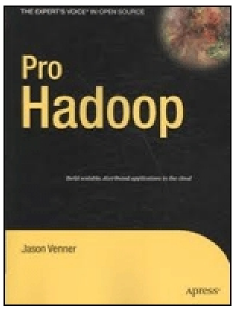


Pro Hadoop

by Jason Venner

Apress. (c) 2009. Copying Prohibited.

Chapter 3: The Basics of Multimachine Clusters

This chapter explains how to set up a multimachine cluster. You'll learn about the makeup of a cluster, the tools for managing clusters, and how to configure a cluster. Here, we'll walkthrough a simple cluster configuration, using the minimum HDFS setup necessary to bring up the cluster. Chapter 4 will go into the details for a high-usage HDFS.

The Makeup of a Cluster

A typical Hadoop Core cluster is made up of machines running a set of cooperating server processes. The machines in the cluster are not required to be homogeneous, and commonly they are not. The cluster machines may even have different CPU architectures and operating systems. But if the machines have similar processing power, memory, and disk bandwidth, cluster administration is a lot easier, because in that case, only one set of configuration files and runtime environments needs to be maintained and distributed.

Figure 3-1 illustrates a typical Hadoop cluster. A cluster will have one JobTracker server, one NameNode server, and one secondary NameNode server, and DataNodes and TaskTrackers. The JobTracker coordinates the activities of the TaskTrackers, and the NameNode manages the DataNodes.

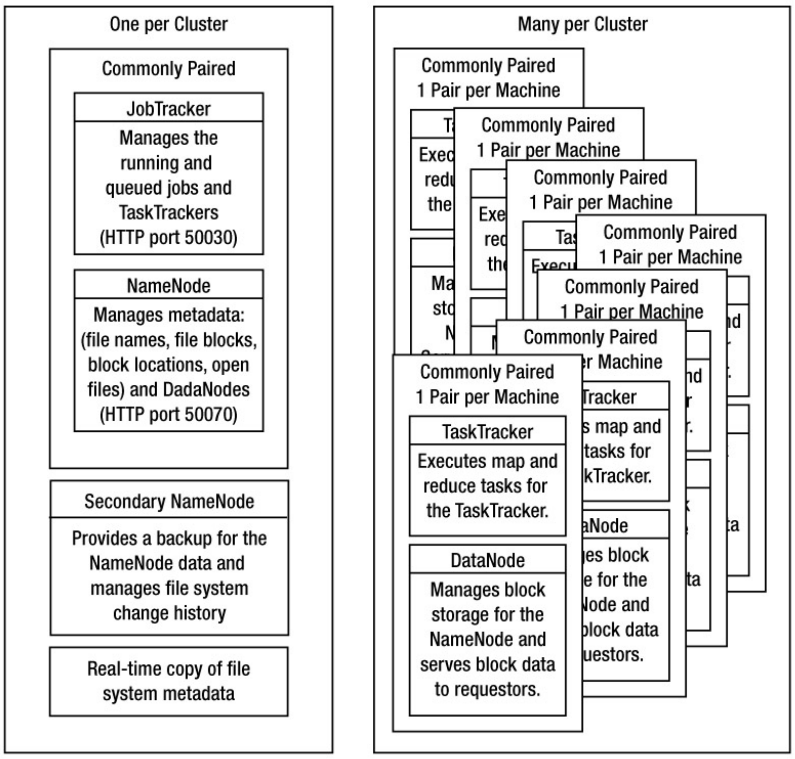

One per Cluster
Commonly Paired
JobTracker
Manages the running and queued jobs and TaskTrackers (HTTP port 50030)
NameNode
Manages metadata: (file names, file blocks, block locations, open files) and DadaNodes (HTTP port 50070)
Secondary NameNode
Provides a backup for the NameNode data and manages file system change history
Real-time copy of file system metadata
Many per Cluster
Commonly Paired 1 Pair per Machine
TaskTracker
Executes map and reduce tasks for the TaskTracker.
DataNode
Manages block storage for the NameNode and serves block data to requestors.

Figure 3-1: A typical Hadoop cluster

In the context of Hadoop, a node/machine running the TaskTracker or DataNode server is considered a *slave* node. It is common to have nodes that run both the TaskTracker and DataNode servers. The Hadoop server processes on the slave nodes are controlled by their respective masters, the JobTracker and NameNode servers.

Let's look at each of the server processes run by the machines in a cluster:

JobTracker: The JobTracker provides command and control for job management. It supplies the primary user interface to a MapReduce cluster. It also handles the distribution and management of tasks. There is one instance of this server running on a cluster. The machine running the JobTracker server is the MapReduce master.

TaskTracker: The TaskTracker provides execution services for the submitted jobs. Each TaskTracker manages the execution of tasks on an individual compute node in the MapReduce cluster. The JobTracker manages all of the TaskTracker processes. There is one instance of this server per compute node.

Note If your MapReduce jobs utilize external packages or services, it is very important that these external packages and services are identically configured across all of your TaskTracker machines. It is not uncommon for external JAR files to be required for the successful running of a task. If these JAR files differ in version or are absent, unexpected and difficult-to-diagnose errors occur.

NameNode: The NameNode provides metadata storage for the shared file system. The NameNode supplies the primary user interface to the HDFS. It also manages all of the metadata for the HDFS. There is one instance of this server running on a cluster. The metadata includes such critical information as the file directory structure and which DataNodes have copies of the data blocks that contain each file's data. The machine running the NameNode server process is the HDFS master.

Secondary NameNode: The secondary NameNode provides both file system metadata backup and metadata compaction. It supplies near real-time backup of the metadata for the NameNode. There is at least one instance of this server running on a cluster, ideally on a separate physical machine than the one running the NameNode. The secondary NameNode also merges the metadata change history, the edit log, into the NameNode's file system image.

Real-time backup of the NameNode data: Many installations configure the NameNode to store the file system metadata to multiple locations, where at least one of these locations resides on a separate physical machine. Other installations use a tool such as DRBD (`http://www.drbd.org/`) to replicate the host file system in near real time to a separate physical machine.

DataNode: The DataNode provides data storage services for the shared file system. Each DataNode supplies block storage services for the HDFS. The NameNode coordinates the storage and retrieval of the individual data blocks managed by a DataNode. There is one instance of this server process per HDFS storage node.

Balancer: During normal usage, the disk utilization on the DataNode machines may become uneven. This is particularly common if some DataNodes have less disk space available for use by HDFS. The Balancer moves data blocks between DataNodes to even out the per-DataNode available disk space. The Balancer will also rebalance the cluster as new DataNodes are added to an existing cluster. The Balancer is not a started automatically. It must be run by the user via the command `bin/hadoop balancer [-threshold<threshold>]`. The optional argument is the maximum amount of variance in disk space utilization between DataNodes for the cluster to be considered balanced. The default is 10%. As of Hadoop 0.19.0, this is not a configuration parameter.

These server processes are typically started once per cluster instance. DataNodes and TaskTrackers may be dynamically added and removed from a running cluster, as described in the next section.

All of these servers are implemented in Java and require at least Java version 1.6.

