Livery Street
Birmingham B3 2PB, UK.

ISBN 978-1-84951-728-7

`www.packtpub.com`

Cover Image by J.Blaminsky (`milak6@wp.pl`)

Credits

Authors
Srinath Perera
Thilina Gunarathne

Reviewers
Masatake Iwasaki
Shinichi Yamashita

Acquisition Editor
Robin de Jongh

Lead Technical Editor
Arun Nadar

Technical Editors
Vrinda Amberkar
Dennis John
Dominic Pereira

Project Coordinator
Amey Sawant

Proofreader
Mario Cecere

Indexer
Hemangini Bari

Graphics
Valentina D'Silva

Production Coordinator
Arvindkumar Gupta

Cover Work
Arvindkumar Gupta

About the Authors

Srinath Perera is a Senior Software Architect at WSO2 Inc., where he overlooks the overall WSO2 platform architecture with the CTO. He also serves as a Research Scientist at Lanka Software Foundation and teaches as a visiting faculty at Department of Computer Science and Engineering, University of Moratuwa. He is a co-founder of Apache Axis2 open source project, and he has been involved with the Apache Web Service project since 2002, and is a member of Apache Software foundation and Apache Web Service project PMC. Srinath is also a committer of Apache open source projects Axis, Axis2, and Geronimo.

He received his Ph.D. and M.Sc. in Computer Sciences from Indiana University, Bloomington, USA and received his Bachelor of Science in Computer Science and Engineering from University of Moratuwa, Sri Lanka.

Srinath has authored many technical and peer reviewed research articles, and more detail can be found from his website. He is also a frequent speaker at technical venues.

He has worked with large-scale distributed systems for a long time. He closely works with Big Data technologies, such as Hadoop and Cassandra daily. He also teaches a parallel programming graduate class at University of Moratuwa, which is primarily based on Hadoop.

I would like to thank my wife Miyuru and my parents, whose never-ending support keeps me going. I also like to thanks Sanjiva from WSO2 who encourage us to make our mark even though project like these are not in the job description. Finally I would like to thank my colleges at WSO2 for ideas and companionship that have shaped the book in many ways.

Thilina Gunarathne is a Ph.D. candidate at the School of Informatics and Computing of Indiana University. He has extensive experience in using Apache Hadoop and related technologies for large-scale data intensive computations. His current work focuses on developing technologies to perform scalable and efficient large-scale data intensive computations on cloud environments.

Thilina has published many articles and peer reviewed research papers in the areas of distributed and parallel computing, including several papers on extending MapReduce model to perform efficient data mining and data analytics computations on clouds. Thilina is a regular presenter in both academic as well as industry settings.

Thilina has contributed to several open source projects at Apache Software Foundation as a committer and a PMC member since 2005. Before starting the graduate studies, Thilina worked as a Senior Software Engineer at WSO2 Inc., focusing on open source middleware development. Thilina received his B.Sc. in Computer Science and Engineering from University of Moratuwa, Sri Lanka, in 2006 and received his M.Sc. in Computer Science from Indiana University, Bloomington, in 2009. Thilina expects to receive his doctorate in the field of distributed and parallel computing in 2013.

This book would not have been a success without the direct and indirect help from many people. Thanks to my wife and my son for putting up with me for all the missing family times and for providing me with love and encouragement throughout the writing period. Thanks to my parents, without whose love, guidance and encouragement, I would not be where I am today.

Thanks to my advisor Prof. Geoffrey Fox for his excellent guidance and providing me with the environment to work on Hadoop and related technologies. Thanks to the HBase, Mahout, Pig, Hive, Nutch, and Lucene communities for developing great open source products. Thanks to Apache Software Foundation for fostering vibrant open source communities.

Thanks to the editorial staff at Packt, for providing me the opportunity to write this book and for providing feedback and guidance throughout the process. Thanks to the reviewers for reviewing this book, catching my mistakes, and for the many useful suggestions.

Thanks to all of my past and present mentors and teachers, including Dr. Sanjiva Weerawarana of WSO2, Prof. Dennis Gannon, Prof. Judy Qiu, Prof. Beth Plale, all my professors at Indiana University and University of Moratuwa for all the knowledge and guidance they gave me. Thanks to all my past and present colleagues for many insightful discussions and the knowledge they shared with me.

About the Reviewers

Masatake Iwasaki is Software Engineer at NTT DATA Corporation. He provides technical consultation for Open Source software such as Hadoop, HBase, and PostgreSQL.

Shinichi Yamashita is a Chief Engineer at OSS professional service unit in NTT DATA Corporation in Japan. He has more than seven years' experience in software and middleware (Apache, Tomcat, PostgreSQL, and Hadoop eco system) engineering. NTT DATA is your Innovation Partner anywhere around the world. It provides professional services from consulting, and system development to business IT outsourcing. In Japan, he has authored some books on Hadoop.

I thank my co-workers.

www.PacktPub.com

Support files, eBooks, discount offers and more

You might want to visit www.PacktPub.com for support files and downloads related to your book.

Did you know that Packt offers eBook versions of every book published, with PDF and ePub files available? You can upgrade to the eBook version at www.PacktPub.com and as a print book customer, you are entitled to a discount on the eBook copy. Get in touch with us at service@packtpub.com for more details.

At www.PacktPub.com, you can also read a collection of free technical articles, sign up for a range of free newsletters and receive exclusive discounts and offers on Packt books and eBooks.

PACKTLIB

http://PacktLib.PacktPub.com

Do you need instant solutions to your IT questions? PacktLib is Packt's online digital book library. Here, you can access, read and search across Packt's entire library of books.

Why Subscribe?

- Fully searchable across every book published by Packt
- Copy and paste, print and bookmark content
- On demand and accessible via web browser

Free Access for Packt account holders

If you have an account with Packt at www.PacktPub.com, you can use this to access PacktLib today and view nine entirely free books. Simply use your login credentials for immediate access.

Table of Contents

Preface

Hadoop MapReduce Cookbook helps readers learn to process large and complex datasets. The book starts in a simple manner, but still provides in-depth knowledge of Hadoop. It is a simple one-stop guide on how to get things done. It has 90 recipes, presented in a simple and straightforward manner, with step-by-step instructions and real world examples.

This product includes software developed at The Apache Software Foundation (`http://www.apache.org/`).

What this book covers

Chapter 1, *Getting Hadoop Up and Running in a Cluster*, explains how to install and run Hadoop both as a single node as well as a cluster.

