第2章 初识Hadoop

本章内容

- Hadoop的结构组成
- 安装Hadoop及其3种工作模式：单机、伪分布和全分布
- 用于监控Hadoop安装的Web工具

本章将作为路线图来指导Hadoop的全程安装。如果在工作环境中，已经有人为你安装好了Hadoop集群，你也许会打算浏览一下本章，只去了解个人开发环境的安装，而跳过通信配置以及不同节点协同这些细节。

在2.1节讨论Hadoop的物理组件之后，在2.2节和2.3节我们将展开介绍集群安装，其中2.3节关注Hadoop的3种工作模式及其设定。在2.4节，你会了解基于Web界面进行集群监控的辅助工具。

2.1 Hadoop的构造模块

在前一章，我们已经讨论了分布式存储和分布式计算的概念。现在让我们来看Hadoop是如何实现这些思想的。在一个全配置的集群上，“运行Hadoop”意味着在网络分布的不同服务器上运行一组守护进程（daemons）。这些守护进程有特殊的角色，一些仅存在于单个服务器上，一些则运行在多个服务器上。它们包括：

- NameNode（名字节点）；[①]
- DataNode（数据节点）；
- Secondary NameNode（次名字节点）；
- JobTracker（作业跟踪节点）；
- TaskTracker（任务跟踪节点）；

我们将逐一讨论并定位它们在Hadoop中的作用。

2.1.1 NameNode

我们从NameNode开始讨论。也许会有人质疑，但我们认为它是Hadoop守护进程中最重要的

① 因为这些名词有指代的作用，故译文中仍用英文来表示。——译者注

一个。Hadoop在分布式计算与分布式存储中都采用了主/从（master/slave）结构。分布式存储系统被称为Hadoop文件系统，或简称为HDFS。NameNode位于HDFS的主端，它指导从端的DataNode执行底层的I/O任务。NameNode是HDFS的书记员，它跟踪文件如何被分割成文件块，而这些块又被哪些节点存储，以及分布式文件系统的整体运行状态是否正常。

运行NameNode消耗大量的内存和I/O资源。因此，为了减轻机器的负载，驻留NameNode的服务器通常不会存储用户数据或者执行MapReduce程序的计算任务。这意味着NameNode服务器不会同时是DataNode或者TaskTracker。

不过NameNode的重要性也带来了一个负面影响——Hadoop集群的单点失效。对于任何其他的守护进程，如果它们所驻留的节点发生软件或硬件失效，Hadoop集群很可能还会继续平稳运行，不然你还可以快速重启这个节点。但这样的方法并不适用于NameNode。

2.1.2 DataNode

每一个集群上的从节点都会驻留一个DataNode守护进程，来执行分布式文件系统的繁重工作——将HDFS数据块读取或者写入到本地文件系统的实际文件中。当希望对HDFS文件进行读写时，文件被分割为多个块，由NameNode告知客户端每个数据块驻留在哪个DataNode。客户端直接与DataNode守护进程通信，来处理与数据块相对应的本地文件。而后，DataNode会与其他DataNode进行通信，复制这些数据块以实现冗余。