Cluster Administration Tools

The Hadoop Core installation provides a number of scripts in the `bin` subdirectory of the installation that are used to stop and start the entire cluster or various pieces of the cluster. There are also administrative scripts for the Hadoop Core servers. Table 3-1 lists the available scripts for administering clusters.

Table 3-1: Cluster Administration Scripts

Script	Description
`start-all.sh` and `stop-all.sh`	Start and stop the full set of Hadoop Core servers in the cluster.
`start-mapred.sh` and `stop-mapred.sh`	Start and stop just the MapReduce servers. These scripts start or stop only the JobTracker and TaskTracker nodes. The JobTracker is expected to run on the machine on which these scripts are executed.
`start-dfs.sh` and `stop-dfs.sh`	Start and stop the HDFS servers, in the same way as the `start-mapred.sh` and `stop-mapred` scripts manage the MapReduce servers.
`start-balancer.sh` and `stop-balancer.sh`	Start and stop the Balancer. The Balancer is expected to run on the machine on which these scripts are executed.
`hadoop-daemon.sh`	Starts or stops a single instance of a server on the current machine. The preceding start and stop scripts actually use the `hadoop-daemon.sh` script to start or stop the servers.
`hadoop-daemons.sh`	Starts or stops a set of servers on the relevant set of machines. This script is used by the start and stop scripts to start the DataNodes, TaskTrackers, and secondary NameNodes. This script will use the

	`hadoop-daemon.sh` script to start the servers on each specific machine in the set of machines on which it operates.
`hadoop-config.sh`	Used by the other scripts to load the Hadoop configuration.
`slaves.sh`	Runs its arguments as a command on each of the hosts listed in the `conf/slaves` file, collecting the output and presenting the output back the user, prefixed with the name of the host on which each output line originated.
`hadoop`	Provides command-level access to the services provided by the HDFS and MapReduce servers.
`rcc`	Provides services for creating RPC interfaces. This feature of Hadoop Core is expected to be discontinued. As of Hadoop 0.17, the Serialization service is the preferred method for handling external data structures.

The administrator has the option of starting or stopping the full set of Hadoop Core servers with the `start-all.sh` and `stop-all.sh` scripts. These scripts start all of the server processes on the cluster machines. The NameNode and JobTracker will be started or stopped on the machine on which the script is run, and DataNodes and TaskTracker nodes will be started on the configured slave machines. Any requested secondary NameNodes will also be started on configured machines.

Cluster Configuration

The Hadoop Core servers load their configuration from files in the `conf` directory of your Hadoop Core installation. As a general rule, identical copies of the configuration files are maintained in the `conf` directory of every machine in the cluster. The current default is to have one set of configuration files that provides the configuration for all of the server processes. It is also common to have the NameNode and the JobTracker servers on the same node, especially in smaller installations.

Note Many difficult-to-diagnose problems occur when the configuration files or the supporting runtime environments differ between TaskTracker nodes.

Hadoop Configuration Files

The configuration files fall into the following groups:

Hadoop Core configuration: The Hadoop Core is configured by two XML files: `hadoop-default.xml` and `hadoop-site.xml`. `hadoop-default.xml` provides reasonable defaults and comes with the Hadoop distribution. The configuration provided by this default file is suitable for a single machine instance and is the configuration used to run the examples in Chapters 1 and 2. `hadoop-site.xml` is where cluster-specific information is specified by the cluster administrator. In this chapter, we will walk through constructing a `hadoop-site.xml` file for a small cluster.

Slaves and masters: Two files are used by the startup and shutdown commands discussed in the previous section to start and stop the DataNode, TaskTracker, and secondary NameNode servers. The `slaves` file contains a list of hosts, one per line, that are to host DataNode and TaskTracker servers. The `masters` file contains a list of hosts, one per line, that are to host secondary NameNode servers. If the `start-all.sh` script is used, a DataNode and TaskTracker will be started on each host in the `slaves` file, and a secondary NameNode will be started on each host in the `masters` file. `start-mapred.sh` starts only the TaskTracker servers. If the `start-dfs.sh` script is used, DataNodes will be started on the hosts listed in `slaves`, and secondary NameNodes will be started on the hosts listed in `masters`.

Per-process runtime environment: The file `hadoop-env.sh` is responsible for tailoring the per-process environment. In particular, it includes the `JAVA_HOME` environment variable, which provides the JVM installation location. This file also offers a way to provide custom parameters for each of the servers. `hadoop-env.sh` is sourced by all of the Hadoop Core scripts provided in the `conf` directory of the installation.

Reporting: Hadoop Core may be configured to report detailed information about the activities on the cluster. Hadoop Core may report to a file, via Ganglia (`http://ganglia.info/`), which provides a framework for displaying graphical reports summarizing the activities of large clusters of machines. The file `hadoop-metrics.properties` controls the reporting. The default is to not report.

Hadoop Core Server Configuration

The `hadoop-default.xml` file defines more than 150 parameters, divided into six groups:

- Global properties
- Logging properties
- I/O properties
- File system properties
- MapReduce properties
- IPC properties

Note Some of the parameters listed in `hadoop-default.xml` may be modified on a per-job basis by setting alternate values using the `JobConf.set*` methods. The administrator may specify that a parameter is final by adding `<final>true</final>` to the parameter's declaration in the `hadoop-site.xml` file. In general, the modification of the server configuration parameters by a job have no effect on the servers.

It is customary to consider the `hadoop-default.xml` file to be read-only, and to make changes only to the `hadoop-site.xml` file. The framework will load configuration files in order, with the values defined in later files superseding those earlier definitions. The loading order is `hadoop-default.xml`, `hadoop-site.xml`, and then any user specified resources.

Note Values that have a `${text}` are replaced with the system property value or a previously defined value. The search order is system properties, then previously defined values.

Three critical parameters must be configured for any Hadoop cluster: `hadoop.tmp.dir`, `fs.default.name`, and `mapred.job.tracker`. Several other parameters are important to tune but not critical: `mapred.tasktracker.map.tasks.maximum`, `mapred.tasktracker.reduce.tasks.maximum`, `mapred.child.java.opts`, and `webinterface.private.actions`. The default values for these parameters are suitable for single-machine, single-CPU temporary use only. The following sections discuss the minimum set for cluster configuration. The parameters required for large-scale HDFS installations are covered in Chapters 4 and 5.

Note The `hadoop-default.xml` file includes some documentation on the parameters that control a cluster and a job, although some configuration parameters are not documented here.

Per-Machine Data

The `hadoop.tmp.dir` parameter is critical to configure. If it is not configured, data loss will occur. This parameter is poorly named. It informs the framework of the directory to use for all Hadoop Core server data storage, as follows:

- The NameNode will store file system metadata in a subdirectory.
- The DataNodes will store per-file blocks in a subdirectory.
- The TaskTrackers will store intermediate output in a subdirectory.
- The JobTracker will store per-job data in a subdirectory.
- The secondary NameNode will store the backup metadata in a subdirectory.

In the default configuration, the parameters listed in Table 3-2 use the value of `${hadoop.tmp.dir}` as the leading component of their paths. A high-performance cluster will have values tailored to minimize I/O contention on individual devices. To maximize performance, the I/O for these various functions needs to be distributed over multiple devices.