Chapter 2, *Advanced HDFS*, introduces a set of advanced HDFS operations that would be useful when performing large-scale data processing with Hadoop MapReduce as well as with non-MapReduce use cases.

Chapter 3, *Advanced Hadoop MapReduce Administration*, explains how to change configurations and security of a Hadoop installation and how to debug.

Chapter 4, *Developing Complex Hadoop MapReduce Applications*, introduces you to several advanced Hadoop MapReduce features that will help you to develop highly customized, efficient MapReduce applications.

Chapter 5, *Hadoop Ecosystem*, introduces the other projects related to Hadoop such HBase, Hive, and Pig.

Chapter 6, *Analytics*, explains how to calculate basic analytics using Hadoop.

Chapter 7, *Searching and Indexing*, introduces you to several tools and techniques that you can use with Apache Hadoop to perform large-scale searching and indexing.

Chapter 8, *Classifications, Recommendations, and Finding Relationships*, explains how to implement complex algorithms such as classifications, recommendations, and finding relationships using Hadoop.

Chapter 9, *Mass Text Data Processing*, explains how to use Hadoop and Mahout to process large text datasets, and how to perform data preprocessing and loading operations using Hadoop.

Chapter 10, *Cloud Deployments: Using Hadoop on Clouds*, explains how to use Amazon Elastic MapReduce (EMR) and Apache Whirr to deploy and execute Hadoop MapReduce, Pig, Hive, and HBase computations on cloud infrastructures.

What you need for this book

All you need is access to a computer running Linux Operating system, and Internet. Also, Java knowledge is required.

Who this book is for

For big data enthusiasts and would be Hadoop programmers. The books for Java programmers who either have not worked with Hadoop at all, or who knows Hadoop and MapReduce but want to try out things and get into details. It is also a one-stop reference for most of your Hadoop tasks.

Conventions

In this book, you will find a number of styles of text that distinguish between different kinds of information. Here are some examples of these styles, and an explanation of their meaning.

Code words in text are shown as follows: "From this point onward, we shall call the unpacked Hadoop directory `HADOOP_HOME`."

A block of code is set as follows:

```
public void map(Object key, Text value, Context context)
  throws IOException, InterruptedException
  {
    StringTokenizer itr = new StringTokenizer(value.toString());
    while (itr.hasMoreTokens())
    {
      word.set(itr.nextToken());
      context.write(word, new IntWritable(1));
     }
  }
```

Any command-line input or output is written as follows:

```
>tar -zxvf hadoop-1.x.x.tar.gz
```

New terms and **important words** are shown in bold. Words that you see on the screen, in menus or dialog boxes for example, appear in the text like this: "Create a S3 bucket to upload the input data by clicking on **Create Bucket**".

> Warnings or important notes appear in a box like this.

> Tips and tricks appear like this.

Reader feedback

Feedback from our readers is always welcome. Let us know what you think about this book—what you liked or may have disliked. Reader feedback is important for us to develop titles that you really get the most out of.

To send us general feedback, simply send an e-mail to `feedback@packtpub.com`, and mention the book title via the subject of your message.

If there is a topic that you have expertise in and you are interested in either writing or contributing to a book, see our author guide on `www.packtpub.com/authors`.

Customer support

Now that you are the proud owner of a Packt book, we have a number of things to help you to get the most from your purchase.

Downloading the example code

You can download the example code files for all Packt books you have purchased from your account at `http://www.PacktPub.com`. If you purchased this book elsewhere, you can visit `http://www.PacktPub.com/support` and register to have the files e-mailed directly to you.

Errata

Although we have taken every care to ensure the accuracy of our content, mistakes do happen. If you find a mistake in one of our books—maybe a mistake in the text or the code—we would be grateful if you would report this to us. By doing so, you can save other readers from frustration and help us improve subsequent versions of this book. If you find any errata, please report them by visiting `http://www.packtpub.com/support`, selecting your book, clicking on the **errata submission form** link, and entering the details of your errata. Once your errata are verified, your submission will be accepted and the errata will be uploaded on our website, or added to any list of existing errata, under the Errata section of that title. Any existing errata can be viewed by selecting your title from `http://www.packtpub.com/support`.

Piracy

Piracy of copyright material on the Internet is an ongoing problem across all media. At Packt, we take the protection of our copyright and licenses very seriously. If you come across any illegal copies of our works, in any form, on the Internet, please provide us with the location address or website name immediately so that we can pursue a remedy.

Please contact us at `copyright@packtpub.com` with a link to the suspected pirated material.

We appreciate your help in protecting our authors, and our ability to bring you valuable content.

Questions

You can contact us at `questions@packtpub.com` if you are having a problem with any aspect of the book, and we will do our best to address it.

1

Getting Hadoop Up and Running in a Cluster

In this chapter, we will cover:

- Setting up Hadoop on your machine
- Writing the WordCount MapReduce sample, bundling it, and running it using standalone Hadoop
- Adding the combiner step to the WordCount MapReduce program
- Setting up HDFS
- Using the HDFS monitoring UI
- HDFS basic command-line file operations
- Setting Hadoop in a distributed cluster environment
- Running the WordCount program in a distributed cluster environment
- Using the MapReduce monitoring UI

Introduction

For many years, users who want to store and analyze data would store the data in a database and process it via SQL queries. The Web has changed most of the assumptions of this era. On the Web, the data is unstructured and large, and the databases can neither capture the data into a schema nor scale it to store and process it.

Google was one of the first organizations to face the problem, where they wanted to download the whole of the Internet and index it to support search queries. They built a framework for large-scale data processing borrowing from the "map" and "reduce" functions of the functional programming paradigm. They called the paradigm **MapReduce**.

Hadoop is the most widely known and widely used implementation of the MapReduce paradigm. This chapter introduces Hadoop, describes how to install Hadoop, and shows you how to run your first MapReduce job with Hadoop.

Hadoop installation consists of four types of nodes—a **NameNode**, **DataNodes**, a **JobTracker**, and **TaskTracker** HDFS nodes (NameNode and DataNodes) provide a distributed filesystem where the JobTracker manages the jobs and TaskTrackers run tasks that perform parts of the job. Users submit MapReduce jobs to the JobTracker, which runs each of the Map and Reduce parts of the initial job in TaskTrackers, collects results, and finally emits the results.

Hadoop provides three installation choices:

- **Local mode**: This is an unzip and run mode to get you started right away where all parts of Hadoop run within the same JVM
- **Pseudo distributed mode**: This mode will be run on different parts of Hadoop as different Java processors, but within a single machine
- **Distributed mode**: This is the real setup that spans multiple machines

We will discuss the local mode in the first three recipes, and Pseudo distributed and distributed modes in the last three recipes.

