CHAPTER 1

Getting Started with Hadoop Core

Applications frequently require more resources than are available on an inexpensive machine. Many organizations find themselves with business processes that no longer fit on a single cost-effective computer. A simple but expensive solution has been to buy specialty machines that have a lot of memory and many CPUs. This solution scales as far as what is supported by the fastest machines available, and usually the only limiting factor is your budget. An alternative solution is to build a high-availability cluster. Such a cluster typically attempts to look like a single machine, and typically requires very specialized installation and administration services. Many high-availability clusters are proprietary and expensive.

A more economical solution for acquiring the necessary computational resources is cloud computing. A common pattern is to have bulk data that needs to be transformed, where the processing of each data item is essentially independent of other data items; that is, using a single-instruction multiple-data (SIMD) algorithm. Hadoop Core provides an open source framework for cloud computing, as well as a distributed file system.

This book is designed to be a practical guide to developing and running software using Hadoop Core, a project hosted by the Apache Software Foundation. This chapter introduces Hadoop Core and details how to get a basic Hadoop Core installation up and running.

Introducing the MapReduce Model

Hadoop supports the MapReduce model, which was introduced by Google as a method of solving a class of petascale problems with large clusters of inexpensive machines. The model is based on two distinct steps for an application:

- *Map*: An initial ingestion and transformation step, in which individual input records can be processed in parallel.
- *Reduce*: An aggregation or summarization step, in which all associated records must be processed together by a single entity.

The core concept of MapReduce in Hadoop is that input may be split into logical chunks, and each chunk may be initially processed independently, by a map task. The results of these individual processing chunks can be physically partitioned into distinct sets, which are then sorted. Each sorted chunk is passed to a reduce task. Figure 1-1 illustrates how the MapReduce model works.

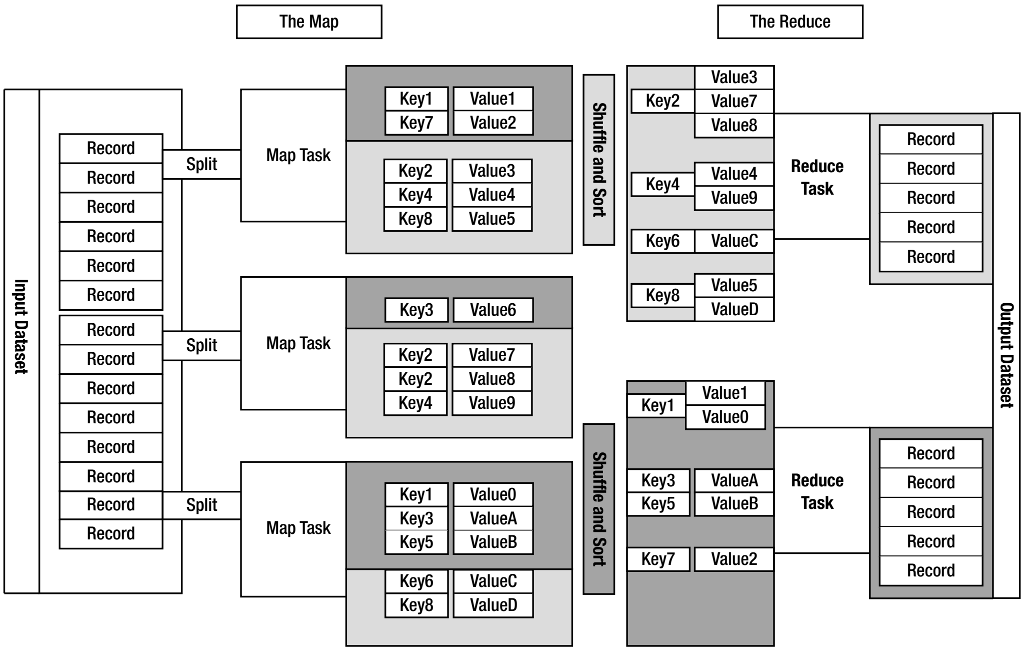

Figure 1-1. *The MapReduce model*

A map task may run on any compute node in the cluster, and multiple map tasks may be running in parallel across the cluster. The map task is responsible for transforming the input records into key/value pairs. The output of all of the maps will be partitioned, and each partition will be sorted. There will be one partition for each reduce task. Each partition's sorted keys and the values associated with the keys are then processed by the reduce task. There may be multiple reduce tasks running in parallel on the cluster.

The application developer needs to provide only four items to the Hadoop framework: the class that will read the input records and transform them into one key/value pair per record, a map method, a reduce method, and a class that will transform the key/value pairs that the reduce method outputs into output records.

My first MapReduce application was a specialized web crawler. This crawler received as input large sets of media URLs that were to have their content fetched and processed. The media items were large, and fetching them had a significant cost in time and resources. The job had several steps:

1. Ingest the URLs and their associated metadata.
2. Normalize the URLs.
3. Eliminate duplicate URLs.

4. Filter the URLs against a set of exclusion and inclusion filters.
5. Filter the URLs against a do not fetch list.
6. Filter the URLs against a recently seen set.
7. Fetch the URLs.
8. Fingerprint the content items.
9. Update the recently seen set.
10. Prepare the work list for the next application.

I had 20 machines to work with on this project. The previous incarnation of the application was very complex and used an open source queuing framework for distribution. It performed very poorly. Hundreds of work hours were invested in writing and tuning the application, and the project was on the brink of failure. Hadoop was suggested by a member of a different team.

After spending a day getting a cluster running on the 20 machines, and running the examples, the team spent a few hours working up a plan for nine map methods and three reduce methods. The goal was to have each map or reduce method take less than 100 lines of code. By the end of the first week, our Hadoop-based application was running substantially faster and more reliably than the prior implementation. Figure 1-2 illustrates its architecture. The fingerprint step used a third-party library that had a habit of crashing and occasionally taking down the entire machine.