Table 3-2: Parameters That Use the hadoop.tmp.dir Value

Parameter	Description
`fs.checkpoint.dir`	Determines where on the local file system the secondary NameNode will store name data.
`dfs.name.dir`	Determines where on the local file system the NameNode metadata is stored. This may be a comma-or space-separated list of directories. All the provided directories are used for redundant storage. This is of critical importance and should be stored on a low-latency device with redundancy.
`dfs.client.buffer.dir`	Determines where on the local file system data to be written to HDFS is accumulated prior to transmission to the DataNodes. This directory will experience bulk I/O that has a short life.

`dfs.data.dir`	Determines where on the local file system a DataNode stores blocks. This may be a commaor space-separated list of directories. The data will be distributed among the directories. By default, HDFS replicates data storage blocks to multiple DataNodes. This directory will experience bulk I/O transactions.
`mapred.local.dir`	The local directory where TaskTracker stores intermediate output. This may be a comma-separated list of directories, preferably on different devices. I/O will be spread among the directories for increased performance. This directory will also experience bulk I/O that has a short life.
`mapred.system.dir`	The shared directory where the JobTracker stores control files. This value must be unique per JobTracker if multiple MapReduce clusters share a single HDFS.
`mapred.temp.dir`	A shared directory for temporary files. The default value of this setting is `${hadoop.tmp.dir}/mapred/temp}`.

The cluster administrator must pick a location or locations for these directories that provide the required I/O performance and the required reliability. For example, consider the `dfs.data.dir` parameter.

The HDFS-level redundant block storage reduces the requirements for highly reliable block storage for individual DataNodes. The directory specified by the `dfs.data.dir` parameter will experience bulk I/O transactions and should be optimized for maximum speed. There will be a large number of files and directories created, each file being an HDFS data block.

The default value for `dfs.data.dir` is `${hadoop.tmp.dir}/dfs/data`. If you don't change this default value for `${hadoop.tmp.dir}`, the HDFS data will be stored in `/tmp` and deleted by the system `/tmp` cleaning service. This causes interesting chaos for users when their HDFS file's data blocks start vanishing about a week after the file was created.

The framework will attempt to create the `hadoop.tmp.dir` directory and all of the subdirectories if they do not exist. The user that the relevant server processes are running as must have the required permissions to be able to create these directories and to add and remove files from them.

A perhaps ideal configuration would be for the NameNode to have a RAID 10 array for the `dfs.name.dir`, and for DataNodes and TaskTrackers to have RAID 0 arrays for `dfs.data.dir`,`mapred.local.dir`, and `dfs.client.buffer.dir`. This configuration assumes that HDFS is doing redundant block storage at the HDFS level.

Default Shared File System URI and NameNode Location for HDFS

The `fs.default.name` parameter is critical to configure. If it is not configured, there is no shared file system. The URI specified here informs the Hadoop Core framework of the default file system. The default value is `file:///`, which instructs the framework to use the local file system.

An example of an HDFS URI is `hdfs://NamenodeHost[:8020]/`. The file system protocol is `hdfs`, the host to contact for services is `NamenodeHost`, and the port to connect to is `8020`, which is the default port for HDFS. If the default 8020 port is used, the URI may be simplified as `hdfs://NamenodeHost/`. This value may be altered by individual jobs. You can choose an arbitrary port for the `hdfs` NameNode.

JobTracker Host and Port

The `mapred.job.tracker` parameter is critical to configure. If it is not configured, only a single machine will be used for task execution The URI specified in this parameter informs the Hadoop Core framework of the JobTracker's location. The default value is `local`, which indicates that no JobTracker server is to be run, and all tasks will be run from a single JVM.

The appropriate value for a cluster is `JobtrackerHost:8021`. The `JobtrackerHost` is the host on which the JobTracker server process will be run. This value may be altered by individual jobs.

Maximum Concurrent Map Tasks Per TaskTracker

The `mapred.tasktracker.map.tasks.maximum` parameter sets the maximum number of map tasks that may be run by a TaskTracker server process on a host at one time. The default value is 2. This value is read only when the TaskTracker is started. Changes made by a job will not be honored or persist.

This parameter should be tuned to ensure that the CPU resources of the TaskTracker nodes are fully utilized. If the machine hosts only the TaskTracker, it is common to set this value to the effective number of CPUs on the node. This may result in a large memory footprint, as each of the JVMs executing tasks will have a full memory allocation.

Many administrators set this value to 1 and require that individual jobs specify that the class `MultiThreadedMapRunner` is to be used via the `JobConf.setMapRunner(MultiThreadedMapRunner.class)` method, and specify the number of threads to use per map task. The default number of threads as of Hadoop 0.18.2 is 10, and this value may be altered by setting the number of threads via the following:

```
JobConf.set("mapred.map.multithreadedrunner.threads", threadCount);
```

This latter choice is preferred, as the number of threads may be set on a per-job basis, allowing the job to customize its CPU consumption.

The following sample snippet demonstrates a common pattern for per-job management of map task parallelism. The choice of 100 was made for demonstration purposes and is not suitable for a CPU-intensive map task.

```
if (conf.getInt("mapred.tasktracker.map.tasks.maximum", 2)==1) {
    conf.setMapRunnerClass(MultithreadedMapRunner.class);
    conf.setInt("mapred.map.multithreadedrunner.threads", 100);
}
```

Maximum Concurrent Reduce Tasks Per TaskTracker

The `mapred.tasktracker.reduce.tasks.maximum` parameter sets the maximum number of reduce tasks that may be run by an individual TaskTracker server at one time. Unlike in a map task, the output key ordering is critical for a reduce tasks, which precludes running multithreaded reduce tasks. This value also determines the number of parts in which your job output is placed. The default value, 2, is specified in the `conf/hadoop-default.xml` file.

Reduce tasks tend to be I/O bound, and it is not uncommon to have the per-machine maximum reduce task value set to 1 or 2. This value is utilized when the cluster is started. Changes made by a job will not be honored or persist.

JVM Options for the Task Virtual Machines

The `mapred.child.java.opts` parameter is commonly used to set a default maximum heap size for tasks. The default value is `-Xmx200m`. Most installation administrators immediately change this value to `-Xmx500m`. A significant and unexpected influence on this is the heap requirements (`io.sort.mb`), which by default will cause 100MB of space to be used for sorting.

During the run phase of a job, there may be up to `mapred.tasktracker.map.tasks.maximum` map tasks and `mapred.tasktracker.reduce.tasks.maximum` reduce tasks running simultaneously on each TaskTracker node, as well as the TaskTracker JVM. The node must have sufficient virtual memory to meet the memory requirements of all of the JVMs. JVMs have non-heap memory requirements; for simplicity, 20MB is assumed.

A cluster that sets the map task maximum to 1, the reduce task maximum to 8, and the JVM heap size to 500MB would need a minimum of (1 + 8 + 1) * (500+20) = 10 * 520 = 5200MB, or 5GB, of virtual memory available for the JVMs on each TaskTracker host. This 5GB value does not include memory for other processes or servers that may be running on the node.

The `mapred.child.java.opts` parameter is configurable by the job.