Setting up Hadoop on your machine

This recipe describes how to run Hadoop in the local mode.

Getting ready

Download and install Java 1.6 or higher version from `http://www.oracle.com/technetwork/java/javase/downloads/index.html`.

How to do it...

Now let us do the Hadoop installation:

1. Download the most recent Hadoop 1.0 branch distribution from `http://hadoop.apache.org/`.
2. Unzip the Hadoop distribution using the following command. You will have to change the `x.x` in the filename with the actual release you have downloaded. If you are using Windows, you should use your favorite archive program such as WinZip or WinRAR for extracting the distribution. From this point onward, we shall call the unpacked Hadoop directory `HADOOP_HOME`.

   ```
   >tar -zxvf hadoop-1.x.x.tar.gz
   ```

3. You can use Hadoop local mode after unzipping the distribution. Your installation is done. Now, you can run Hadoop jobs through `bin/hadoop` command, and we will elaborate that further in the next recipe.

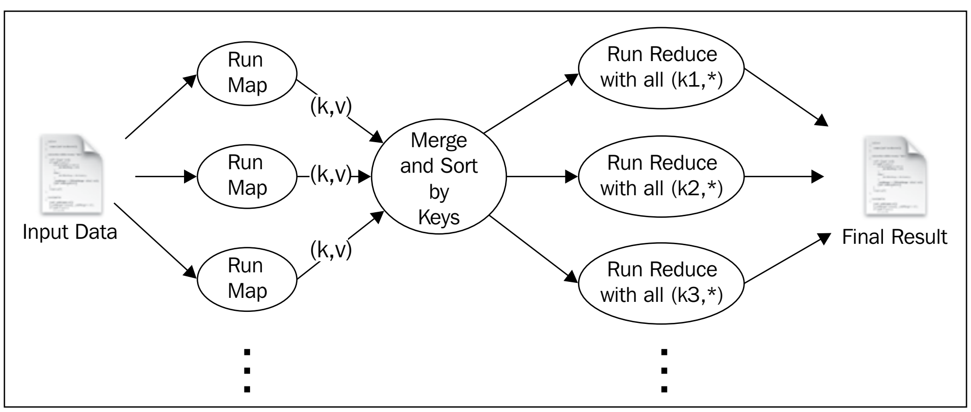
How it works...

Hadoop local mode does not start any servers but does all the work within the same JVM. When you submit a job to Hadoop in the local mode, that job starts a JVM to run the job, and that JVM carries out the job. The output and the behavior of the job is the same as a distributed Hadoop job, except for the fact that the job can only use the current node for running tasks. In the next recipe, we will discover how to run a MapReduce program using the unzipped Hadoop distribution.

> **Downloading the example code**
>
> You can download the example code files for all Packt books you have purchased from your account at `http://www.packtpub.com`. If you purchased this book elsewhere, you can visit `http://www.packtpub.com/support` and register to have the files e-mailed directly to you.

Writing a WordCount MapReduce sample, bundling it, and running it using standalone Hadoop

This recipe explains how to write a simple MapReduce program and how to execute it.



To run a MapReduce job, users should furnish a `map` function, a `reduce` function, input data, and an output data location. When executed, Hadoop carries out the following steps:

1. Hadoop breaks the input data into multiple data items by new lines and runs the `map` function once for each data item, giving the item as the input for the function. When executed, the `map` function outputs one or more key-value pairs.
2. Hadoop collects all the key-value pairs generated from the `map` function, sorts them by the key, and groups together the values with the same key.
3. For each distinct key, Hadoop runs the reduce function once while passing the key and list of values for that key as input.
4. The `reduce` function may output one or more key-value pairs, and Hadoop writes them to a file as the final result.

Getting ready

From the source code available with this book, select the source code for the first chapter, `chapter1_src.zip`. Then, set it up with your favorite **Java Integrated Development Environment** (**IDE**); for example, Eclipse. You need to add the `hadoop-core` JAR file in `HADOOP_HOME` and all other JAR files in the `HADOOP_HOME/lib` directory to the classpath of the IDE.

Download and install Apache Ant from `http://ant.apache.org/`.

How to do it...

Now let us write our first Hadoop MapReduce program.

1. The WordCount sample uses MapReduce to count the number of word occurrences within a set of input documents. Locate the sample code from `src/chapter1/Wordcount.java`. The code has three parts—mapper, reducer, and the main program.
2. The mapper extends from the `org.apache.hadoop.mapreduce.Mapper` interface. When Hadoop runs, it receives each new line in the input files as an input to the mapper. The `map` function breaks each line into substrings using whitespace characters such as the separator, and for each token (word) emits `(word,1)` as the output.

```
public void map(Object key, Text value, Context context
                    ) throws IOException, InterruptedException
{
      StringTokenizer itr = new StringTokenizer(value.toString());
       while (itr.hasMoreTokens())
```

```
        {
            word.set(itr.nextToken());
            context.write(word, new IntWritable(1));
        }
}
```

3. The `reduce` function receives all the values that have the same key as the input, and it outputs the key and the number of occurrences of the key as the output.

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

4. The `main` program puts the configuration together and submits the job to Hadoop.

```
Configuration conf = new Configuration();
String[] otherArgs = new GenericOptionsParser(conf, args).
getRemainingArgs();
if (otherArgs.length != 2) {
System.err.println("Usage: wordcount <in><out>");
System.exit(2);
}
Job job = new Job(conf, "word count");
job.setJarByClass(WordCount.class);
job.setMapperClass(TokenizerMapper.class);
//Uncomment this to
//job.setCombinerClass(IntSumReducer.class);
job.setReducerClass(IntSumReducer.class);
job.setOutputKeyClass(Text.class);
job.setOutputValueClass(IntWritable.class);
FileInputFormat.addInputPath(job, new Path(otherArgs[0]));
FileOutputFormat.setOutputPath(job, new Path(otherArgs[1]));
System.exit(job.waitForCompletion(true) ? 0 : 1);
```

5. You can compile the sample by running the following command, which uses Apache Ant, from the root directory of the sample code:

   ```
   >ant build
   ```

   If you have not done this already, you should install Apache Ant by following the instructions given at `http://ant.apache.org/manual/install.html`. Alternatively, you can use the compiled JAR file included with the source code.