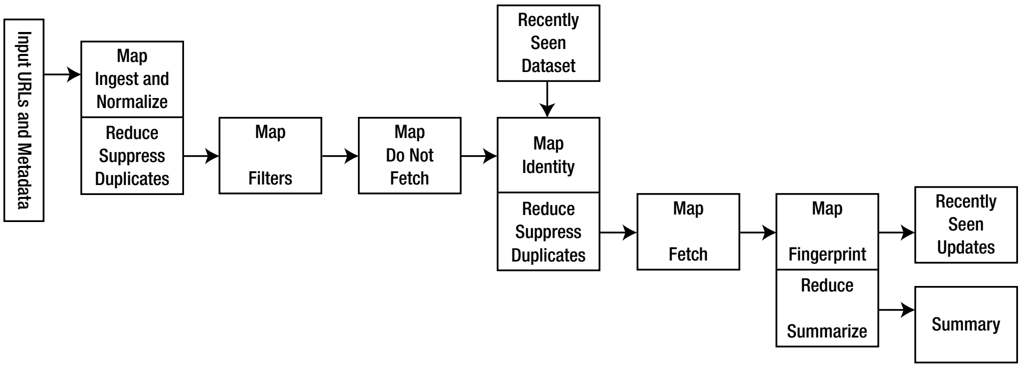

Figure 1-2. *The architecture of my first MapReduce application*

The ease with which Hadoop distributed the application across the cluster, along with the ability to continue to run in the event of individual machine failures, made Hadoop one of my favorite tools.

Both Google and Yahoo handle applications on the petabyte scale with MapReduce clusters. In early 2008, Google announced that it processes 20 petabytes of data a day with MapReduce (see `http://googleblog.blogspot.com/2008/11/sorting-1pb-with-mapreduce.html`).

Introducing Hadoop

Hadoop is the Apache Software Foundation top-level project that holds the various Hadoop subprojects that graduated from the Apache Incubator. The Hadoop project provides and supports the development of open source software that supplies a framework for the development of highly scalable distributed computing applications. The Hadoop framework handles the processing details, leaving developers free to focus on application logic.

Note The Hadoop logo is a stuffed yellow elephant. And Hadoop happened to be the name of a stuffed yellow elephant owned by the child of the principle architect.

The introduction on the Hadoop project web page (`http://hadoop.apache.org/`) states:

The Apache Hadoop project develops open-source software for reliable, scalable, distributed computing, including:

Hadoop Core, our flagship sub-project, provides a distributed filesystem (HDFS) and support for the MapReduce distributed computing metaphor.

HBase builds on Hadoop Core to provide a scalable, distributed database.

Pig is a high-level data-flow language and execution framework for parallel computation. It is built on top of Hadoop Core.

ZooKeeper is a highly available and reliable coordination system. Distributed applications use ZooKeeper to store and mediate updates for critical shared state.

Hive is a data warehouse infrastructure built on Hadoop Core that provides data summarization, adhoc querying and analysis of datasets.

The Hadoop Core project provides the basic services for building a cloud computing environment with commodity hardware, and the APIs for developing software that will run on that cloud. The two fundamental pieces of Hadoop Core are the MapReduce framework, the cloud computing environment, and he Hadoop Distributed File System (HDFS).

Note Within the Hadoop Core framework, MapReduce is often referred to as `mapred`, and HDFS is often referred to as `dfs`.

The Hadoop Core MapReduce framework requires a shared file system. This shared file system does not need to be a system-level file system, as long as there is a distributed file system plug-in available to the framework. While Hadoop Core provides HDFS, HDFS is not required. In Hadoop JIRA (the issue-tracking system), item 4686 is a tracking ticket to separate HDFS into its own Hadoop project. In addition to HDFS, Hadoop Core supports the Cloud-Store (formerly Kosmos) file system (`http://kosmosfs.sourceforge.net/`) and Amazon Simple Storage Service (S3) file system (`http://aws.amazon.com/s3/`). The Hadoop Core framework comes with plug-ins for HDFS, CloudStore, and S3. Users are also free to use any distributed file system that is visible as a system-mounted file system, such as Network File System (NFS), Global File System (GFS), or Lustre.

When HDFS is used as the shared file system, Hadoop is able to take advantage of knowledge about which node hosts a physical copy of input data, and will attempt to schedule the task that is to read that data, to run on that machine. This book mainly focuses on using HDFS as the file system.

Hadoop Core MapReduce

The Hadoop Distributed File System (HDFS)MapReduce environment provides the user with a sophisticated framework to manage the execution of map and reduce tasks across a cluster of machines. The user is required to tell the framework the following:

- The location(s) in the distributed file system of the job input
- The location(s) in the distributed file system for the job output
- The input format
- The output format
- The class containing the map function
- Optionally. the class containing the reduce function
- The JAR file(s) containing the map and reduce functions and any support classes

If a job does not need a reduce function, the user does not need to specify a reducer class, and a reduce phase of the job will not be run. The framework will partition the input, and schedule and execute map tasks across the cluster. If requested, it will sort the results of the map task and execute the reduce task(s) with the map output. The final output will be moved to the output directory, and the job status will be reported to the user.

MapReduce is oriented around key/value pairs. The framework will convert each record of input into a key/value pair, and each pair will be input to the map function once. The map output is a set of key/value pairs—nominally one pair that is the transformed input pair, but it is perfectly acceptable to output multiple pairs. The map output pairs are grouped and sorted by key. The reduce function is called one time for each key, in sort sequence, with the key and the set of values that share that key. The reduce method may output an arbitrary number of key/value pairs, which are written to the output files in the job output directory. If the reduce output keys are unchanged from the reduce input keys, the final output will be sorted.

The framework provides two processes that handle the management of MapReduce jobs:

- TaskTracker manages the execution of individual map and reduce tasks on a compute node in the cluster.
- JobTracker accepts job submissions, provides job monitoring and control, and manages the distribution of tasks to the TaskTracker nodes.

Generally, there is one JobTracker process per cluster and one or more TaskTracker processes per node in the cluster. The JobTracker is a single point of failure, and the JobTracker will work around the failure of individual TaskTracker processes.

■**Note** One very nice feature of the Hadoop Core MapReduce environment is that you can add TaskTracker nodes to a cluster while a job is running and have the job spread out onto the new nodes.

The Hadoop Distributed File System

HDFS is a file system that is designed for use for MapReduce jobs that read input in large chunks of input, process it, and write potentially large chunks of output. HDFS does not handle random access particularly well. For reliability, file data is simply mirrored to multiple storage nodes. This is referred to as *replication* in the Hadoop community. As long as at least one replica of a data chunk is available, the consumer of that data will not know of storage server failures.