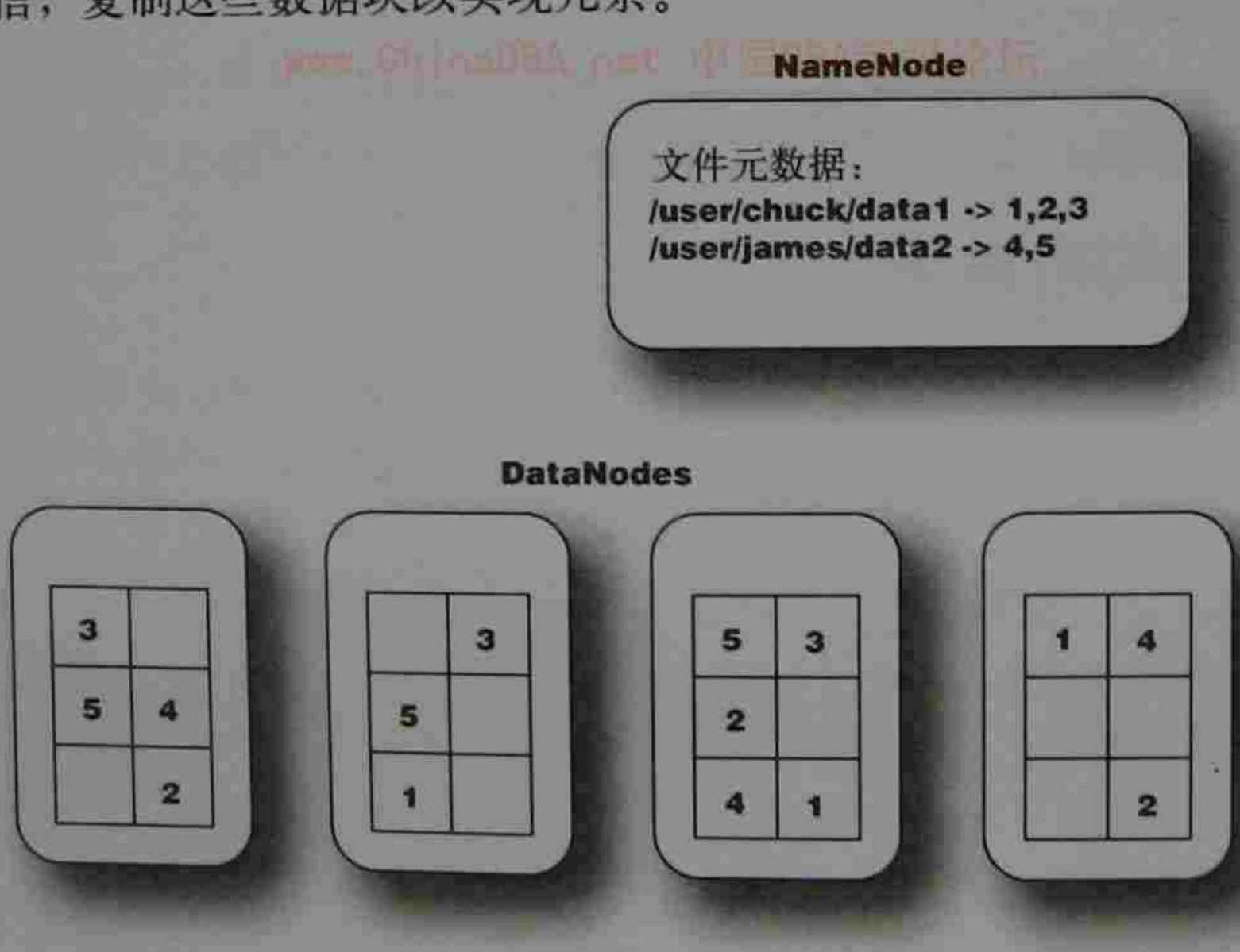

图2-1　NameNode/DataNode在HDFS中的交互。NameNode跟踪文件的元数据——描述系统中所包含的文件以及每个文件如何被分割为数据块。DataNode提供数据块的备份存储，并持续不断地向NameNode报告，以保持元数据为最新状态

图2-1说明了NameNode和DataNode的角色。图中显示了两个数据文件，一个位于目录/user/chuck/data1，另一个位于/user/james/data2。文件data1有3个数据块，表示为1、2和3，而文件data2由数据块4和5组成。这些文件的内容分散在几个DataNode上。这个示例中，每个数据块

有3个副本。例如，数据块1（属于文件data1）被复制在图中右侧的3个DataNode上。这确保了如果任何一个DataNode崩溃或者无法通过网络访问时，你仍然可以读取这些文件。

DataNode不断向NameNode报告。初始化时，每个DataNode将当前存储的数据块告知NameNode。在这个初始映射完成后，DataNode仍会不断地更新NameNode，为之提供本地修改的相关信息，同时接收指令创建、移动或删除本地磁盘上的数据块。

2.1.3 Secondary NameNode

Secondary NameNode（SNN）是一个用于监测HDFS集群状态的辅助守护进程。像NameNode一样，每个集群有一个SNN，它通常也独占一台服务器，该服务器不会运行其他的DataNode或TaskTracker守护进程。SNN与NameNode的不同在于它不接收或记录HDFS的任何实时变化。相反，它与NameNode通信，根据集群所配置的时间间隔获取HDFS元数据的快照。