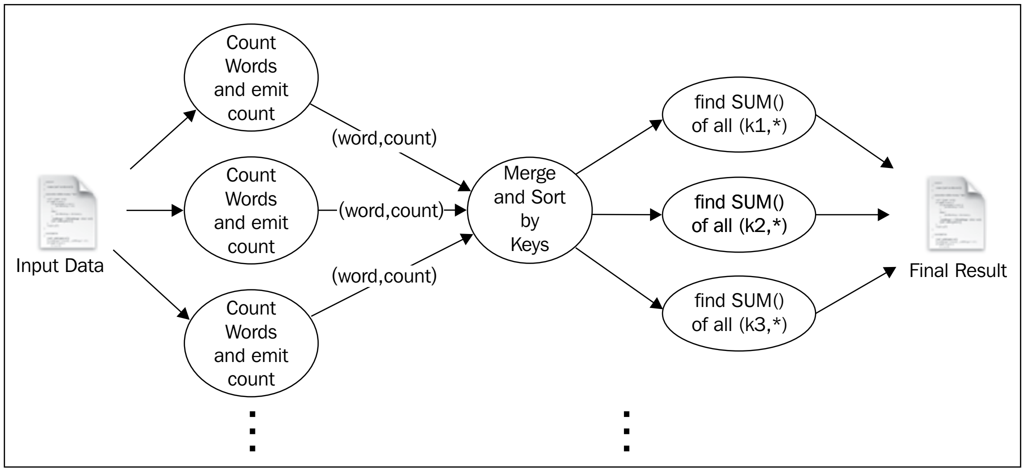
6. Change the directory to `HADOOP_HOME`, and copy the `hadoop-cookbook-chapter1.jar` file to the `HADOOP_HOME` directory. To be used as the input, create a directory called `input` under `HADOOP_HOME` and copy the `README.txt` file to the directory. Alternatively, you can copy any text file to the `input` directory.
7. Run the sample using the following command. Here, `chapter1.WordCount` is the name of the `main` class we need to run. When you have run the command, you will see the following terminal output:

   ```
   >bin/hadoop jar hadoop-cookbook-chapter1.jar chapter1.WordCount
   input output
   12/04/11 08:12:44 INFO input.FileInputFormat: Total input paths to
   process : 16
   12/04/11 08:12:45 INFO mapred.JobClient: Running job: job_
   local_0001
   12/04/11 08:12:45 INFO mapred.Task: Task:attempt_
   local_0001_m_000000_0 is done. And is in the process of commiting
   ...........
   .....
   12/04/11 08:13:37 INFO mapred.JobClient: Job complete: job_
   local_0001
   .....
   ```

8. The output directory will have a file named like `part-r-XXXXX`, which will have the count of each word in the document. Congratulations! You have successfully run your first MapReduce program.

How it works...

In the preceding sample, MapReduce worked in the local mode without starting any servers and using the local filesystem as the storage system for inputs, outputs, and working data. The following diagram shows what happened in the WordCount program under the covers:



The workflow is as follows:

1. Hadoop reads the input, breaks it by new line characters as the separator and then runs the map function passing each line as an argument.
2. The map function tokenizes the line, and for each token (word), emits a key value pair `(word, 1)`.
3. Hadoop collects all the `(word,1)` pairs, sorts them by the word, groups all the values emitted against each unique key, and invokes the reduce once for each unique key passing the key and values for that key as an argument.
4. The reduce function counts the number of occurrences of each word using the values and emits it as a key-value pair.
5. Hadoop writes the final output to the output directory.

There's more...

As an optional step, copy the `input` directory to the top level of the IDE-based project (Eclipse project) that you created for samples. Now you can run the `WordCount` class directly from your IDE passing `input output` as arguments. This will run the sample the same as before. Running MapReduce jobs from IDE in this manner is very useful for debugging your MapReduce jobs.

Although you ran the sample with Hadoop installed in your local machine, you can run it using distributed Hadoop cluster setup with a HDFS-distributed filesystem. The recipes of this chapter, *Setting up HDFS* and *Setting Hadoop in a distributed cluster environment* will discuss how to run this sample in a distributed setup.

Adding the combiner step to the WordCount MapReduce program

After running the `map` function, if there are many key-value pairs with the same key, Hadoop has to move all those values to the `reduce` function. This can incur a significant overhead. To optimize such scenarios, Hadoop supports a special function called **combiner**. If provided, Hadoop will call the combiner from the same node as the map node before invoking the reducer and after running the mapper. This can significantly reduce the amount of data transferred to the reduce step.

This recipe explains how to use the combiner with the WordCount sample introduced in the previous recipe.

How to do it...

Now let us run the MapReduce job adding the combiner:

1. Combiner must have the same interface as the `reduce` function. For the WordCount sample, we will reuse the `reduce` function as the combiner.
2. To ask the MapReduce job to use the combiner, let us uncomment the line `//job.setCombinerClass(IntSumReducer.class);` in the sample and recompile the code.
3. Copy the `hadoop-cookbook-chapter1.jar` file to the `HADOOP_HOME` directory and run the WordCount as done in the earlier recipe. Make sure to delete the old output directory before running the job.
4. Final results will be available from the `output` directory.

How it works...

To activate a combiner, users should provide a mapper, a reducer, and a combiner as input to the MapReduce job. In that setting, Hadoop executes the combiner in the same node as the mapper function just after running the mapper. With this method, the combiner can pre-process the data generated by the mapper before sending it to the reducer, thus reducing the amount of data that is getting transferred.

For example, with the WordCount, combiner receives `(word,1)` pairs from the map step as input and outputs a single `(word, N)` pair. For example, if an input document has 10,000 occurrences of word "the", the mapper will generate 10,000 `(the,1)` pairs, while the combiner will generate one `(the,10,000)` thus reducing the amount of data transferred to the reduce task.

However, the combiner only works with commutative and associative functions. For example, the same idea does not work when calculating mean. As mean is not communicative and associative, a combiner in that case will yield a wrong result.

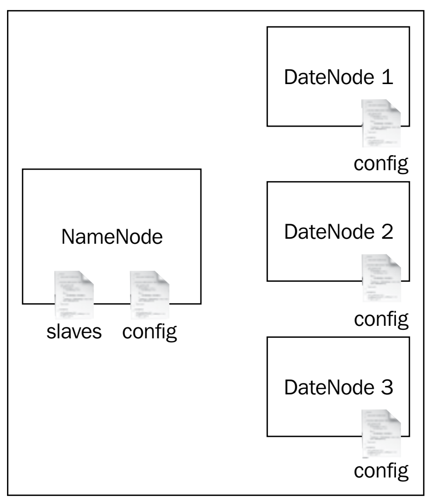
There's more...