HDFS services are provided by two processes:

- NameNode handles management of the file system metadata, and provides management and control services.
- DataNode provides block storage and retrieval services.

There will be one NameNode process in an HDFS file system, and this is a single point of failure. Hadoop Core provides recovery and automatic backup of the NameNode, but no hot failover services. There will be multiple DataNode processes within the cluster, with typically one DataNode process per storage node in a cluster.

■**Note** It is common for a node in a cluster to provide both TaskTracker services and DataNode services. It is also common for one node to provide the JobTracker and NameNode services.

Installing Hadoop

As with all software, you need some prerequisite pieces before you can actually use Hadoop. It is possible to run and develop Hadoop applications under Windows, provided that Cygwin is installed. It is strongly suggested that nodes in a production Hadoop cluster run a modern Linux distribution.

■**Note** To use Hadoop, you'll need a basic working knowledge of Linux and Java. All the examples in this book are set up for the bash shell.

The Prerequisites

The examples in this book were developed with the following:

- Fedora 8
- Sun Java 1.6
- Hadoop 0.19.0 or later

Hadoop versions prior to 0.18.2 make much less use of generics, and the book examples are unlikely to compile with those versions. Java versions prior to 1.6 will not support all of the language features that Hadoop Core requires. In addition, Hadoop Core appears to run most stably with the Sun Java Development Kits (JDKs); there are periodic requests for help from users of other vendors' JDKs. The examples in later chapters of this book are based on Hadoop 0.19.0, which requires JDK 1.6.

Any modern Linux distribution will work. I prefer the Red Hat Package Manager (RPM) tool that is used by Red Hat, Fedora, and CentOS, and the examples reference RPM-based installation procedures.

The wonderful folks of the Fedora project provide *torrents* (downloaded with BitTorrent) for most of the Fedora versions at `http://torrent.fedoraproject.org/`. For those who want to bypass the update process, the people of Fedora Unity provide distributions of Fedora releases that have the updates applied, at `http://spins.fedoraunity.org/spins`. These are referred to as *re-spins*. They do not provide older releases. The re-spins require the use of the custom download tool Jigdo.

For the novice Linux user who just wants to play around a bit, the Live CD and a USB stick for permanent storage can provide a simple and quick way to boot up a test environment. For a more sophisticated user, VMware Linux installation images are readily available at `http://www.vmware.com/appliances/directory/cat/45?sort=changed`.

Hadoop on a Linux System

After Linux is installed, it is necessary to work out where the JDK is installed on the computer so that the `JAVA_HOME` environment variable and the `PATH` environment variable may be set correctly.

The `rpm` command has options that will tell you which files were in an RPM package: `-q` to query, `-l` to list, and `-p` to specify that the path to package you are querying is the next argument. Look for the string `'/bin/javac$'`, using the `egrep` program, which searches for simple regular expressions in its input stream:

```
cloud9: ~/Downloads$ rpm -q -l -p ~/Downloads/jdk-6u7-linux-i586.rpm ➥
| egrep '/bin/javac$'
```

```
/usr/java/jdk1.6.0_07/bin/javac
```

■Note The single quotes surrounding the `/bin/javac$` are required. If you don't use quotes, or use double quotes, the shell may try to resolve the `$` character as a variable.

This assumes a working directory of `~/Downloads` when running the JDK installation program, as the installer unpacks the bundled RPM files in the current working directory.

The output indicates that the JDK was installed in `/usr/java/jdk1.6.0_07` and the `java` programs are in the directory `/usr/java/jdk1.6.0_07/bin`.

Add the following two lines to your `.bashrc` or `.bash_profile`:

```
export JAVA_HOME=/usr/java/jdk1.6.0_07
export PATH=${JAVA_HOME}/bin:${PATH}
```

The `update_env.sh` script, shown in Listing 1-1, will attempt to do this setup for you (this script is provided along with the downloadable code for this book). This script assumes you downloaded the RPM installer for the JDK.

Listing 1-1. *The update_env.sh Script*

```
#! /bin/sh

# This script attempts to work out the installation directory of the jdk,
# given the installer file.
# The script assumes that the installer is an rpm based installer and
# that the name of the downloaded installer ends in
# -rpm-bin
#
# The script first attempts to verify there is one argument and the
# argument is an existing file
# The file may be either the installer binary, the -rpm.bin
# or the actual installation rpm that was unpacked by the installer
#
```

```
# The script will use the rpm command to work out the
# installation package name from the rpm file, and then
# use the rpm command to query the installation database,
# for where the files of the rpm were installed.

# This query of the installation is done rather than
# directly querying the rpm, on the off
# chance that the installation was installed in a different root
# directory than the default.

# Finally, the proper environment set commands are appended
# to the user's .bashrc and .bash_profile file, if they exist, and
# echoed to the standard out so the user may apply them to
# their currently running shell sessions.

# Verify that there was a single command line argument
# which will be referenced as $1
if [ $# != 1 ]; then
    echo "No jdk rpm specified"
    echo "Usage: $0 jdk.rpm" 1>&2
    exit 1
fi

# Verify that the command argument exists in the file system
if [ ! -e $1 ]; then
    echo "the argument specified ($1) for the jdk rpm does not exist" 1>&2
    exit 1
fi

# Does the argument end in '-rpm.bin' which is the suggested install
# file, is the argument the actual .rpm file, or something else
# set the variable RPM to the expected location of the rpm file that
# was extracted from the installer file
if echo $1 | grep -q -e '-rpm.bin'; then
    RPM=`dirname $1`/`basename $1 -rpm.bin`.rpm
elif echo $1 | grep -q -e '.rpm'; then
    RPM=$1
else
    echo -n "$1 does not appear to be the downloaded rpm.bin file or" 1>&2
    echo " the extracted rpm file" 1>&2
    exit 1
fi

# Verify that the rpm file exists and is readable
```

```
if [ ! -r $RPM ]; then
    echo -n "The jdk rpm file (${RPM}) does not appear to exist" 1>&2
    echo -n " have you run "sh ${RPM}" as root?" 1>&2
    exit 1
fi

# Work out the actual installed package name using the rpm command
#. man rpm for details
INSTALLED=`rpm -q --qf %{Name}-%{Version}-%{Release} -p ${RPM}`
if [ $? -ne 0 ]; then
    (echo -n "Unable to extract package name from rpm (${RPM}),"
     Echo " have you installed it yet?") 1>&2
    exit 1
fi

# Where did the rpm install process place the java compiler program 'javac'
JAVAC=`rpm -q -l ${INSTALLED} | egrep '/bin/javac$'`

# If there was no javac found, then issue an error
if [ $? -ne 0 ]; then
    (echo -n "Unable to determine the JAVA_HOME location from $RPM, "
     echo "was the rpm installed? Try rpm -Uvh ${RPM} as root.") 1>&2
    exit 1
fi

# If we found javac, then we can compute the setting for JAVA_HOME
JAVA_HOME=`echo $JAVAC | sed -e 's;/bin/javac;;'`


echo "The setting for the JAVA_HOME environment variable is ${JAVA_HOME}"

echo -n "update the user's .bashrc if they have one with the"
echo " setting for JAVA_HOME and the PATH."
if [ -w ~/.bashrc ]; then
    echo "Updating the ~/.bashrc file with the java environment variables";
    (echo export JAVA_HOME=${JAVA_HOME} ;
        echo export PATH='${JAVA_HOME}'/bin:'${PATH}' ) >> ~/.bashrc
    echo
fi
```

```
echo -n "update the user's .bash_profile if they have one with the"
echo " setting for JAVA_HOME and the PATH."
if [ -w ~/.bash_profile ]; then
    echo "Updating the ~/.bash_profile file with the java environment variables";
    (echo export JAVA_HOME=${JAVA_HOME} ;
        echo export PATH='${JAVA_HOME}'/bin:'${PATH}' ) >> ~/.bash_profile
    echo
fi

echo "paste the following two lines into your running shell sessions"
echo export JAVA_HOME=${JAVA_HOME}
echo export PATH='${JAVA_HOME}'/bin:'${PATH}'
```

