APPENDIX A

Installing Apache Hadoop

It's easy to install Hadoop on a single machine to try it out. (For installation on a cluster, refer to Chapter 10.)

In this appendix, we cover how to install Hadoop Common, HDFS, MapReduce, and YARN using a binary tarball release from the Apache Software Foundation. Instructions for installing the other projects covered in this book are included at the start of the relevant chapters.

Another option is to use a virtual machine (such as Cloudera's Quick-Start VM) that comes with all the Hadoop services preinstalled and configured.

The instructions that follow are suitable for Unix-based systems, including Mac OS X (which is not a production platform, but is fine for development).

Prerequisites

Make sure you have a suitable version of Java installed. You can check the Hadoop wiki to find which version you need. The following command confirms that Java was installed correctly:

```
% java -version
java version "1.7.0_25"
Java(TM) SE Runtime Environment (build 1.7.0_25-b15)
Java HotSpot(TM) 64-Bit Server VM (build 23.25-b01, mixed mode)
```

Installation

Start by deciding which user you'd like to run Hadoop as. For trying out Hadoop or developing Hadoop programs, you can run Hadoop on a single machine using your own user account.

Download a stable release, which is packaged as a gzipped tar file, from the Apache Hadoop releases page, and unpack it somewhere on your filesystem:

```
% tar xzf hadoop-x.y.z.tar.gz
```

Before you can run Hadoop, you need to tell it where Java is located on your system. If you have the `JAVA_HOME` environment variable set to point to a suitable Java installation, that will be used, and you don't have to configure anything further. (It is often set in a shell startup file, such as *~/.bash_profile* or *~/.bashrc*.) Otherwise, you can set the Java installation that Hadoop uses by editing *conf/hadoop-env.sh* and specifying the `JAVA_HOME` variable. For example, on my Mac, I changed the line to read:

```
export JAVA_HOME=/Library/Java/JavaVirtualMachines/jdk1.7.0_25.jdk/Contents/Home
```

to point to the installed version of Java.

It's very convenient to create an environment variable that points to the Hadoop installation directory (`HADOOP_HOME`, by convention) and to put the Hadoop binary directories on your command-line path. For example:

```
% export HADOOP_HOME=~/sw/hadoop-x.y.z
% export PATH=$PATH:$HADOOP_HOME/bin:$HADOOP_HOME/sbin
```

Note that the *sbin* directory contains the scripts for running Hadoop daemons, so it should be included if you plan to run the daemons on your local machine.

Check that Hadoop runs by typing:

```
% hadoop version
Hadoop 2.5.1
Subversion https://git-wip-us.apache.org/repos/asf/hadoop.git -r 2e18d179e4a8065
b6a9f29cf2de9451891265cce
Compiled by jenkins on 2014-09-05T23:11Z
Compiled with protoc 2.5.0
From source with checksum 6424fcab95bfff8337780a181ad7c78
This command was run using /Users/tom/sw/hadoop-2.5.1/share/hadoop/common/hadoop
-common-2.5.1.jar
```

Configuration

Each component in Hadoop is configured using an XML file. Common properties go in *core-site.xml*, and properties pertaining to HDFS, MapReduce, and YARN go into the appropriately named file: *hdfs-site.xml*, *mapred-site.xml*, and *yarn-site.xml*. These files are all located in the *etc/hadoop* subdirectory.

You can see the default settings for all the properties that are governed by these configuration files by looking in the *share/doc* directory hierarchy of your Hadoop installation for files called *core-default.xml*, *hdfs-default.xml*, *mapred-default.xml*, and *yarn-default.xml*.

Hadoop can be run in one of three modes:

Standalone (or local) mode
There are no daemons running and everything runs in a single JVM. Standalone mode is suitable for running MapReduce programs during development, since it is easy to test and debug them.

Pseudodistributed mode
The Hadoop daemons run on the local machine, thus simulating a cluster on a small scale.

Fully distributed mode
The Hadoop daemons run on a cluster of machines. This setup is described in Chapter 10.

To run Hadoop in a particular mode, you need to do two things: set the appropriate properties, and start the Hadoop daemons. Table A-1 shows the minimal set of properties to configure each mode. In standalone mode, the local filesystem and the local MapReduce job runner are used. In the distributed modes, the HDFS and YARN daemons are started, and MapReduce is configured to use YARN.

Table A-1. Key configuration properties for different modes

Component	Property	Standalone	Pseudodistributed	Fully distributed
Common	`fs.defaultFS`	`file:///` (default)	`hdfs://localhost/`	`hdfs://`*`namenode`*`/`
HDFS	`dfs.replication`	N/A	1	3 (default)
MapReduce	`mapreduce.framework.name`	`local` (default)	`yarn`	`yarn`
YARN	`yarn.resourcemanager.hostname`	N/A	`localhost`	*`resourcemanager`*
	`yarn.nodemanager.aux-services`	N/A	`mapreduce_shuffle`	`mapreduce_shuffle`

You can read more about configuration in "Hadoop Configuration" on page 292.

Standalone Mode

In standalone mode, there is no further action to take, since the default properties are set for standalone mode and there are no daemons to run.