Although in the sample we reused the `reduce` function implementation as the combiner function, you may write your own combiner function just like we did for the `map` and `reduce` functions in the previous recipe. However, the signature of the combiner function must be identical to that of the `reduce` function.

In a local setup, using a combiner will not yield significant gains. However, in the distributed setups as described in *Setting Hadoop in a distributed cluster environment* recipe, combiner can give significant gains.

Setting up HDFS

HDFS is the distributed filesystem that is available with Hadoop. MapReduce tasks use HDFS to read and write data. HDFS deployment includes a single NameNode and multiple DataNodes.



For the HDFS setup, we need to configure NameNodes and DataNodes, and then specify the DataNodes in the `slaves` file. When we start the NameNode, startup script will start the DataNodes.

Getting ready

You may follow this recipe either using a single machine or multiple machines. If you are using multiple machines, you should choose one machine as the master node where you will run the HDFS NameNode. If you are using a single machine, use it as both the NameNode as well as the DataNode.

1. Install Java in all machines that will be used to set up the HDFS cluster.
2. If you are using Windows machines, install Cygwin and SSH server in each machine. The link `http://pigtail.net/LRP/printsrv/cygwin-sshd.html` provides step-by-step instructions.

How to do it...

Now let us set up HDFS in the distributed mode.

1. Enable SSH from master nodes to slave nodes. Check that you can login to the localhost and all other nodes using SSH without a passphrase by running one of the following commands:
   - `>ssh localhost`
   - `>ssh IPaddress`
2. If the above command returns an error or asks for a password, create SSH keys by executing the following command:

   ```
   >ssh-keygen -t dsa -P '' -f ~/.ssh/id_dsa
   ```

   Move the `~/.ssh/id_dsa.pub` file to the all the nodes in the cluster. Then add the SSH keys to the `~/.ssh/authorized_keys` file in each node by running the following command (if the `authorized_keys` file does not exist, run the following command. Else, skip to the `cat` command):

   ```
   >touch ~/.ssh/authorized_keys && chmod 600 ~/.ssh/authorized_keys
   ```

   Now with permissions set, add your key to the `~/.ssh/authorized_keys` file.

   ```
   >cat ~/.ssh/id_dsa.pub >> ~/.ssh/authorized_keys
   ```

   Then you can log in with the following command:

   ```
   >ssh localhost
   ```

   This command creates an SSH key pair in the `.ssh/`directory of the home directory, and registers the generated public key with SSH as a trusted key.

3. In each machine, create a directory for storing HDFS data. Let's call that directory `HADOOP_DATA_DIR`. Now let us create two sub directories, `HADOOP_DATA_DIR/data` and `HADOOP_DATA_DIR/name`. Change the directory permissions to `755` by running the following command for each directory:

   ```
   >chmod 755 <name of dir>
   ```

4. In the NameNode, change directory to the unzipped `HADOOP_HOME` directory. Then place the IP address of all slave nodes in the `HADOOP_HOME/conf/slaves` file, each on a separate line. When we start the NameNode, it will use the `slaves` file to start the DataNodes.
5. In all machines, edit the `HADOOP_HOME/conf/hadoop-env.sh` file by uncommenting the `JAVA_HOME` line and pointing it to your local Java installation. For example, if Java is in `/opt/jdk1.6`, change the `JAVA_HOME` line to `export JAVA_HOME=/opt/jdk1.6`.
6. Inside each node's `HADOOP_HOME/conf` directory, add the following code to the `core-site.xml` and `hdfs-site.xml` files. Before adding the configurations, replace the `MASTER_NODE` strings with the IP address of the master node and `HADOOP_DATA_DIR` with the directory you created in the first step.

   `HADOOP_HOME/conf/core-site.xml`

   ```
   <configuration>
   <property>
   <name>fs.default.name</name>
   <!-- URL of MasterNode/NameNode -->
   <value>hdfs://MASTER_NODE:9000/</value>
   </property>
   </configuration>
   ```

   `HADOOP_HOME/conf/hdfs-site.xml`

   ```
   <configuration>
   <property>
   <name>dfs.name.dir</name>
   <!-- Path to store namespace and transaction logs -->
   <value>HADOOP_DATA_DIR/name</value>
   </property>
   <property>
   <name>dfs.data.dir</name>
   <!-- Path to store data blocks in datanode -->
   <value>HADOOP_DATA_DIR/data</value>
   </property>
   </configuration>
   ```

7. From the NameNode, run the following command to format a new filesystem:

```
>bin/hadoop namenode –format

12/04/09 08:44:50 INFO namenode.NameNode: STARTUP_MSG:
/************************************************************
…
12/04/09 08:44:51 INFO common.Storage: Storage directory /Users/
srinath/playground/hadoop-book/hadoop-temp/dfs/name has been
successfully formatted.
12/04/09 08:44:51 INFO namenode.NameNode: SHUTDOWN_MSG:
/************************************************************
SHUTDOWN_MSG: Shutting down NameNode at Srinath-s-MacBook-Pro.
local/172.16.91.1
************************************************************/
```

8. Start the HDFS setup with the following command:

```
>bin/start-dfs.sh
```

This command will first start a NameNode. It will then look at the `HADOOP_HOME/conf/slaves` file and start the DataNodes. It will print a message like the following to the console.

```
starting namenode, logging to /root/hadoop-setup-srinath/
hadoop-1.0.0/libexec/../logs/hadoop-root-namenode-node7.beta.out
209.126.198.72: starting datanode, logging to /root/hadoop-setup-
srinath/hadoop-1.0.0/libexec/../logs/hadoop-root-datanode-node7.
beta.out
209.126.198.71: starting datanode, logging to /root/hadoop-setup-
srinath/hadoop-1.0.0/libexec/../logs/hadoop-root-datanode-node6.
beta.out
209.126.198.72: starting secondarynamenode, logging to /root/
hadoop-setup-srinath/hadoop-1.0.0/libexec/../logs/hadoop-root-
secondarynamenode-node7.beta.out
```

Hadoop uses a centralized architecture for metadata. In this design, the NameNode holds the information of all the files and where the data blocks for each file are located. The NameNode is a single point of failure, and on failure it will stop all the operations of the HDFS cluster. To avoid this, Hadoop supports a secondary NameNode that will hold a copy of all data in NameNode. If the NameNode fails, the secondary NameNode takes its place.