Run the script in Listing 1-1 to work out the JDK installation directory and update your environment so that the JDK will be used by the examples:

```
update_env.sh "FULL_PATH_TO_DOWNLOADED_JDK"

./update_env.sh ~/Download/jdk-6u7-linux-i586-rpm.bin
```

```
The setting for the JAVA_HOME environment variable is /usr/java/jdk1.6.0_07

update the user's .bashrc if they have one with the setting ➥
for JAVA_HOME and the PATH.

Updating the ~/.bashrc file with the java environment variables


update the user's .bash_profile if they have one with the setting ➥
for JAVA_HOME and the PATH.

Updating the ~/.bash_profile file with the java environment variables


paste the following two lines into your running shell sessions

export JAVA_HOME=/usr/java/jdk1.6.0_07

export PATH=${JAVA_HOME}/bin:${PATH}
```

Hadoop on a Windows System: How To and Common Problems

To use Hadoop on a Windows system, you will need to install the Sun JDK and the Cygwin environment (available from `http://sources.redhat.com/cygwin`).

Run a Cygwin bash shell by clicking the icon shown in Figure 1-3. You will need to make a symbolic link in the ~ directory between the JDK installation directory and `java`, so that `cd ~/java` will change the working directory to the root of the JDK directory. The appropriate setting for `JAVA_HOME` becomes `export JAVA_HOME=~/java`. This will set up your default process environment to have the `java` programs in your path and let programs, such as Hadoop, know where the Java installation is on your computer.

Figure 1-3. *The Cygwin bash shell icon*

I was unable to make the `bin/hadoop` script work if the path in the `JAVA_HOME` environment variable contained space characters, and the normal installation is in `C:\Program Files\java\jdkRELEASE_VERSION`. When a symbolic link is made and the `JAVA_HOME` set to the symbolic link location, `bin/hadoop` works well enough to use. For my Cygwin installation, I have the following:

```
$ echo $JAVA_HOME
```

```
/home/Jason/jdk1.6.0_12/
```

```
$ ls -l /home/Jason/jdk1.6.0_12
```

```
lrwxrwxrwx 1 Jason None 43 Mar 20 16:32 /home/Jason/jdk1.6.0_12 ➥
 -> /cygdrive/c/Program Files/Java/jdk1.6.0_12/
```

Cygwin maps Windows drive letters to the path `/cygdrive/X`, where `X` is the drive letter, and the Cygwin path element separator character is `/`, compared to Windows use of `\`.

You must keep two views of your files in mind, particularly when running Java programs via the `bin/hadoop` script. The `bin/hadoop` script and all of the Cygwin utilities see a file system that is a subtree of the Windows file system, with the Windows drives mapped to the `/cygdrive` directory. The Windows programs see the traditional `C:\` file system. An example of this is `/tmp`. In a standard Cygwin installation, the `/tmp` directory is also the `C:\cygwin\tmp` directory. Java will parse `/tmp` as `C:\tmp`, a completely different directory. When you receive File Not Found errors from Windows applications launched from Cygwin, the common problem is that the Windows application (Java being a Windows application) is looking in a different directory than you expect.

■Note You will need to customize the Cygwin setup for your system. The exact details change with different Sun JDK releases and with different Windows installations. In particular, the username will probably not be `Jason`, the JDK version may not be `1.6.0_12`, and the Java installation location may not be `C:\Program Files\Java`.

Getting Hadoop Running

After you have your Linux or Cygwin under Windows environment set up, you're ready to download and install Hadoop.

Go to the Hadoop download site at `http://www.apache.org/dyn/closer.cgi/hadoop/core/`. From there, find the tar.gz file of the distribution of your choice, bearing in mind what I said in the introduction, and download that file.

If you are a cautious person, go to the backup site and get the PGP checksum or the MD5 checksum of the download file.

Unpack the `tar` file in the directory where you would like your test installation installed. I typically unpack this in a `src` directory, off my personal home directory:

```
~jason/src.
mkdir ~/src
cd ~/src
tar zxf ~/Downloads/hadoop-0.19.0.tar.gz
```

This will create a directory named `hadoop-0.19.0` in my `~/src` directory.