Pseudodistributed Mode

In pseudodistributed mode, the configuration files should be created with the following contents and placed in the *etc/hadoop* directory. Alternatively, you can copy the *etc/hadoop* directory to another location, and then place the **-site.xml* configuration files there. The advantage of this approach is that it separates configuration settings from the installation files. If you do this, you need to set the `HADOOP_CONF_DIR` environment variable to the alternative location, or make sure you start the daemons with the `--config` option:

```
<?xml version="1.0"?>
<!-- core-site.xml -->
<configuration>
  <property>
    <name>fs.defaultFS</name>
    <value>hdfs://localhost/</value>
  </property>
</configuration>

<?xml version="1.0"?>
<!-- hdfs-site.xml -->
<configuration>
  <property>
    <name>dfs.replication</name>
    <value>1</value>
  </property>
</configuration>

<?xml version="1.0"?>
<!-- mapred-site.xml -->
<configuration>
  <property>
    <name>mapreduce.framework.name</name>
    <value>yarn</value>
  </property>
</configuration>

<?xml version="1.0"?>
<!-- yarn-site.xml -->
<configuration>
  <property>
    <name>yarn.resourcemanager.hostname</name>
    <value>localhost</value>
  </property>
  <property>
    <name>yarn.nodemanager.aux-services</name>
    <value>mapreduce_shuffle</value>
  </property>
</configuration>
```

Configuring SSH

In pseudodistributed mode, we have to start daemons, and to do that using the supplied scripts we need to have SSH installed. Hadoop doesn't actually distinguish between pseudodistributed and fully distributed modes; it merely starts daemons on the set of hosts in the cluster (defined by the *slaves* file) by SSHing to each host and starting a daemon process. Pseudodistributed mode is just a special case of fully distributed mode in which the (single) host is localhost, so we need to make sure that we can SSH to localhost and log in without having to enter a password.

First, make sure that SSH is installed and a server is running. On Ubuntu, for example, this is achieved with:

```
% sudo apt-get install ssh
```

On Mac OS X, make sure Remote Login (under System Preferences→Sharing) is enabled for the current user (or all users).

Then, to enable passwordless login, generate a new SSH key with an empty passphrase:

```
% ssh-keygen -t rsa -P '' -f ~/.ssh/id_rsa
% cat ~/.ssh/id_rsa.pub >> ~/.ssh/authorized_keys
```

You may also need to run *ssh-add* if you are running *ssh-agent*.

Test that you can connect with:

```
% ssh localhost
```

If successful, you should not have to type in a password.

Formatting the HDFS filesystem

Before HDFS can be used for the first time, the filesystem must be formatted. This is done by running the following command:

```
% hdfs namenode -format
```

Starting and stopping the daemons

To start the HDFS, YARN, and MapReduce daemons, type:

```
% start-dfs.sh
% start-yarn.sh
% mr-jobhistory-daemon.sh start historyserver
```

If you have placed configuration files outside the default *conf* directory, either export the `HADOOP_CONF_DIR` environment variable before running the scripts, or start the daemons with the `--config` option, which takes an absolute path to the configuration directory:

```
% start-dfs.sh --config path-to-config-directory
% start-yarn.sh --config path-to-config-directory
% mr-jobhistory-daemon.sh --config path-to-config-directory
  start historyserver
```

The following daemons will be started on your local machine: a namenode, a secondary namenode, a datanode (HDFS), a resource manager, a node manager (YARN), and a history server (MapReduce). You can check whether the daemons started successfully by looking at the logfiles in the *logs* directory (in the Hadoop installation directory) or by looking at the web UIs, at *http://localhost:50070/* for the namenode, *http://localhost:8088/* for the resource manager, and *http://localhost:19888/* for the history server. You can also use Java's `jps` command to see whether the processes are running.

Stopping the daemons is done as follows:

```
% mr-jobhistory-daemon.sh stop historyserver
% stop-yarn.sh
% stop-dfs.sh
```

Creating a user directory

Create a home directory for yourself by running the following:

```
% hadoop fs -mkdir -p /user/$USER
```

Fully Distributed Mode

Setting up a cluster of machines brings many additional considerations, so this mode is covered in Chapter 10.

APPENDIX B
Cloudera's Distribution Including Apache Hadoop

Cloudera's Distribution Including Apache Hadoop (hereafter *CDH*) is an integrated Apache Hadoop–based stack containing all the components needed for production, tested and packaged to work together. Cloudera makes the distribution available in a number of different formats: Linux packages, virtual machine images, tarballs, and tools for running CDH in the cloud. CDH is free, released under the Apache 2.0 license, and available at *http://www.cloudera.com/cdh*.

As of CDH 5, the following components are included, many of which are covered elsewhere in this book:

Apache Avro
: A cross-language data serialization library; includes rich data structures, a fast/compact binary format, and RPC

Apache Crunch
: A high-level Java API for writing data processing pipelines that can run on MapReduce or Spark

Apache DataFu (incubating)
: A library of useful statistical UDFs for doing large-scale analyses

Apache Flume
: Highly reliable, configurable streaming data collection

Apache Hadoop
: Highly scalable data storage (HDFS), resource management (YARN), and processing (MapReduce)

Apache HBase
: Column-oriented real-time database for random read/write access