9. Access the link `http://MASTER_NODE:50070/` and verify that you can see the HDFS startup page. Here, replace `MASTER_NODE` with the IP address of the master node running the HDFS NameNode.
10. Finally, shut down the HDFS cluster using the following command:

    ```
    >bin/stop-dfs.sh
    ```

How it works...

When started, the NameNode will read the `HADOOP_HOME/conf/slaves` files, find the DataNodes that need to be started, start them, and set up the HDFS cluster. In the *HDFS basic command line file operations* recipe, we will explore how to use HDFS to store and manage files.

HDFS setup is only a part of the Hadoop installation. The *Setting Hadoop in a distributed cluster environment* recipe describes how to set up the rest of the Hadoop.

Using HDFS monitoring UI

HDFS comes with a monitoring web console to verify the installation and monitor the HDFS cluster. It also lets users explore the content of the HDFS filesystem. In this recipe, we will look at how we can access the HDFS monitoring UI and verify the installation.

Getting ready

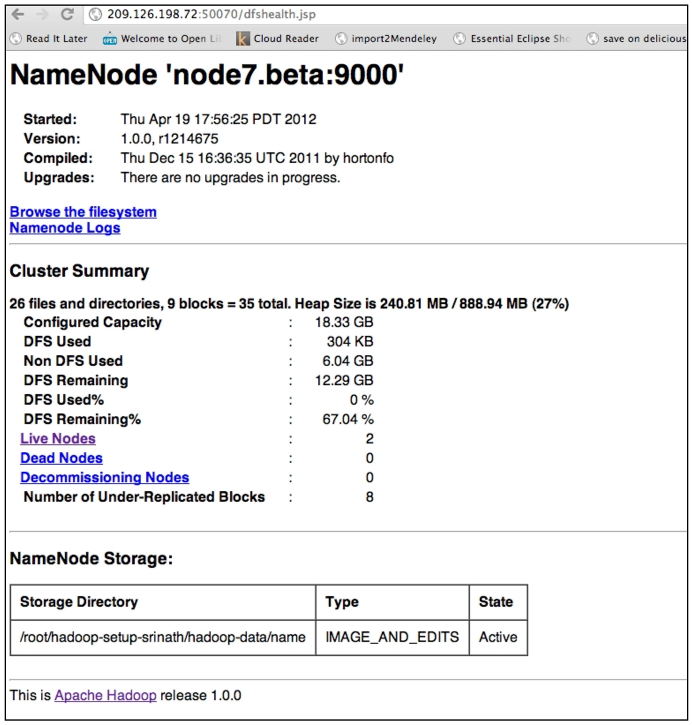
Start the HDFS cluster as described in the previous recipe.

How to do it...

Let us access the HDFS web console.

1. Access the link `http://MASTER_NODE:50070/` using your browser, and verify that you can see the HDFS startup page. Here, replace `MASTER_NODE` with the IP address of the master node running the HDFS NameNode.

2. The following screenshot shows the current status of the HDFS installation including the number of nodes, total storage, storage taken by each node. It also allows users to browse the HDFS filesystem.



HDFS basic command-line file operations

HDFS is a distributed filesystem, and just like a Unix filesystem, it allows users to manipulate the filesystem using shell commands. This recipe explains how to use the HDFS basic command line to execute those commands.

It is worth noting that HDFS commands have a one-to-one correspondence with Unix commands. For example, consider the following command:

```
>hadoop dfs -cat /data/foo.txt
```

The command reads the `/data/foo.txt` file and prints it to the screen, just like the `cat` command in Unix system.

Getting ready

Start the HDFS server by following the *Setting up HDFS* recipe.

How to do it...

1. Change the directory to `HADOOP_HOME`.
2. Run the following command to create a new directory called `/test`:

   >bin/hadoop dfs -mkdir /test

3. HDFS filesystem has `/` as the root directory just like the Unix filesystem. Run the following command to list the content of the HDFS root directory:

   >bin/hadoop dfs -ls /

4. Run the following command to copy the local readme file to /test

   >bin/hadoop dfs -put README.txt /test

5. Run the following command to list the `/test` directory:

   ```
   >bin/hadoop dfs -ls /test


   Found 1 items
   -rw-r--r--   1 srinath supergroup       1366 2012-04-10 07:06 /
   test/README.txt
   ```

6. Run the following command to copy the `/test/README.txt` to local directory:

   >bin/hadoop dfs -get /test/README.txt README-NEW.txt

How it works...

When a command is issued, the client will talk to the HDFS NameNode on the user's behalf and carry out the operation. Generally, we refer to a file or a folder using the path starting with `/`; for example, `/data`, and the client will pick up the NameNode from configurations in the `HADOOP_HOME/conf` directory.

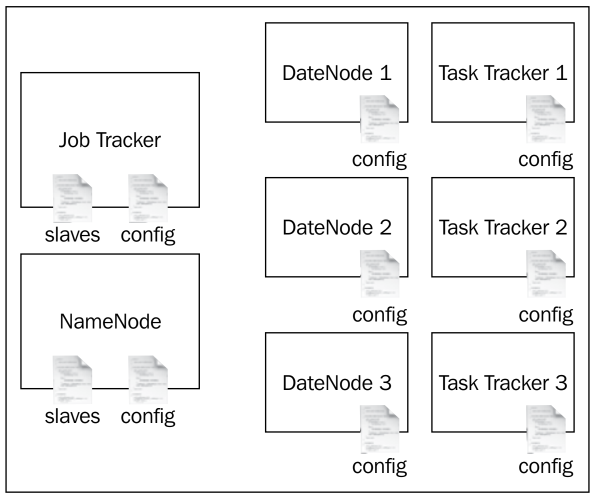
However, if needed, we can use a fully qualified path to force the client to talk to a specific NameNode. For example, `hdfs://bar.foo.com:9000/data` will ask the client to talk to NameNode running on `bar.foo.com` at the port `9000`.

There's more...

HDFS supports most of the Unix commands such as `cp`, `mv`, and `chown`, and they follow the same pattern as the commands discussed above. The document `http://hadoop.apache.org/docs/r1.0.3/file_system_shell.html` provides a list of all commands. We will use these commands throughout, in the recipes of the book.

Setting Hadoop in a distributed cluster environment

Hadoop deployment includes a HDFS deployment, a single job tracker, and multiple TaskTrackers. In the preceding recipe, *Setting up HDFS*, we discussed the HDFS deployment. For the Hadoop setup, we need to configure JobTrackers and TaskTrackers and then specify the TaskTrackers in the `HADOOP_HOME/conf/slaves` file. When we start the JobTracker, it will start the TaskTracker nodes. The following diagram illustrates a Hadoop deployment:



Getting ready

You may follow this recipe either using a single machine or multiple machines. If you are using multiple machines, you should choose one machine as the master node where you will run the HDFS NameNode and the JobTracker. If you are using a single machine, use it as both the master node as well as a slave node.