Add the following two lines to your `.bashrc` or `.bash_profile` file and execute them in your current shell:

```
export HADOOP_HOME=~/src/hadoop-0.19.0
export PATH=${HADOOP_HOME}/bin:${PATH}
```

If you chose a different directory than `~/src`, adjust these `export` statements to reflect your chosen location.

Checking Your Environment

After installing Hadoop, you should check that you have updated your shell environment with the `JAVA_HOME` and `HADOOP_HOME` environment variables correctly; that your `PATH` environment variable has `${JAVA_HOME}/bin` and `${HADOOP_HOME}/bin` to the left of any other Java or Hadoop installations in your path, preferably as the first to elements of your `PATH`; and that your shell's current working directory is `${HADOOP_HOME}`. These settings are required to run the examples in this book.

The shell script `check_basic_env.sh`, shown in Listing 1-2, will verify your runtime environment (this script is provided along with the other downloadable code for this book).

Listing 1-2. *The check_basic_env.sh Script*

```
#! /bin/sh

# This block is trying to do the basics of checking to see if
# the HADOOP_HOME and the JAVA_HOME variables have been set correctly
# and if they are not been set, suggest a setting in line with the earlier examples

# The script actually tests for:
# the presence of the java binary and the hadoop script,
# and verifies that the expected versions are present
# that the version of java and hadoop is as expected (warning if not)
# that the version of java and hadoop referred to by the
# JAVA_HOME and HADOOP_HOME environment variables are default version to run.
#
#
# The 'if [' construct you see is a shortcut for 'if test' ....
# the -z tests for a zero length string
# the -d tests for a directory
# the -x tests for the execute bit
# -eq tests numbers
# = tests strings
# man test will describe all of the options

# The '1>&2' construct directs the standard output of the
# command to the standard error stream.

if [ -z "$HADOOP_HOME" ]; then
    echo "The HADOOP_HOME environment variable is not set" 1>&2
    if [ -d ~/src/hadoop-0.19.0 ]; then
        echo "Try export HADOOP_HOME=~/src/hadoop-0.19.0" 1>&2
    fi
    exit 1;
fi

# This block is trying to do the basics of checking to see if
# the JAVA_HOME variable has been set
# and if it hasn't been set, suggest a setting in line with the earlier examples


if [ -z "$JAVA_HOME" ]; then
    echo "The JAVA HOME environment variable is not set" 1>&2
    if [ -d /usr/java/jdk1.6.0_07 ]; then
        echo "Try export JAVA_HOME=/usr/java/jdk1.6.0_07" 1>&2
    fi
    exit 1
fi
```

```
# We are now going to see if a java program and hadoop programs
# are in the path, and if they are the ones we are expecting.
# The which command returns the full path to the first instance
# of the program in the PATH environment variable
#
JAVA_BIN=`which java`
HADOOP_BIN=`which hadoop`

# Check for the presence of java in the path and suggest an
# appropriate path setting if java is not found
if [ -z "${JAVA_BIN}" ]; then
    echo "The java binary was not found using your PATH settings" 1>&2
    if [ -x ${JAVA_HOME}/bin/java ]; then
        echo 'Try export PATH=${JAVA_HOME}/bin' 1>&2
    fi
    exit 1
fi

# Check for the presence of hadoop in the path and suggest an
# appropriate path setting if java is not found
if [ -z "${HADOOP_BIN}" ]; then
    echo "The hadoop binary was not found using your PATH settings" 1>&2
    if [ -x ${HADOOP_HOME}/bin/hadoop ]; then
        echo 'Try export PATH=${HADOOP_HOME}/bin:${PATH}' 1>&2
    fi
    exit 1
fi


# Double check that the version of java installed in ${JAVA_HOME}
# is the one stated in the examples.
# If you have installed a different version your results may vary.
#
if ! ${JAVA_HOME}/bin/java -version 2>&1 | grep -q 1.6.0_07; then
    (echo -n "Your JAVA_HOME version of java is not the"
     echo -n " 1.6.0_07 version, your results may vary from"
     echo " the book examples.") 1>&2
fi

# Double check that the java in the PATH is the expected version.
if ! java -version 2>&1 | grep -q 1.6.0_07; then
    (echo -n "Your default java version is not the 1.6.0_07 "
     echo -n "version, your results may vary from the book"
     echo " examples.") 1>&2
fi
```

```
# Try to get the location of the hadoop core jar file
# This is used to verify the version of hadoop installed
HADOOP_JAR=`ls -1 ${HADOOP_HOME}/hadoop-0.19.0-core.jar`
HADOOP_ALT_JAR=`ls -1 ${HADOOP_HOME}/hadoop-*-core.jar`
# If a hadoop jar was not found, either the installation
# was incorrect or a different version installed
if [ -z "${HADOOP_JAR}" -a -z "${HADOOP_ALT_JAR}" ]; then
    (echo -n "Your HADOOP_HOME does not provide a hadoop"
     echo -n " core jar. Your installation probably needs"
     echo -n " to be redone or the HADOOP_HOME environment"
     echo variable needs to be correctly set.") 1>&2
    exit 1
fi

if [ -z "${HADOOP_JAR}" -a ! -z "${HADOOP_ALT_JAR}" ]; then
    (echo -n "Your hadoop version appears to be different"
     echo -n " than the 0.19.0 version, your results may vary"
     echo " from the book examples.") 1>&2
fi

if [ `pwd` != ${HADOOP_HOME} ]; then
    (echo -n 'Please change your working directory to"
     echo -n " ${HADOOP_HOME}. cd ${HADOOP_HOME} <Enter>") 1>&2
    exit 1
fi

echo "You are good to go"
echo -n "your JAVA_HOME is set to ${JAVA_HOME} which "
echo "appears to exist and be the right version for the examples."
echo -n "your HADOOP_HOME is set to ${HADOOP_HOME} which "
echo "appears to exist and be the right version for the examples."
echo "your java program is the one in ${JAVA_HOME}"
echo "your hadoop program is the one in ${HADOOP_HOME}"
echo -n "The shell current working directory is ${HADOOP_HOME} "
echo "as the examples require."


if [ "${JAVA_BIN}" = "${JAVA_HOME}/bin/java" ]; then
    echo "Your PATH appears to have the JAVA_HOME java program as the default java."
else
    echo -n "Your PATH does not appear to provide the JAVA_HOME"
    echo " java program as the default java."
fi
```

```
if [ "${HADOOP_BIN}" = "${HADOOP_HOME}/bin/hadoop" ]; then
    echo -n "Your PATH appears to have the HADOOP_HOME"
    echo " hadoop program as the default hadoop."
else
    echo -n "Your PATH does not appear to provide the the HADOOP_HOME "
    echo "hadoop program as the default hadoop program."
fi


exit 0
```