Enable Job Control Options on the Web Interfaces

Both the JobTracker and the NameNode provide a web interface for monitoring and control. By default, the JobTracker provides web service on `http://JobtrackerHost:50030` and the NameNode provides web service on `http://NamenodeHost:50070`. If the `webinterface.private.actions` parameter is set to `true`, the JobTracker web interface will add Kill This Job and Change Job Priority options to the per-job detail page. The default location of these additional options is the bottom-left corner of the page (so you usually need to scroll down the page to see them).

A Sample Cluster Configuration

In this section, we will walk through a simple configuration of a six-node Hadoop cluster. The cluster will be composed of six machines: `master01`, `slave01`, `slave02`, `slave03`, `slave04`, and `slave05`. The JobTracker and NameNode will reside on the machine `master01`, and a secondary NameNode will be placed on `slave01`. The DataNodes and TaskTrackers will be colocated on the same machines, and the nodes will be named `slave01` through `slave05`. Figure 3-2 shows this setup.

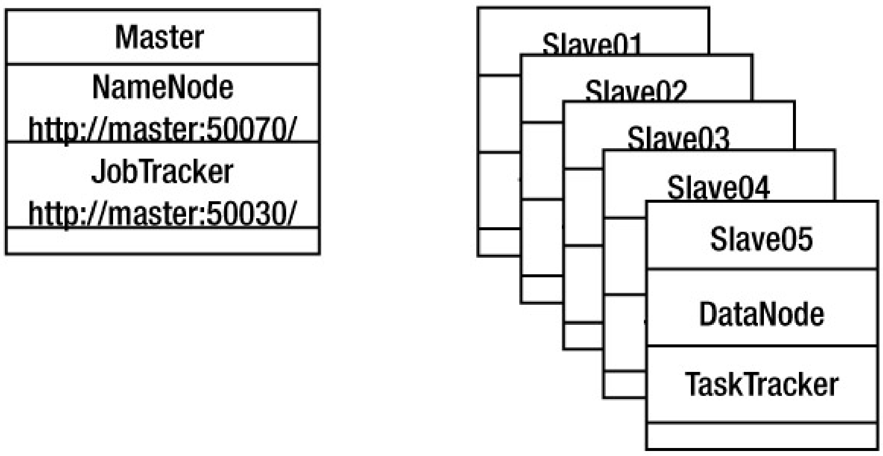


Figure 3-2: A simple six-node cluster

The standard machine configuration is usually an eight-CPU machine with 8GB of RAM, and a hardware RAID controller presenting a single partition to the operating system. This configuration is the favorite of IT departments. The single partition presentation is not ideal for Hadoop Core, as there is no opportunity to segregate the I/O for MapReduce and for HDFS.

Configuration Requirements

This configuration will require the customization of several files in the `conf` directory and compliance with some simple network requirements. The user that will own the Hadoop processes must also be determined.

Network Requirements

Hadoop Core uses Secure Shell (SSH) to launch the server processes on the slave nodes. Hadoop Core requires that passwordless SSH work between the master machines and all of the slave and secondary machines.

For example, for OpenSSH, you can generate an unencrypted key for the user that will own the Hadoop Core server processes. The user will need to have a directory, `~/.ssh`, that only the user has access permissions for, on all machines in the cluster. The following command generates a `dsa` key with an empty password in the file`~/.ssh/id_dsa`:

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

This command will generate two files in the`~/.ssh` directory: `id_dsa` and `id_dsa.pub`. The quotes in the command are a pair of single quote characters, side by side.

For each machine in the cluster, append the contents of th`~/.ssh/id_dsa.pub` file to the`~/.ssh/authorized_keys` file. If required, create the`~/.ssh` directory and the`~/.ssh/authorized_keys` file.

Execute the command `chmod og-rwx ~/.ssh` on each machine in the cluster.

You should now be able to run `bin/slaves.sh uptime` and receive the output of uptime from each of the machines listed in the `conf/slaves` file, as follows:

```
bin/slaves.sh uptime | sort
```

```
slave01: 21:18:41 up 47 days, 22:22, 9 users, load average: 1.22, 1.23, 1.26
. . .
slave06: 21:18:22 up 20 days, 11:59, 3 users, load average: 0.02, 0.09, 0.13
```

These slave servers will need to contact their specific master (either the NameNode or the JobTracker), and this will require that several ports in the low 50000 range be unblocked and available. Table 3-3 lists the default ports that must be unfiltered and available.

Table 3-3: Default Ports Used by Hadoop Core

Port	Setting	Description
50030	`mapred.job.tracker.http.address`	JobTracker administrative web GUI
50070	`dfs.http.address`	NameNode administrative web GUI
50010	`dfs.datanode.address`	DataNode control port (each DataNode listens on this port and registers it with the NameNode on startup)
50020	`dfs.datanode.ipc.address`	DataNode IPC port, used for block transfer
50060	`mapred.task.tracker.http.address`	Per TaskTracker web interface
50075	`dfs.datanode.http.address`	Per DataNode web interface
50090	`dfs.secondary.http.address`	Per secondary NameNode web interface
50470	`dfs.https.address`	NameNode web GUI via HTTPS
50475	`dfs.datanode.https.address`	Per DataNode web GUI via HTTPS

Note Hadoop Core uses a number of TCP ports in the low 50000 range. If other applications, such as Squid, are also using ports in this range, difficult-to-diagnose problems may occur.

Advanced Networking: Support for Multihomed Machines

In installations with more complex network topologies, it can become important to control which network interface is used by Hadoop for interprocess communications (IPC). Some installations have multiple network interfaces per machine. Hadoop provides two parameters to control this:

- `dfs.datanode.dns.interface`: If set, this parameter is the name of the network interface to be used for HDFS transactions to the DataNode. The IP address of this interface will be advertised by the DataNode as its contact address.
- `dfs.datanode.dns.nameserver`: If set, this parameter is the hostname or IP address of a machine to use to perform a reverse host lookup on the IP address associated with the specified network interface.

An example of a more complex network setup is for the machines in the Amazon cloud. Each Amazon cloud machine has two network interfaces: one for Internet access and one for intracloud traffic. It is helpful to set `dfs.datanode.dns.interface` to the name of the intracloud network interface.

Machine Configuration Requirements

In the simplest case, all of the machines in the cluster will be identically configured. They will have the same number of CPUs, the same amount of physical RAM, and the same disk capacity and configuration, with the same file system mount points. The same version of the JVM will be installed in the same location, and the Hadoop installation directory will be the same on all of the machines.

Hadoop Core maintains process ID files in the directory `/tmp`, by default. This directory must exist and be writable by the user that will own the Hadoop server processes. This directory is configurable by editing the `conf/hadoop-env.sh` file and uncommenting and optionally altering the setting for the environment variable `HADOOP_PID_DIR`.

Caution Remember that the parent of the directory to be used for `hadoop.tmp.dir` must exist and be writable by the Hadoop server user, or the `hadoop.tmp.dir` directory must exist and be writable by the Hadoop server user.

Configuration Files for the Sample Cluster

The examples provided in this section were run using the VMware images provided by Cloudera as part of its boot camp (`http://www.cloudera.com/hadoop-training-basic`). The boot camp VMware image 0.2 is used.