如前所述，NameNode是Hadoop集群的单一故障点，而SNN的快照可以有助于减少停机的时间并降低数据丢失的风险。然而，NameNode的失效处理需要人工的干预，即手动地重新配置集群，将SNN用作主要的NameNode。在第8章，在讲述完集群管理的最佳实践经验之后，我们将讨论到恢复的过程。

2.1.4 JobTracker

JobTracker守护进程是应用程序和Hadoop之间的纽带。一旦提交代码到集群上，JobTracker就会确定执行计划，包括决定处理哪些文件、为不同的任务分配节点以及监控所有任务的运行。如果任务失败，JobTracker将自动重启任务，但所分配的节点可能会不同，同时受到预定义的重试次数限制。

每个Hadoop集群只有一个JobTracker守护进程。它通常运行在服务器集群的主节点上。

2.1.5 TaskTracker

与存储的守护进程一样，计算的守护进程也遵循主/从架构：JobTracker作为主节点，监测MapReduce作业的整个执行过程，同时，TaskTracker管理各个任务在每个从节点上的执行情况。图2-2说明了这种交互关系。

每个TaskTracker负责执行由JobTracker分配的单项任务。虽然每个从节点上仅有一个TaskTracker，但每个TaskTracker可以生成多个JVM（Java虚拟机）来并行地处理许多map或reduce任务。

TaskTracker的一个职责是持续不断地与JobTracker通信。如果JobTracker在指定的时间内没有收到来自TaskTracker的“心跳”，它会假定TaskTracker已经崩溃了，进而重新提交相应的任务到集群中的其他节点中。

讨论了Hadoop的各个守护进程之后，我们在图2-3中描绘了一个典型Hadoop集群的拓扑结构。

这种拓扑结构的特点是在主节点上运行NameNode和JobTracker的守护进程，并使用独立的节点运行SNN以防主节点失效。在小型集群中，SNN也可以驻留在某一个从节点上，而在大型集群中，连NameNode和JobTracker都会分别驻留在两台机器上。每个从节点均驻留一个DataNode和TaskTracker，从而在存储数据的同一节点上执行任务。