Run the script as follows:

```
[scyrus@localhost ~]$ ./check_basic_env.sh
```

```
Please change your working directory to ${HADOOP_HOME}. cd ➥
${HADOOP_HOME} <Enter>
```

```
[scyrus@localhost ~]$ cd $HADOOP_HOME
[scyrus@localhost hadoop-0.19.0]$

[scyrus@localhost hadoop-0.19.0]$ ~/check_basic_env.sh
```

```
You are good to go
your JAVA_HOME is set to /usr/java/jdk1.6.0_07 which appears to exist
and be the right version for the examples.
your HADOOP_HOME is set to /home/scyrus/src/hadoop-0.19.0 which appears
to exist and be the right version for the examples.
your java program is the one in /usr/java/jdk1.6.0_07
your hadoop program is the one in /home/scyrus/src/hadoop-0.19.0
The shell current working directory is /home/scyrus/src/hadoop-0.19.0 as
the examples require.
Your PATH appears to have the JAVA_HOME java program as the default
java.
Your PATH appears to have the HADOOP_HOME hadoop program as the default
hadoop.
```

Running Hadoop Examples and Tests

The Hadoop installation provides JAR files with sample programs and tests that you can run. Before you run these, you should have verified that your installation is complete and that your runtime environment is set up correctly. As discussed in the previous section, the `check_basic_env.sh` script will help verify your installation and suggest corrections if any are required.

Hadoop Examples

The `hadoop-0.19.0-examples.jar` file includes ready-to-run sample programs. Included in the JAR file are the programs listed in Table 2-1.

Table 2-1. *Examples in hadoop-0.19.0-examples.jar*

Program	Description
`aggregatewordcount`	An aggregate-based MapReduce program that counts the words in the input files
`aggregatewordhist`	An aggregate-based MapReduce program that computes the histogram of the words in the input files
`grep`	A MapReduce program that counts the matches of a regular expression in the input
`join`	A job that performs a join over sorted, equally partitioned datasets
`multifilewc`	A job that counts words from several files
`pentomino`	A MapReduce tile-laying program to find solutions to pentomino problems
`pi`	A MapReduce program that estimates pi using the Monte Carlo method
`randomtextwriter`	A MapReduce program that writes 10GB of random textual data per node
`randomwriter`	A MapReduce program that writes 10GB of random data per node
`sleep`	A job that sleeps at each map and reduce task
`sort`	A MapReduce program that sorts the data written by the random writer
`sudoku`	A sudoku solver
`wordcount`	A MapReduce program that counts the words in the input files

To demonstrate using the Hadoop examples, let's walk through running the `pi` program.

Running the Pi Estimator

The `pi` example estimates pi using the Monte Carlo method. The web site `http://www.chem.unl.edu/zeng/joy/mclab/mcintro.html` provides a good discussion of this technique. The number of samples is the number of points randomly set in the square. The larger this value, the more accurate the calculation of pi. For the sake of simplicity, we are going to make a very poor estimate of pi by using very few operations.

The `pi` program takes two integer arguments: the number of maps and the number of samples per map. The total number of samples used in the calculation is the number of maps times the number of samples per map.

The map task generates a random point in a 1×1 area. For each sample where $X^2+Y^2 <=1$, the point is inside; otherwise, the point is outside. The map outputs a key of 1 or 0 and a value of 1 for a point that is inside or outside the circle, diameter 1. The reduce task sums the number of inside points and the number of outside points. The ratio between this is, in the limit, pi.

In this example, to help the job run quicker and with less output, you will choose 2 maps, with 10 samples each, for a total of 20 samples.

To run the example, change the working directory of your shell to `HADOOP_HOME` (via `cd ${HADOOP_HOME}`) and enter the following:

```
jason@cloud9:~/src/hadoop-0.19.0$ hadoop jar hadoop-0.19.0-examples.jar pi 2 10
```

The `bin/hadoop jar` command submits jobs to the cluster. The command-line arguments are processed in three steps, with each step consuming some of the command-line arguments. We'll see this in detail in Chapter 5, but for now the `hadoop-0.19.0-examples.jar` file contains the main class for the application. The next three arguments are passed to this class.

Examining the Output: Input Splits, Shuffles, Spills, and Sorts

Your output will look something like that shown in Listing 2-3.