The hadoop-site.xml File

The `hadoop-site.xml` file contains the XML-based configuration data for our sample cluster. The source code for this file is provided with the rest of this book's downloadable code.

The partition `/hadoop` is used as the value for `hadoop.tmp.dir`, and is assumed to be on the hardware RAID partition.

```
<?xml version="1.0" encoding="UTF-8"?>
<?xml-stylesheet type="text/xsl" href="configuration.xsl"?>
```

```
<configuration>

    <property>
        <name>hadoop.tmp.dir</name>
        <value>/hadoop</value>
        <description>A base for other temporary directories. Set to a
            directory off of the user's home directory for the simple test.
        </description>
    </property>
```

The NameNode is configured by setting `fs.default.name` to `hdfs://master01`. The port of 8020 is implicit in the protocol declaration.

```
<property>
    <name>fs.default.name</name>
    <value>hdfs://master01</value>
    <description>The name of the default file system. A URI whose
        scheme and authority determine the FileSystem implementation. The
        uri's scheme determines the config property (fs.SCHEME.impl) naming
        the FileSystem implementation class. The uri's authority is used to
        determine the host, port, etc. for a filesystem. Pass in the hostname
        via the -Dhadoop.namenode=NAMENODE_HOST java option.
     </description>
</property>
```

The JobTracker is configured by setting `mapred.job.tracker` to `master01:8011`. The value for `mapred.job.tracker` is not interpreted as a URI, but rather as a `host:port` pair. The port must be specified explicitly.

```
<property>
    <name>mapred.job.tracker</name>
    <value>master01:8012</value>
    <description>The host and port that the MapReduce job tracker runs
        at. If "local", then jobs are run in-process as a single map
        and reduce task.
        Pass in the jobtracker hostname via the
        -Dhadoop.jobtracker=JOBTRACKER_HOST java option.
    </description>
</property>
```

For setting the maximum number of map and reduce tasks per TaskTracker, several assumptions are made. The first assumption is that the map tasks will be threaded, and the individual jobs will choose a thread count that optimizes CPU utilization. This results in a setting of 1 for `mapred.tasktracker.map.tasks.maximum`.

```
<property>
    <name>mapred.tasktracker.map.tasks.maximum
    </name>
    <value>1</value>
    <description>The maximum number of map tasks that will be run
        simultaneously by a task tracker.
    </description>
</property>
```

The DataNode also will run on the same machine; therefore, you must budget for CPU and memory resources. In the best of all worlds, the DataNode would use a different set of disks for I/O than the TaskTracker. The setting for the `mapred.tasktracker.reduce.tasks.maximum` parameter is 6. This leaves CPU and I/O available for the DataNode. It is possible that the reduce tasks jobs are CPU-bound, but generally, the shuffle phase is CPU-bound and the reduce phase is I/O bound. In a high-performance cluster, these parameters will be carefully tuned for specific jobs.

```
<property>
    <name>mapred.tasktracker.reduce.tasks.maximum
    </name>
    <value>6</value>
    <description>The maximum number of reduce tasks that will be run
        simultaneously by a task tracker.
    </description>
</property>
```

Very few MapReduce jobs will run in the 200MB default heap size specified for the JVMs. To alter this to a more reasonable default, `mapred.child.java.opts` is set to -`Xmx512m -server`. The `-server` configures the JVM for the

HotSpot JVM and provides other performance optimizations.

```
<property>
    <name>mapred.child.java.opts</name>
    <value>-Xmx512m -server</value>
    <description>Java opts for the task tracker child processes.
        The following symbol, if present, will be interpolated: @taskid@ is
        replaced by current TaskID. Any other occurrences of '@' will go
        unchanged.
        For example, to enable verbose gc logging to a file named
        for the taskid in /tmp and to set the heap maximum to be a gigabyte,
        pass a 'value' of:
        -Xmx1024m -verbose:gc -Xloggc:/tmp/@taskid@.gc
        The configuration variable mapred.child.ulimit can be used to control
        the maximum virtual memory of the child processes.
        Leave this unchanged from the default as io.sort.mb has been reduced for
        our test purposes.
     </description>
</property>
```

Finally, `webinterface.private.actions` is set to `true`, the recommended value for any cluster that doesn't require significant security.

```
    <property>
        <name>webinterface.private.actions</name>
        <value>true</value>
        <description> If set to true, the web interfaces of JT and NN may
            contain actions, such as kill job, delete file, etc., that should
            not be exposed to public. Enable this option if the interfaces
            are only reachable by those who have the right authorization.
            Enable this option if at all possible as it greatly simplifies
            debugging.
        </description>
    </property>

</configuration>
```

The slaves and masters Files

The `slaves` file contains five lines, each with one slave machine's hostname.

```
slave01
slave02
slave03
slave04
slave05
```

The `masters` file controls which machines run secondary NameNodes. The default configuration contains a single line containing `localhost`, which provides no real protection from machine or disk failure. It is a wise precaution to have a secondary NameNode on another machine. For this simple configuration example, the `masters` file has a single line with `slave01`.

```
slave01
```

The hadoop-metrics.properties File

The `hadoop-metrics.properties` file assumes that Ganglia is set up on the machine `master01`, and that some machine is set up with the Ganglia web interface and pulling data from `master01`. Installation and use of Ganglia are covered in Chapter 8.

```
# Configuration of the "dfs" context for null
#dfs.class=org.apache.hadoop.metrics.spi.NullContext

# Configuration of the "dfs" context for file
#dfs.class=org.apache.hadoop.metrics.file.FileContext
#dfs.period=10
#dfs.fileName=/tmp/dfsmetrics.log

# Configuration of the "dfs" context for ganglia
dfs.class=org.apache.hadoop.metrics.ganglia.GangliaContext
dfs.period=10
dfs.servers=master01:8649


# Configuration of the "mapred" context for null
# mapred.class=org.apache.hadoop.metrics.spi.NullContext

# Configuration of the "mapred" context for file
#mapred.class=org.apache.hadoop.metrics.file.FileContext
#mapred.period=10
#mapred.fileName=/tmp/mrmetrics.log

# Configuration of the "mapred" context for ganglia
mapred.class=org.apache.hadoop.metrics.ganglia.GangliaContext
mapred.period=10
mapred.servers=master01:8649


# Configuration of the "jvm" context for null
#jvm.class=org.apache.hadoop.metrics.spi.NullContext

# Configuration of the "jvm" context for file
#jvm.class=org.apache.hadoop.metrics.file.FileContext
#jvm.period=10
#jvm.fileName=/tmp/jvmmetrics.log

# Configuration of the "jvm" context for ganglia
jvm.class=org.apache.hadoop.metrics.ganglia.GangliaContext
jvm.period=10
jvm.servers=master01:8649
```

Distributing the Configuration

One of the reasons for requiring that all of the machines be essentially identical is that this greatly simplifies managing the cluster configuration. All of the core configuration files can be identical, which allows the use of the Unix `rsync` command to distribute the configuration files.

The command I like to use assumes that `HADOOP_HOME` is set correctly:

```
for a in `sort -u $HADOOP_HOME/conf/{slaves,masters}`; do ➥
rsync -e ssh -a --exclude 'logs/*' --exclude 'src/* ➥
--exclude 'docs/*' "${HADOOP_HOME}" ${a}:"${HADOOP_HOME}"; done
```

Note This command assumes that the current machine is not listed in the `slaves` or `masters` file. It's a very good test of the passwordless SSH configuration.