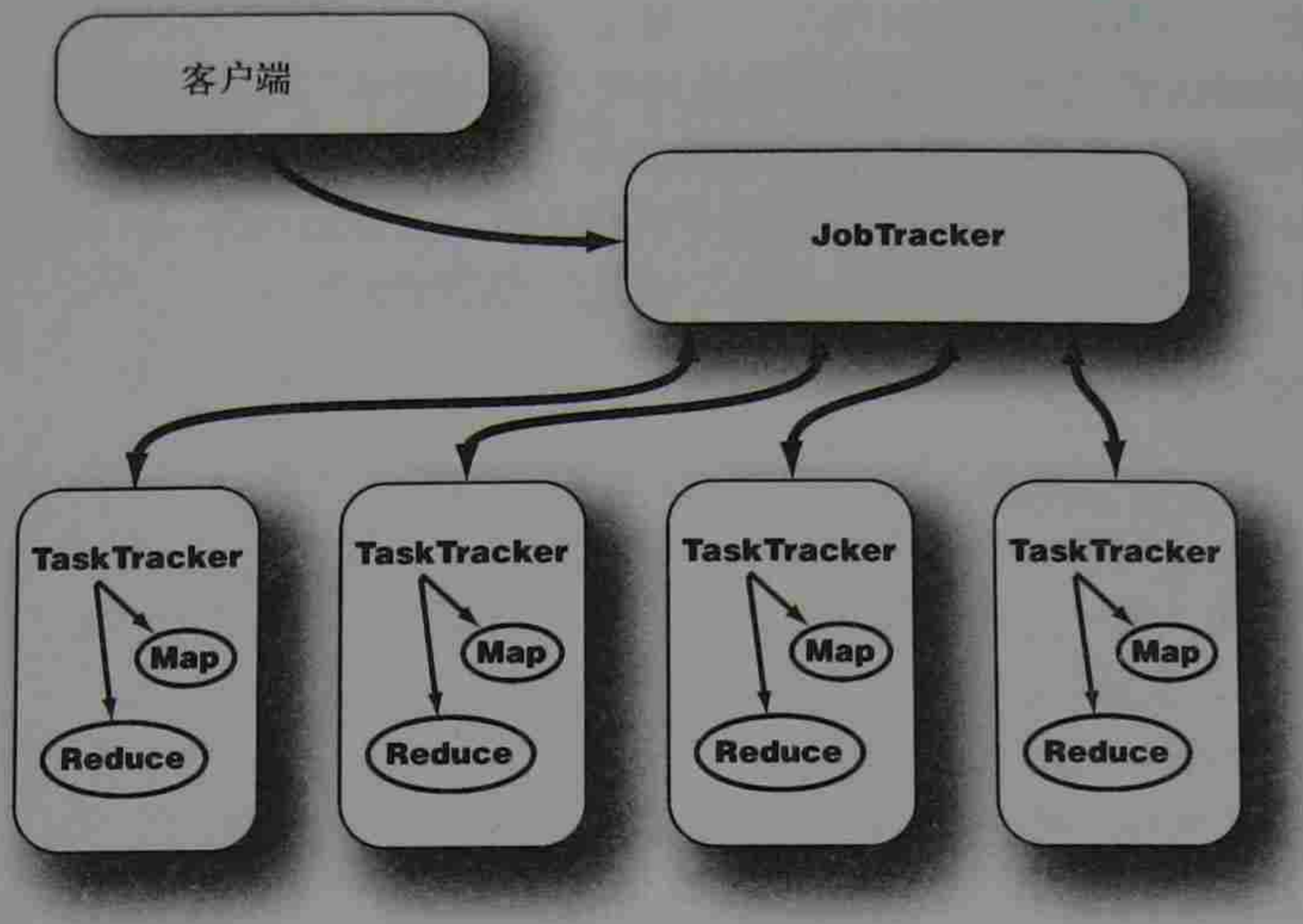

图2-2 JobTracker和TaskTracker的交互。当客户端调用JobTracker来启动一个数据处理作业时，JobTracker会将工作切分，并分配不同的map和reduce任务到集群中的每个TaskTracker上