Listing 2-3. *Output from the Sample Pi Program*

```
Number of Maps = 2 Samples per Map = 10
Wrote input for Map #0
Wrote input for Map #1
Starting Job
jvm.JvmMetrics: Initializing JVM Metrics with processName=JobTracker, sessionId=
mapred.FileInputFormat: Total input paths to process : 2
mapred.FileInputFormat: Total input paths to process : 2
mapred.JobClient: Running job: job_local_0001
mapred.FileInputFormat: Total input paths to process : 2
mapred.FileInputFormat: Total input paths to process : 2
mapred.MapTask: numReduceTasks: 1
mapred.MapTask: io.sort.mb = 100
mapred.MapTask: data buffer = 79691776/99614720
mapred.MapTask: record buffer = 262144/327680
mapred.JobClient:  map 0% reduce 0%
mapred.MapTask: Starting flush of map output
mapred.MapTask: bufstart = 0; bufend = 32; bufvoid = 99614720
mapred.MapTask: kvstart = 0; kvend = 2; length = 327680
mapred.LocalJobRunner: Generated 1 samples
mapred.MapTask: Index: (0, 38, 38)
mapred.MapTask: Finished spill 0
mapred.LocalJobRunner: Generated 1 samples.
mapred.TaskRunner: Task 'attempt_local_0001_m_000000_0' done.
mapred.TaskRunner: Saved output of task 'attempt_local_0001_m_000000_0' ➥
to file:/home/jason/src/hadoop-0.19.0/test-mini-mr/outmapred.
MapTask: numReduceTasks: 1
mapred.MapTask: io.sort.mb = 100
mapred.JobClient:  map 0% reduce 0%
mapred.LocalJobRunner: Generated 1 samples
mapred.MapTask: data buffer = 79691776/99614720
mapred.MapTask: record buffer = 262144/327680
mapred.MapTask: Starting flush of map output
mapred.MapTask: bufstart = 0; bufend = 32; bufvoid = 99614720
mapred.MapTask: kvstart = 0; kvend = 2; length = 327680
mapred.JobClient:  map 100% reduce 0%
mapred.MapTask: Index: (0, 38, 38)
mapred.MapTask: Finished spill 0
mapred.LocalJobRunner: Generated 1 samples.
mapred.TaskRunner: Task 'attempt_local_0001_m_000001_0' done.
mapred.TaskRunner: Saved output of task 'attempt_local_0001_m_000001_0' ➥
to file:/home/jason/src/hadoop-0.19.0/test-mini-mr/out
mapred.ReduceTask: Initiating final on-disk merge with 2 files
mapred.Merger: Merging 2 sorted segments
mapred.Merger: Down to the last merge-pass, with 2 segments left of ➥
total size: 76 bytes
mapred.LocalJobRunner: reduce > reduce
mapred.TaskRunner: Task 'attempt_local_0001_r_000000_0' done.
mapred.TaskRunner: Saved output of task 'attempt_local_0001_r_000000_0' ➥
to file:/home/jason/src/hadoop-0.19.0/test-mini-mr/out
mapred.JobClient: Job complete: job_local_0001
mapred.JobClient: Counters: 11
mapred.JobClient:   File Systems
mapred.JobClient:     Local bytes read=314895
mapred.JobClient:     Local bytes written=359635
mapred.JobClient:   Map-Reduce Framework
mapred.JobClient:     Reduce input groups=2
mapred.JobClient:     Combine output records=0
mapred.JobClient:     Map input records=2
mapred.JobClient:     Reduce output records=0
mapred.JobClient:     Map output bytes=64
mapred.JobClient:     Map input bytes=48
mapred.JobClient:     Combine input records=0
mapred.JobClient:     Map output records=4
mapred.JobClient:     Reduce input records=4
Job Finished in 2.322 seconds
Estimated value of PI is 3.8
```

■**Note** The Hadoop projects use the Apache Foundation's log4j package for logging. By default, all output by the framework will have a leading date stamp, a log level, and the name of the class that emitted the message. In addition, the default is only to emit log messages of level `INFO` or higher. For brevity, I've removed the data stamp and log level from the output reproduced in this book.

Of particular interest here is that the last line of output states something of the form "Estimated value of PI is...". In that case, you know that your local installation of Hadoop is ready for you to play with.

Now we will go through the output in Listing 2-3 chunk by chunk, so that you have an understanding of what is going on and can recognize when something is wrong.

The first section is output by the pi estimator as it is setting up the job. Here, you requested 2 maps and 10 samples:

```
Number of Maps = 2 Samples per Map = 10
Wrote input for Map #0
Wrote input for Map #1
```

The framework has taken over at this point and sets up input splits (each fragment of input is called an *input split*) for the map tasks.

The following line provides the job ID, which you could use to refer to this job with the job control tools:

```
Running job: job_local_0001
```

The following lines let you know that there are two input files and two input splits:

```
jvm.JvmMetrics: Initializing JVM Metrics with processName=JobTracker, sessionId=
mapred.FileInputFormat: Total input paths to process : 2
mapred.FileInputFormat: Total input paths to process : 2
mapred.JobClient: Running job: job_local_0001
mapred.FileInputFormat: Total input paths to process : 2
mapred.FileInputFormat: Total input paths to process : 2
```

The map output key/value pairs are partitioned, and then the partitions are sorted, which is referred to as the *shuffle*. The file created for each sorted partition is called a *spill*. There will be one spill file for each configured reduce task. For each reduce task, the framework will pull its spill from the output of each map task, and merge-sort these spills. This step is referred to as the *sort*.

In Listing 2-3, the next block provides detailed information on the map task and shuffle process that was run. The framework is expecting to produce output for one reduce task (`numReduceTasks: 1`), which receives all of the map task output records. Also, it expects that the map outputs have been partitioned and sorted and stored in the file system (`Finished spill 0`). If there were multiple reduce tasks specified, you would see a `Finished spill` *N* for each reduce task. The rest of the lines primarily have to do with output buffering and may be ignored.

Next, you see the following:

```
mapred.MapTask: numReduceTasks: 1
...
mapred.MapTask: Finished spill 0
mapred.LocalJobRunner: Generated 1 samples.

mapred.TaskRunner: Task 'attempt_local_0001_m_000000_0' ➥
done.mapred.TaskRunner: Saved output of task ➥
'attempt_local_0001_m_000000_0' ➥
to file:/home/jason/src/hadoop-0.19.0/test-mini-mr/out
```

`Generated 1 samples` is the output of the ending status of the map job. The Hadoop framework is telling you that the first map task is done via `Task 'attempt_local_0001_m_000000_0' done`, and that the output was saved to the default file system at `file:/home/jason/src/hadoop-0.19.0/test-mini-mr/out`.

The following block handles the sort:

```
mapred.ReduceTask: Initiating final on-disk merge with 2 files
mapred.Merger: Merging 2 sorted segments
mapred.Merger: Down to the last merge-pass, with 2 segments left of ➥
total size: 76 bytes
```

Listing 2-3 has exactly two map tasks, per your command-line instructions to the task, and one reduce task, per the job design.. With a single reduce task, each map task's output is placed into a single partition and sorted. This results in two files, or *spills*, as input to the framework sort phase. Each reduce task in a job will have its output go to the output directory and be named `part-0N`, where `N` is the ordinal number starting from zero of the reduce task. The numeric portion of the name is traditionally five digits, with leading zeros as needed.