This command says that for each host that will run a server, distribute all of the Hadoop installation files except for the `src` (`--exclude 'src/*'`), logs (`--exclude 'logs/*'`), and docs (`--exclude 'docs/*'`). This ensures that all servers are running the same configuration files and the same Hadoop JARs. The `-e ssh` forces `rsync` to use SSH to establish the remote machine connections.

If your installation requires different configuration files on a per-machine basis, some other mechanism will be required to ensure consistency and correctness for the Hadoop installations and configuration files.

Verifying the Cluster Configuration

You should take a few steps to verify that that the cluster is installed and configured properly. At this point, you can assume that the Hadoop installation has been replicated to the same location across the cluster machines and that passwordless SSH is working. So, you should check the location of the JVM and make sure that `HADOOP_PID_DIR` can be written.

To verify that the JVM is in place and that `JAVA_HOME` is set correctly, run the following commands from the master machine:

```
master01% $for a in ➥
```

```
`sort -u "${HADOOP_HOME}"/conf/slaves ${HADOOP_HOME}"/conf/masters`; ➥
do echo -n $a " ";ssh $a -n ls -1 '"${JAVA_HOME}"/bin/java'; done
```

```
slave01 /usr/java/..../bin/java
slave02 /usr/java/..../bin/java
slave03 /usr/java/..../bin/java
slave04 /usr/java/..../bin/java
slave05 /usr/java/..../bin/java
```

```
master01% ls -1 "${JAVA_HOME}"/bin/java
```

```
/usr/java/..../bin/java
```

Every slave machine, as well as the local machine, should have an output line. If Java is not available on a machine in the expected location, install the JVM on that machine and set the `JAVA_HOME` environment variable to reflect the JVM installation directory.

The next item to verify is that the various required paths exist with the proper permissions or that the proper paths can be created. There are two directories to check: the directory specified as the `hadoop.tmp.dir`, in this case `/hadoop`, and the directory specified in the `conf/hadoop.env.sh` script for `HADOOP_PID_DIR`, in this case `/var/hadoop/pids`.

> **Caution** All of the commands in this section must be run from the master machine and as the user that will own the cluster servers. Failure to do this will invalidate the verification process and may cause the cluster startup to fail in complex ways. The shell environment is also expected to be set up, as detailed in Chapter 2, such that both `java` and the `bin` directory of the Hadoop installation are the first two components of the `PATH` environment variable. The user that owns the cluster processes will be referred to in some of the following text as `CLUSTER_USER`.

Execute the following command to verify that `HADOOP_PID_DIR` and `hadmp.tmp.dir` are writable.

```
master01% for a in ➥
`sort -u "${HADOOP_HOME}/conf/slaves" "${HADOOP_HOME}/conf/masters"` `hostname`; ➥
do echo "${a} "; touch /var/hadoop/pids/dummy; touch /hadoop/dummy; done
```

```
slave01
slave02
slave03
slave04
slave05
master01
```

The command must not have any errors. Any error message about being unable to create the file `dummy` must be corrected before the next step is attempted.

At this point, you have checked that the cluster installation and configuration are correct, and that passwordless SSH is enabled. It is time to format the HDFS file system and to start the HDFS and MapReduce services.

Formatting HDFS

The command to format HDFS is very simple. If there has been an existing HDFS file system with the same `hadoop.tmp.dir`, it is best to remove all traces of it with the Unix `rm` command before formatting a new file system.

```
hadoop namenode -format
```

```
08/12/21 19:45:32 INFO dfs.NameNode: STARTUP_MSG:
/************************************************************
STARTUP_MSG: Starting NameNode
STARTUP_MSG:   host = master01/127.0.0.1
STARTUP_MSG:   args = [-format]
STARTUP_MSG:   version = 0.18.2-dev
```

```
STARTUP_MSG:   build =  -r ; compiled by 'jason' on Sun Nov 16 20:16:42 PST 2008
************************************************************/

fs.FSNamesystem: fsOwner=jason,jason,lp
fs.FSNamesystem: supergroup=supergroup
fs.FSNamesystem: isPermissionEnabled=true
dfs.Storage: Image file of size 79 saved in 0 seconds.
dfs.Storage: Storage directory /hadoop/dfs/name has been successfully formatted.
dfs.NameNode: SHUTDOWN_MSG:
/************************************************************
SHUTDOWN_MSG: Shutting down NameNode at at/127.0.0.1
************************************************************/
```

Starting HDFS

Starting HDFS is also quite simple. It may be started with the MapReduce portion of the cluster via `start-all.sh`. Here, I'll show you how to start HDFS separately to demonstrate the commands, as it is common to separate the HDFS master and configuration from the MapReduce master and configuration.

```
master01% bin/start-dfs.sh
```

```
starting namenode, logging to ➥
/home/training/hadoop-0.19.0/bin/../logs/HT-namenode-master01.out
slave01: starting datanode, logging to ➥
/home/training/hadoop-0.19.0/bin/../logs/HT-datanode-slave01.out
slave02: starting datanode, logging to ➥
/home/training/hadoop-0.19.0/bin/../logs/HT-datanode-slave02.out
slave05: starting datanode, logging to ➥
/home/training/hadoop-0.19.0/bin/../logs/HT-datanode-slave05.out
slave03: starting datanode, logging to ➥
/home/training/hadoop-0.19.0/bin/../logs/HT-datanode-slave03.out
slave04: starting datanode, logging to ➥
/home/training/hadoop-0.19.0/bin/../logs/HT-datanode-slave04.out
slave01: starting secondarynamenode, logging to ➥
➥/home/training/hadoop-0.19.0/bin/../logs/HT-secondarynamenode-slave01.out
```

Your output should look similar to the preceding sample output. Some common reasons for failures are listed in the next section.

After roughly one minute, allowing the DataNodes time to start and connect to the NameNode, issue the command that reports on the status of the DataNodes:

```
hadoop dfsadmin -report
```

```
Total raw bytes: 5368709120 (5 GB)
Remaining raw bytes: 5368709120 (5 GB)
Used raw bytes: 0 (0 GB)
% used: 0.00%

Total effective bytes: 0 (0 KB)
Effective replication multiplier: Infinity
-------------------------------------------------
Datanodes available: 5

Name: 192.168.0.10:50010
State          : In Service
Total raw bytes: 1073741824 (1 GB)
Remaining raw bytes: 1073741824(1.04 GB)
Used raw bytes: 0 (2.18 GB)
% used: 0.00%
Last contact: Sun Dec 21 19:30:14 PST 2008

Name: 192.168.0.11:50010
State          : In Service
Total raw bytes: 1073741824 (1 GB)
```

```
Remaining raw bytes: 1073741824(1.04 GB)
Used raw bytes: 0 (2.18 GB)
% used: 0.00%
Last contact: Sun Dec 21 19:30:14 PST 2008


Name: 192.168.0.12:50010
State          : In Service
Total raw bytes: 1073741824 (1 GB)
Remaining raw bytes: 1073741824(1.04 GB)
Used raw bytes: 0 (2.18 GB)
% used: 0.00%
Last contact: Sun Dec 21 19:30:14 PST 2008


Name: 192.168.0.13:50010
State          : In Service
Total raw bytes: 1073741824 (1 GB)
Remaining raw bytes: 1073741824(1.04 GB)
Used raw bytes: 0 (2.18 GB)
% used: 0.00%
Last contact: Sun Dec 21 19:30:14 PST 2008


Name: 192.168.0.14:50010
State          : In Service
Total raw bytes: 1073741824 (1 GB)
Remaining raw bytes: 1073741824(1.04 GB)
Used raw bytes: 0 (2.18 GB)
% used: 0.00%
Last contact: Sun Dec 21 19:30:14 PST 2008
```

In the sample cluster configuration, five DataNodes should report. If there are not five DataNodes, or the HDFS reports that it is in safe mode, something has gone wrong. Detailed log messages will be available in the `logs` directory of the Hadoop installation on the slave that is not reporting. The log file will be named `hadoop-datanode-slave`*`XX`*`.log`.