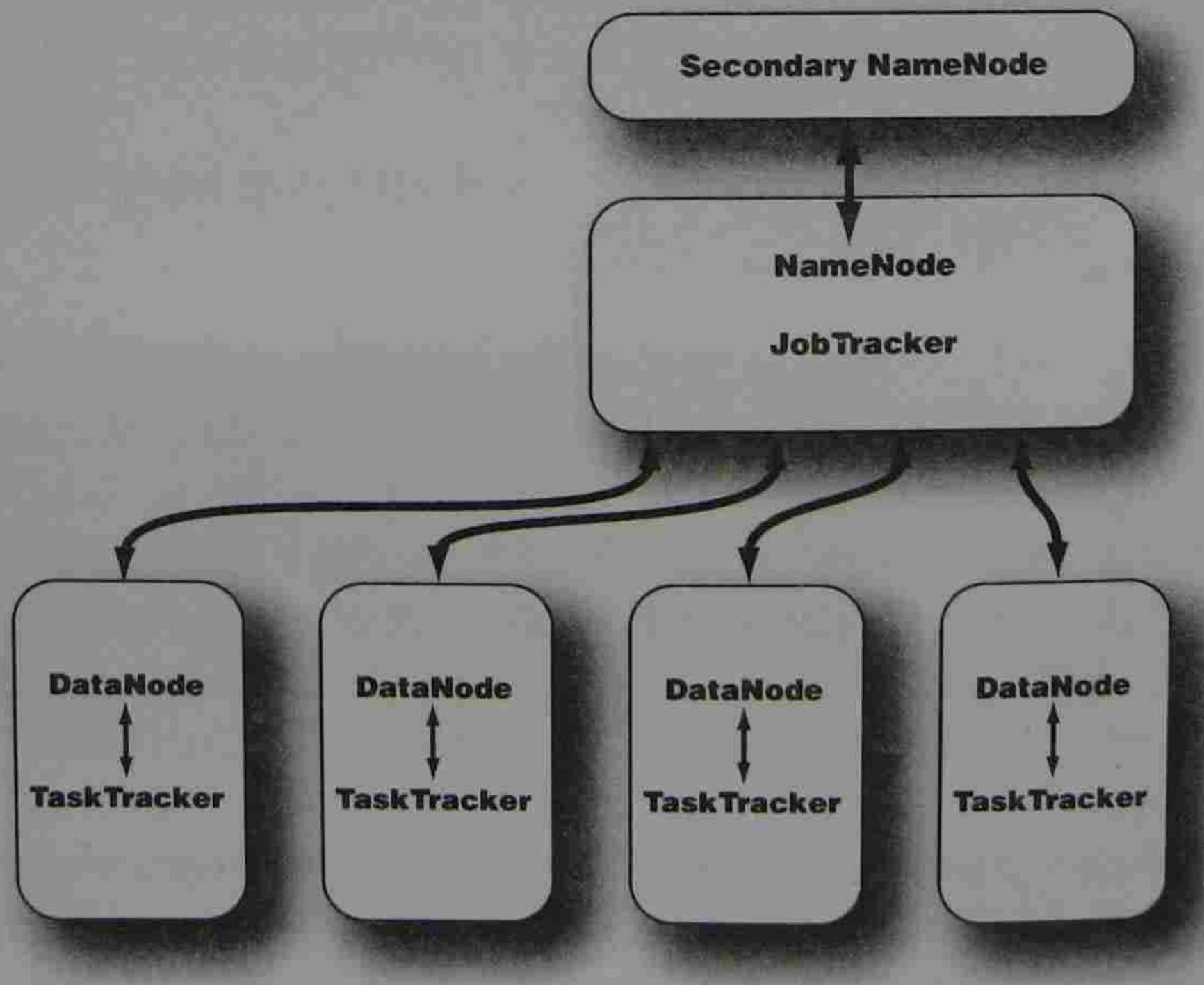

图2-3 一个典型Hadoop集群的拓扑图。这是一个主/从架构，其中NameNode和JobTracker为主端，DataNode和TaskTracker为从端

我们致力于来组建这样一个完整的Hadoop集群，这项工作首先得从建立主节点以及节点之间的控制通道开始。如果已经有了一个可用的Hadoop集群，你可以跳过下一节，它主要讲述如何在节点间建立SSH(Secure Shell)通道。另外，还有几个选项允许你仅仅使用一台机器来运行Hadoop，即单机或者伪分布模式。在程序开发时可以使用这些模式。在2.3节中我们将讨论如何把Hadoop配置成上述两种工作模式，或者如何配置成标准的集群安装模式（全分布模式）。

2.2 为 Hadoop 集群安装 SSH

安装一个Hadoop集群时，需要专门指定一个服务器作为主节点。如图2-3所示，这个服务器通常会驻留NameNode和JobTracker的守护进程。它也将作为一个基站，负责联络并激活所有从节点上的DataNode和TaskTracker守护进程。因此，我们需要为主节点定制一种手段，使它能够远程地访问到集群中的每个节点。

为此，Hadoop使用了无口令的（passphraseless）SSH协议。SSH采用标准的公钥加密来生成一对用户验证密钥——一个公钥、一个私钥。公钥被本地存储在集群的每个节点上，私钥则由主节点在试图访问远端节点时发送过来。结合这两段信息，目标机可以对这次登录尝试进行验证。

2.2.1 定义一个公共账号

从一个节点访问另一个节点这种说法一直被我们作为通用术语来使用，但其实更准确的描述应该是从一个节点的用户账号到目标机上的另一个用户账号。对于Hadoop，所有节点上的账号应该有相同的用户名（本书中我们使用hadoop-user），出于安全的考虑，我们建议你把这个账号设置为用户级别。它仅用于管理Hadoop集群。一旦集群的守护进程启动并运行，你就可以从其他的账号运行实际的MapReduce作业。