The next block describes the single reduce task that will be run:

```
mapred.LocalJobRunner: reduce > reduce
mapred.TaskRunner: Task 'attempt_local_0001_r_000000_0' done.
mapred.TaskRunner: Saved output of task 'attempt_local_0001_r_000000_0' to ➥
file:/home/jason/src/hadoop-0.19.0/test-mini-mr/out
```

The output of this reduce task is written to `attempt_local_0001_r_000000_0`, and then will be renamed to `part-00000` in the job output directory.

The next block of output provides summary information about the completed job:

```
mapred.JobClient: Job complete: job_local_0001
mapred.JobClient: Counters: 11
mapred.JobClient:   File Systems
mapred.JobClient:     Local bytes read=314895
mapred.JobClient:     Local bytes written=359635
mapred.JobClient:   Map-Reduce Framework
mapred.JobClient:     Reduce input groups=2
mapred.JobClient:     Combine output records=0
mapred.JobClient:     Map input records=2
mapred.JobClient:     Reduce output records=0
mapred.JobClient:     Map output bytes=64
mapred.JobClient:     Map input bytes=48
mapred.JobClient:     Combine input records=0
mapred.JobClient:     Map output records=4
mapred.JobClient:     Reduce input records=4
```

The final two lines are printed by the `PiEstimator` code, not the framework.

```
Job Finished in 2.322 seconds
Estimated value of PI is 3.8
```

Hadoop Tests

Hadoop provides a JAR that contains tests `hadoop-0.19.0-test.jar`, which are primarily for testing the distributed file system or MapReduce jobs on top of the distributed file system. Table 2-2 lists the tests provided

Table 2-2. *Tests in hadoop-0.19.0-test.jar*

Test	Description
`DFSCIOTest`	Distributed I/O benchmark of `libhdfs`, a shared library that provides HDFS file services for C/C++ applications
`DistributedFSCheck`	Distributed checkup of the file system consistency
`TestDFSIO`	Distributed I/O benchmark
`clustertestdfs`	A pseudo distributed test for the distributed file system
`dfsthroughput`	Measures HDFS throughput
`filebench`	Benchmark `SequenceFileInputFormat` and `SequenceFileOutputFormat`, with `BLOCK` compression, `RECORD` compression, and no compression; and `TextInputFormat` and `TextOutputFormat`, compressed and uncompressed
`loadgen`	Generic MapReduce load generator
`mapredtest`	A MapReduce test check
`mrbench`	A MapReduce benchmark that can create many small jobs
`nnbench`	A benchmark that stresses the NameNode
`testarrayfile`	A test for flat files of binary key/value pairs
`testbigmapoutput`	A MapReduce program that works on a very big nonsplittable file and does an identity MapReduce
`testfilesystem`	A test for file system read/write
`testipc`	A test for Hadoop Core interprocess communications
`testmapredsort`	A MapReduce program that validates the MapReduce framework's sort
`testrpc`	A test for remote procedure calls
`testsequencefile`	A test for flat files of binary key/value pairs
`testsequencefileinputformat`	A test for sequence file input format
`testsetfile`	A test for flat files of binary key/value pairs
`testtextinputformat`	A test for text input format
`threadedmapbench`	A MapReduce benchmark that compares the performance of maps with multiple spills over maps with one spill

Troubleshooting

The issues that can cause problems in running the examples in this book will most likely be due to environment differences. You may also experience problems if you have space shortages on your computer.

The following environment variables are important:

`JAVA_HOME`: This is the root of the Java installations. All of the examples assume that the `JAVA_HOME` environment variable contains the root of the Sun JDK 1.6_07 installation, which is expected to be installed into the directory `/usr/java/jdk1.6.0_07`. So, this environment variable is set as follows: `export JAVA_HOME=/usr/java/jdk1.6.0_07`.

`HADOOP_HOME`: This is the root of the Hadoop installations. You should have unpacked the `hadoop-0.19.0.tar.gz` downloaded file with a parent directory of `~/src`, such that the Hadoop program is available as `~/src/hadoop-0.19.0/bin/hadoop`. The `HADOOP_HOME` environment variable is expected to be set to the root of the Hadoop installation, which in the examples is `~/src/hadoop-0.19.0`. This environment variable is set as follows: `export HADOOP_HOME=~/src/hadoop-0.19.0`.

`PATH`: The user's path is expected to have `${JAVA_HOME}/bin` and `${HADOOP_HOME}/bin` as the first two elements. This environment variable is set as follows: `export PATH=${JAVA_HOME}/bin:${HADOOP_HOME}/bin:${PATH}`.

For Windows users, `C:\cygwin\bin;C:\cygwin\usr\bin` must be added to the system environment `Path` variable, or the Hadoop Core servers will not start. You can set this system variable through the System Control Panel. In the System Properties dialog box, click the Advanced tab, and then click the Environment Variables button. In the System Variables section of the Environment Variables dialog box, select Path, click Edit, and add the following string:

```
C:\cygwin\bin;C:\cygwin\usr\bin
```

The semicolon (`;`) is the separator character.

While not critical, the current working directory for the shell session used for running the examples is expected to be `${HADOOP_HOME}`.

If you see the message `java.lang.OutOfMemoryError: Java heap space` in your output, your computer either has insufficient free RAM or the Java heap is set too small. The `PiEstimator` example with 2 maps and 100 samples will run with a Java Virtual Machine (JVM) maximum heap size of 128MB (`-Xmx128m`). To force this, you may execute the following:

```
HADOOP_OPTS="-Xmx128m" hadoop  jar hadoop-0.19.0-examples.jar pi 2 100
```

Summary

Hadoop Core provides a robust framework for distributing tasks across large numbers of general-purpose computers. Application developers just need to write the map and reduce methods for their data, and use one of the existing input and output formats. The framework provides a rich set of input and output handlers, and you can create custom handlers, if necessary.

Getting over the installation hurdle can be difficult, but it is getting simpler as more people and organizations understand the issues and refine the processes and procedures. Cloudera (`http://www.cloudera.com`) now provides a self-installing Hadoop distribution in RPM format.

New features and functionality are being tried. Read the information on the `http://hadoop.apache.org/core` web site, join the mailing lists referenced there (to join the Core user mailing list, send an e-mail to `core-user-subscribe@hadoop.apache.org`), and have fun writing your applications.

The chapters to come will guide you through the trouble spots as you develop your own applications with Hadoop.