1. Install Java in all machines that will be used to set up Hadoop.
2. If you are using Windows machines, first install Cygwin and SSH server in each machine. The link `http://pigtail.net/LRP/printsrv/cygwin-sshd.html` provides step-by-step instructions.

How to do it...

Let us set up Hadoop by setting up the JobTracker and TaskTrackers.

1. In each machine, create a directory for Hadoop data. Let's call this directory `HADOOP_DATA_DIR`. Then create three directories, `HADOOP_DATA_DIR/data`, `HADOOP_DATA_DIR/local`, and `HADOOP_DATA_DIR/name`.
2. Set up SSH keys to all machines so that we can log in to all from the master node. The *Setting up HDFS* recipe describes the SSH setup in detail.
3. Unzip the Hadoop distribution at the same location in all machines using the `>tar -zxvf hadoop-1.x.x.tar.gz` command. You can use any of the Hadoop 1.0 branch distributions.
4. In all machines, edit the `HADOOP_HOME/conf/hadoop-env.sh` file by uncommenting the `JAVA_HOME` line and point it to your local Java installation. For example, if Java is in `/opt/jdk1.6`, change the `JAVA_HOME` line to `export JAVA_HOME=/opt/jdk1.6`.
5. Place the IP address of the node used as the master (for running JobTracker and NameNode) in `HADOOP_HOME/conf/masters` in a single line. If you are doing a single-node deployment, leave the current value, `localhost`, as it is.

   ```
   209.126.198.72
   ```

6. Place the IP addresses of all slave nodes in the `HADOOP_HOME/conf/slaves` file, each in a separate line.

   ```
   209.126.198.72
   209.126.198.71
   ```

7. Inside each node's `HADOOP_HOME/conf` directory, add the following to the `core-site.xml`, `hdfs-site.xml` and `mapred-site.xml`. Before adding the configurations, replace the `MASTER_NODE` with the IP of the master node and `HADOOP_DATA_DIR` with the directory you created in the first step.

   Add URL of the NameNode to `HADOOP_HOME/conf/core-site.xml`.

   ```
   <configuration>
   <property>
   <name>fs.default.name</name>
   <value>hdfs://MASTER_NODE:9000/</value>
   </property>
   </configuration>
   ```

   Add locations to store metadata (names) and data within `HADOOP_HOME/conf/hdfs-site.xml` to submit jobs:

   ```
   <configuration>
   <property>
   ```

```
<name>dfs.name.dir</name>
<value>HADOOP_DATA_DIR/name</value>
</property>
<property>
<name>dfs.data.dir</name>
<value>HADOOP_DATA_DIR/data</value>
</property>
</configuration>
```

Map reduce local directory is the location used by Hadoop to store temporary files used. Add JobTracker location to `HADOOP_HOME/conf/mapred-site.xml`. Hadoop will use this for the jobs. The final property sets the maximum map tasks per node, set it the same as the amount of cores (CPU).

```
<configuration>
<property>
<name>mapred.job.tracker</name>
<value>MASTER_NODE:9001</value>
</property>
<property>
<name>mapred.local.dir</name>
<value>HADOOP_DATA_DIR/local</value>
</property>
<property>
<name>mapred.tasktracker.map.tasks.maximum</name>
<value>8</value>
</property>
</configuration>
```

8. To format a new HDFS filesystem, run the following command from the Hadoop NameNode (master node). If you have done this as part of the HDFS installation in earlier recipe, you can skip this step.

```
>bin/hadoop namenode -format
...
/Users/srinath/playground/hadoop-book/hadoop-temp/dfs/name has
been successfully formatted.
12/04/09 08:44:51 INFO namenode.NameNode: SHUTDOWN_MSG:
/************************************************************
SHUTDOWN_MSG: Shutting down NameNode at Srinath-s-MacBook-Pro.
local/172.16.91.1
************************************************************/
```

9. In the master node, change the directory to `HADOOP_HOME` and run the following commands:

```
>bin/start-dfs.sh

starting namenode, logging to /root/hadoop-setup-srinath/
hadoop-1.0.0/libexec/../logs/hadoop-root-namenode-node7.beta.out
209.126.198.72: starting datanode, logging to /root/hadoop-setup-
srinath/hadoop-1.0.0/libexec/../logs/hadoop-root-datanode-node7.
beta.out
209.126.198.71: starting datanode, logging to /root/hadoop-setup-
srinath/hadoop-1.0.0/libexec/../logs/hadoop-root-datanode-node6.
beta.out
209.126.198.72: starting secondarynamenode, logging to /root/
hadoop-setup-srinath/hadoop-1.0.0/libexec/../logs/hadoop-root-
secondarynamenode-node7.beta.out
>bin/start-mapred.sh
starting jobtracker, logging to /root/hadoop-setup-srinath/
hadoop-1.0.0/libexec/../logs/hadoop-root-jobtracker-node7.beta.out
209.126.198.72: starting tasktracker, logging to /root/
hadoop-setup-srinath/hadoop-1.0.0/libexec/../logs/hadoop-root-
tasktracker-node7.beta.out
209.126.198.71: starting tasktracker, logging to /root/
hadoop-setup-srinath/hadoop-1.0.0/libexec/../logs/hadoop-root-
tasktracker-node6.beta.out
```

10. Verify the installation by listing the processes through the `ps | grep java` command (if you are using Linux) or via Task Manager (if you are in Windows), in the master node and slave nodes. Master node will list four processes—NameNode, DataNode, JobTracker, and TaskTracker and slaves will have a DataNode and TaskTracker.
11. Browse the web-based monitoring pages for namenode and JobTracker:
    - **NameNode**: `http://MASTER_NODE:50070/`.
    - **JobTracker**: `http://MASTER_NODE:50030/`.
12. You can find the logfiles under `${HADOOP_HOME}/logs`.
13. Make sure HDFS setup is OK by listing the files using HDFS command line.

```
bin/hadoop dfs -ls /

Found 2 items
drwxr-xr-x   - srinath supergroup    0 2012-04-09 08:47 /Users
drwxr-xr-x   - srinath supergroup    0 2012-04-09 08:47 /tmp
```

How it works...