The final test is to attempt to copy a file into HDFS, as follows:

```
bin/hadoop dfs -touchz my_first_file
bin/hadoop dfs -ls my_first_file.
```

```
-rw-r--r--   3 training supergroup          0 2009-04-03 01:57 ➥
/user/training/my_first_file
```

A zero-length file named `my_first_file` will be created in the `/user/`*`USERNAME`* directory, where *`USERNAME`* is the username of the user running the `touchz` command.

Correcting Errors

The most common errors should not occur if the verification steps detailed in the preceding sections completed with no errors. Here are some errors you might come across:

- `HADOOP_PID_DIR` is not writable by the `CLUSTER_USER`.
- The directory specified for `hadoop.tmp.dir` does not exist with full access permissions for `CLUSTER_USER`, or could not be created.
- The JVM may be missing or installed in a different location.
- The environment set up on login for `CLUSTER_USER` may not set up `JAVA_HOME` and `HADOOP_HOME` correctly.
- `HADOOP_HOME` is not in an identical location on all of the cluster machines.
- Passwordless SSH to some subset of the machines may not be working, or the master machine cannot connect to the

slave machines via SSH due to firewall or network topology reasons. This will be clear from the error response of the start command.

- The servers on the slave machines may not be able to connect to their respective master server due to firewall or network topology issues. If this is a problem, there will be a somewhat descriptive error message in the server (TaskTracker or DataNode) log file on the slave machine. Resolving network topology and firewall issues will require support from your local network administrator.

The most exotic error should not occur at this point, as your installation should be using the Hadoop Core default classpath. If there is an error message in a log file that indicates that Jetty could not start its web server, there is a nonvalidating XML parser in the classpath ahead of the validating XML parser that Hadoop Core supplies. This may be fixed by reordering the classpath or by explicitly setting the XML parser by setting a Java property. You can modify the `HADOOP_OPTS` environment variable to include this string:

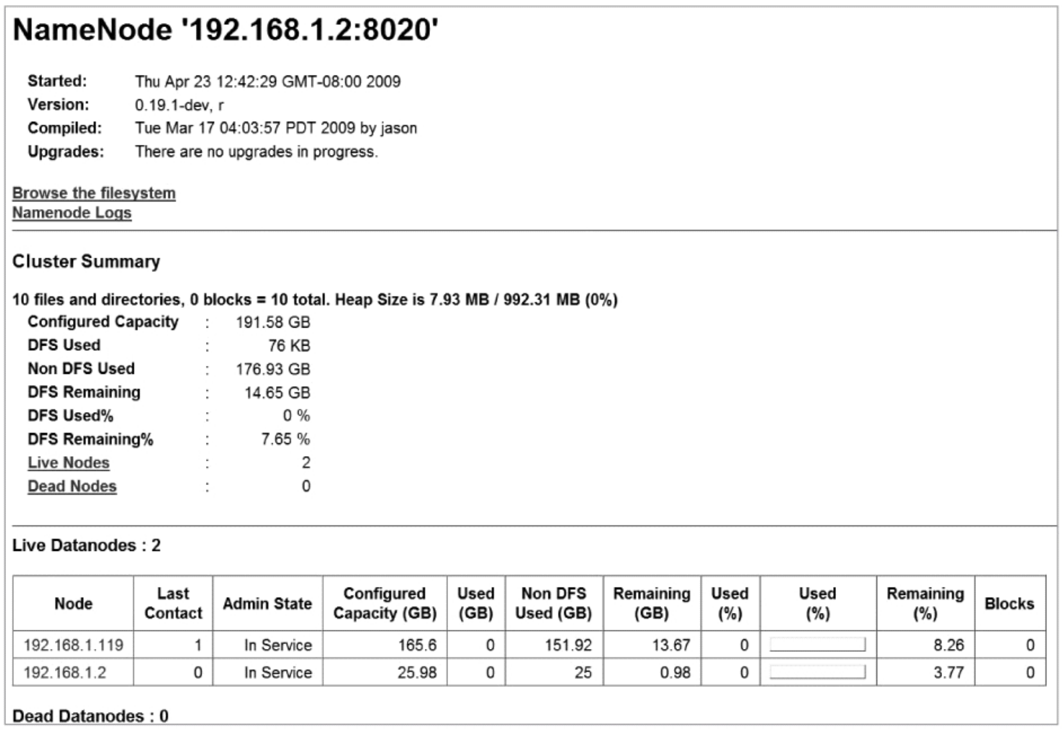
```
-Djavax.xml.parsers.SAXParserFactory=org.apache.xerces.jaxp.SAXParserFactoryImpl
```

Alternatively, you can alter the setting of `HADOOP_OPTS` in `conf/hadoop-env.sh`.

Caution It is highly recommended that you verify the termination of all Hadoop Core server processes across the cluster before attempting a restart. It is also strongly recommended that the `hadoop.tmp.dir` have its contents wiped on all cluster machines between attempts to start a new HDFS, and then the file system be reformatted as described in this chapter.

The Web Interface to HDFS

The NameNode web interface will be available via HTTP on port 50070, `http://master01:50070/`. As shown in Figure 3-3, this interface shows information about the cluster. It also provides links to browse the file system, view the NameNode logs, and to drill down to specific node information.



NameNode '192.168.1.2:8020'

Started:	Thu Apr 23 12:42:29 GMT-08:00 2009
Version:	0.19.1-dev, r
Compiled:	Tue Mar 17 04:03:57 PDT 2009 by jason
Upgrades:	There are no upgrades in progress.

Browse the filesystem
Namenode Logs

Cluster Summary

10 files and directories, 0 blocks = 10 total. Heap Size is 7.93 MB / 992.31 MB (0%)

Configured Capacity	:	191.58 GB
DFS Used	:	76 KB
Non DFS Used	:	176.93 GB
DFS Remaining	:	14.65 GB
DFS Used%	:	0 %
DFS Remaining%	:	7.65 %
Live Nodes	:	2
Dead Nodes	:	0

Live Datanodes : 2

Node	Last Contact	Admin State	Configured Capacity (GB)	Used (GB)	Non DFS Used (GB)	Remaining (GB)	Used (%)	Used (%)	Remaining (%)	Blocks
192.168.1.119	1	In Service	165.6	0	151.92	13.67	0		8.26	0
192.168.1.2	0	In Service	25.98	0	25	0.98	0		3.77	0

Dead Datanodes : 0

Figure 3-3: NameNode web interface

Starting MapReduce

The MapReduce portion of the cluster will be started by the `start-mapred.sh` command.