As described in the introduction to the chapter, Hadoop installation consists of HDFS nodes, a JobTracker and worker nodes. When we start the NameNode, it finds the slaves through the `HADOOP_HOME/slaves` file and uses SSH to start the DataNodes in the remote server at the startup. Also when we start the JobTracker, it finds the slaves through the `HADOOP_HOME/slaves` file and starts the TaskTrackers.

There's more...

In the next recipe, we will discuss how to run the aforementioned WordCount program using the distributed setup. The following recipes will discuss how to use MapReduce monitoring UI to monitor the distributed Hadoop setup.

Running the WordCount program in a distributed cluster environment

This recipe describes how to run a job in a distributed cluster.

Getting ready

Start the Hadoop cluster.

How to do it...

Now let us run the WordCount sample in the distributed Hadoop setup.

1. To use as inputs to the WordCount MapReduce sample that we wrote in the earlier recipe, copy the `README.txt` file in your Hadoop distribution to the HDFS filesystem at the location `/data/input1`.

```
>bin/hadoop dfs -mkdir /data/
>bin/hadoop dfs -mkdir /data/input1
>bin/hadoop dfs -put README.txt /data/input1/README.txt
>bin/hadoop dfs -ls /data/input1

Found 1 items
-rw-r--r--   1 srinath supergroup       1366 2012-04-09 08:59 /
data/input1/README.txt
```

2. Now, let's run the WordCount example from the `HADOOP_HOME` directory.

```
>bin/hadoop jar hadoop-examples-1.0.0.jar wordcount /data/input1 /
data/output1

12/04/09 09:04:25 INFO input.FileInputFormat: Total input paths to
process : 1
12/04/09 09:04:26 INFO mapred.JobClient: Running job:
job_201204090847_0001
12/04/09 09:04:27 INFO mapred.JobClient:  map 0% reduce 0%
12/04/09 09:04:42 INFO mapred.JobClient:  map 100% reduce 0%
12/04/09 09:04:54 INFO mapred.JobClient:  map 100% reduce 100%
12/04/09 09:04:59 INFO mapred.JobClient: Job complete:
job_201204090847_0001
.....
```

3. Run the following commands to list the output directory and then look at the results.

```
>bin/hadoop dfs -ls /data/output1

Found 3 items
-rw-r--r--   1 srinath supergroup          0 2012-04-09 09:04 /
data/output1/_SUCCESS
drwxr-xr-x   - srinath supergroup          0 2012-04-09 09:04 /
data/output1/_logs
-rw-r--r--   1 srinath supergroup       1306 2012-04-09 09:04 /
data/output1/part-r-00000

>bin/hadoop dfs -cat /data/output1/*

(BIS),  1
(ECCN)   1
(TSU)  1
(see  1
5D002.C.1,  1
740.13)  1
```

How it works...

Job submission to the distributed Hadoop works in a similar way to the job submissions to local Hadoop installation, as described in the *Writing a WordCount MapReduce sample, bundling it and running it using standalone Hadoop* recipe. However, there are two main differences.

First, Hadoop stores both the inputs for the jobs and output generated by the job in HDFS filesystem. Therefore, we use step 1 to store the inputs in the HDFS filesystem and we use step 3 read outputs from the HDFS filesystem.

Secondly, when job is submitted, local Hadoop installation runs the job as a local JVM execution. However, the distributed cluster submits it to the JobTracker, and it executes the job using nodes in the distributed Hadoop cluster.

There's more...

You can see the results of the WordCount application also through the HDFS monitoring UI, as described in the *Using HDFS monitoring UI* recipe, and also you can see the statistics about the WordCount job as explained in the next recipe, *Using MapReduce Monitoring UI*.

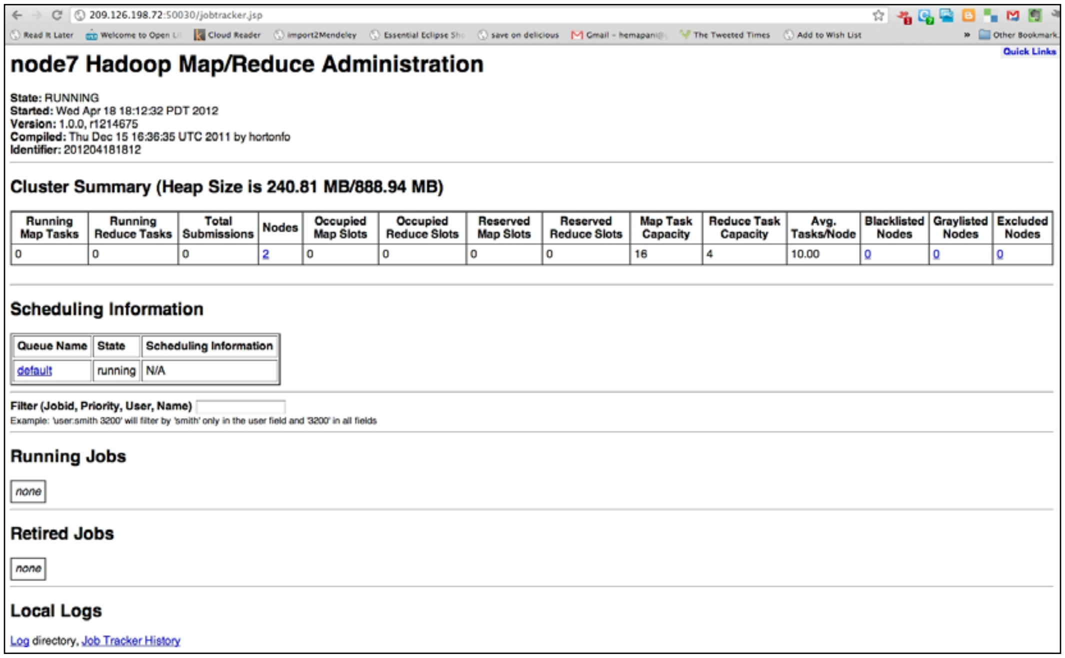
Using MapReduce monitoring UI

This recipe describes how to use the Hadoop monitoring web console to verify Hadoop installation, and to monitor the allocations and uses of each part of the Hadoop cluster.

How to do it...

Now let us visit the Hadoop monitoring web console.

1. Access `http://MASTER_NODE:50030/` using the browser where `MASTER_NODE` is the IP address of the master node.
2. The web page shows the current status of the MapReduce installation, including running and completed jobs.



How it works...

Hadoop monitoring UI lets users access the JobTracker of the Hadoop installation and find different nodes in the installation, their configurations, and usage. For example, users can use the UI to see current running jobs and logs associated with the jobs.