If there are any errors when starting the TaskTrackers, the detailed error message will be in the `logs` directory on the specific slave node of the failed TaskTracker. The log file will be `hadoop-slave`*`XX`*`-tasktracker.log`. The common reasons for failure are very similar to those for HDFS node startup failure.

For example, when I ran the example, on `slave01`, there was a process using the standard TaskTracker port that I was unaware of, and when the cluster was started, the TaskTracker did not start on `slave01`. I received the log message shown in Listing 3-1. This is a clear indication that some process is holding open a required port. The log file was in `${HADOOP_HOME}/logs/HT-tasktracker-slave01.log`.

Listing 3-1: Tasktracker Error Log Message Due to TCP Port Unavailability

```
2009-04-03 01:42:00,511 INFO org.apache.hadoop.mapred.TaskTracker: STARTUP_MSG:
/************************************************************
STARTUP_MSG: Starting TaskTracker
STARTUP_MSG:   host = slave01/192.168.1.121
STARTUP_MSG:   args = []
STARTUP_MSG:   version = 0.19.0
STARTUP_MSG:   build = ➥
https://svn.apache.org/repos/asf/hadoop/core/branches/branch-0.19 -r 713890; ➥
compiled by 'ndaley' on Fri Nov 14 03:12:29 UTC 2008
************************************************************/
INFO org.mortbay.http.HttpServer: Version Jetty/5.1.4
INFO org.mortbay.util.Credential: Checking Resource aliases
INFO org.mortbay.util.Container: Started ➥
org.mortbay.jetty.servlet.WebApplicationHandler@dc57db
INFO org.mortbay.util.Container: Started WebApplicationContext[/static,/static]
INFO org.mortbay.util.Container: Started ➥
org.mortbay.jetty.servlet.WebApplicationHandler@8e32e7
INFO org.mortbay.util.Container: Started WebApplicationContext[/logs,/logs]
INFO org.mortbay.util.Container: Started ➥
org.mortbay.jetty.servlet.WebApplicationHandler@15253d5
WARN org.mortbay.http.HttpContext: Can't reuse /tmp/Jetty__50060__, using ➥
/tmp/Jetty__50060___954083005982349324
INFO org.mortbay.util.Container: Started WebApplicationContext[/,/]
WARN org.mortbay.util.ThreadedServer: Failed to start: SocketListener0@0.0.0.0:50060
ERROR org.apache.hadoop.mapred.TaskTracker: Can not start task tracker because ➥
java.net.BindException: Address already in use
    at java.net.PlainSocketImpl.socketBind(Native Method)
    at java.net.PlainSocketImpl.bind(PlainSocketImpl.java:359)
    at java.net.ServerSocket.bind(ServerSocket.java:319)
    at java.net.ServerSocket.<init>(ServerSocket.java:185)
    at org.mortbay.util.ThreadedServer.newServerSocket(ThreadedServer.java:391)
    at org.mortbay.util.ThreadedServer.open(ThreadedServer.java:477)
    at org.mortbay.util.ThreadedServer.start(ThreadedServer.java:503)
    at org.mortbay.http.SocketListener.start(SocketListener.java:203)
    at org.mortbay.http.HttpServer.doStart(HttpServer.java:761)
    at org.mortbay.util.Container.start(Container.java:72)
    at org.apache.hadoop.http.HttpServer.start(HttpServer.java:321)
    at org.apache.hadoop.mapred.TaskTracker.<init>(TaskTracker.java:894)
    at org.apache.hadoop.mapred.TaskTracker.main(TaskTracker.java:2698)
```

Running a Test Job on the Cluster

To test the cluster configuration, let's run our old friend the Hadoop Core example `pi` (introduced in Chapter 1). Recall that this program takes two arguments: the number of maps and the number of samples. In this case, the cluster has five map slots, so you will set the number of maps to five. You have a couple of machines, so you can set the number of samples to a moderately large number—say 10,000. Your results should be similar to the following:

```
cd $HADOOP_HOME; hadoop jar hadoop-0.18.2-examples.jar 5 10000
```

```
Number of Maps = 5 Samples per Map = 10000
Wrote input for Map #0
Wrote input for Map #1
Wrote input for Map #2
Wrote input for Map #3
Wrote input for Map #4
Starting Job
```

```
jvm.JvmMetrics: Initializing JVM Metrics with processName=JobTracker, sessionId=
mapred.FileInputFormat: Total input paths to process : 5
mapred.FileInputFormat: Total input paths to process : 5
mapred.JobClient: Running job: job_local_0001
mapred.FileInputFormat: Total input paths to process : 5
mapred.FileInputFormat: Total input paths to process : 5
mapred.LocalJobRunner: Generated 9001 samples.
mapred.JobClient: map 100% reduce 0%
mapred.LocalJobRunner: Generated 9001 samples.
mapred.LocalJobRunner: Generated 9001 samples.
mapred.LocalJobRunner: Generated 9001 samples.
mapred.LocalJobRunner: Generated 9001 samples.
mapred.ReduceTask: Initiating final on-disk merge with 5 files
mapred.Merger: Merging 5 sorted segments
mapred.Merger: Down to the last merge-pass, with 5 ➥
segments left of total size: 190 bytes
mapred.LocalJobRunner: reduce > reduce
mapred.JobClient:  map 100% reduce 100%
mapred.JobClient: Job complete: job_local_0001
mapred.JobClient: Counters: 11
mapred.JobClient:   File Systems
mapred.JobClient:     Local bytes read=632904
mapred.JobClient:     Local bytes written=731468
mapred.JobClient:   Map-Reduce Framework
mapred.JobClient:     Reduce input groups=2
mapred.JobClient:     Combine output records=0
mapred.JobClient:     Map input records=5
mapred.JobClient:     Reduce output records=0
mapred.JobClient:     Map output bytes=160
mapred.JobClient:     Map input bytes=120

mapred.JobClient:     Combine input records=0
mapred.JobClient:     Map output records=10
mapred.JobClient:     Reduce input records=10
Job Finished in 1.807 seconds
Estimated value of PI is 3.14816
```

Summary

This chapter provided a simple walk-through of configuring a small Hadoop Core cluster. It did not discuss the tuning parameters required for a larger or a high-performance cluster. These parameters will be covered in Chapters 4 and 6.

For a multimachine cluster to run, the configuration must include the following:

- List of slave machines (`conf/slaves`)
- Network location of the JobTracker server (`mapred.job.tracker`)
- Network location of the NameNode server (`fs.default.name`)
- Persistent location on the cluster machines to store the data for HDFS (`hadoop.tmp.dir`)

You now have an understanding of what a master node is, what the NameNode and Job-Tracker servers do, and what DataNode and TaskTracker servers are. You may have even set up a multimachine cluster and run jobs over it. Go forth and grid compute.