2.2.2 验证 SSH 安装

第一步是检查节点上是否安装了SSH。我们可以轻松地使用UNIX命令"which"来实现。

```
[hadoop-user@master]$ which ssh
/usr/bin/ssh

[hadoop-user@master]$ which sshd
/usr/bin/sshd

[hadoop-user@master]$ which ssh-keygen
/usr/bin/ssh-keygen
```

如果接收到类似这样的一个错误消息：

```
/usr/bin/which: no ssh in (/usr/bin:/bin:/usr/sbin...
```

你可以通过Linux安装包管理器安装OpenSSH（www.openssh.com）或者直接下载其源码。（但更好的办法是让你的系统管理员帮你去做。）

2.2.3 生成 SSH 密钥对

验证了SSH在集群上所有节点正确安装之后，我们使用主节点上的ssh-keygen来生成一个

RSA密钥对。务必要避免输入口令，否则，主节点每次试图访问其他节点时，你都得手动地输入这个口令。

```
[hadoop-user@master]$ ssh-keygen -t rsa
Generating public/private rsa key pair.
Enter file in which to save the key (/home/hadoop-user/.ssh/id_rsa):
Enter passphrase (empty for no passphrase):
Enter same passphrase again:
Your identification has been saved in /home/hadoop-user/.ssh/id_rsa.
Your public key has been saved in /home/hadoop-user/.ssh/id_rsa.pub.
```

生成密钥对之后，公钥的形式为

```
[hadoop-user@master]$ more /home/hadoop-user/.ssh/id_rsa.pub
ssh-rsa AAAAB3NzaC1yc2EAAAABIwAAAQEA1WS3RG8LrZH4zL2/1oYgkV1OmVc1Q2OO5vRi0Nd
K51Sy3wWpBVHx82F3x3ddoZQjBK3uvLMaDhXvncJG31JPfU7CTAfmtgINYv0kdUbDJq4TKG/fuO5q
J9CqHV71thN2M310gcJ0Y9YCN6grmsiWb2iMcXpy2pqg8UM3ZKApyIPx99O1vREWm+4moFTg
YwIl5be23ZCyxNjgZFWk5MR1T1p1TxB68jqNbPQtU7fIafS7Sasy7h4eyIy7cbLh8x0/V4/mcQsY
5dvReitNvFVte6on18YdmnMpAh6nwCvog3UeWWJjVZTEBFkTZuV1i9HeYHxpm1wAzcnf7az78jT
IRQ== hadoop-user@master
```

下面我们需要把这个公钥分布到集群中。

2.2.4　将公钥分布并登录验证

尽管有些烦琐，仍需逐一将公钥复制到主节点以及每个从节点上。

```
[hadoop-user@master]$ scp ~/.ssh/id_rsa.pub hadoop-user@target:~/master_key
```

手动登录到目标节点，并设置主节点的密钥为授权密钥（如果还有其他授权密钥，则把主节点密钥追加到授权密钥列表中）：

```
[hadoop-user@target]$ mkdir ~/.ssh
[hadoop-user@target]$ chmod 700 ~/.ssh
[hadoop-user@target]$ mv ~/master_key ~/.ssh/authorized_keys
[hadoop-user@target]$ chmod 600 ~/.ssh/authorized_keys
```

生成该密钥之后，可以尝试从主节点登录到目标节点来验证它的正确性。

```
[hadoop-user@master]$ ssh target
The authenticity of host 'target (xxx.xxx.xxx.xxx)' can't be established.
RSA key fingerprint is 72:31:d8:1b:11:36:43:52:56:11:77:a4:ec:82:03:1d.
Are you sure you want to continue connecting (yes/no)? yes
Warning: Permanently added 'target' (RSA) to the list of known hosts.
Last login: Sun Jan 4 15:32:22 2009 from master
```

一旦目标节点确认了对主节点的授权，以后再登录就不会出现这些提示，将显示为

```
[hadoop-user@master]$ ssh target
Last login: Sun Jan 4 15:32:49 2009 from master
```

现在，我们已经有了在集群上运行Hadoop的基础。下面来讨论各种Hadoop模式，在项目中你可能会用到它们。

2.3　运行 Hadoop

在运行Hadoop之前需要做一些配置。让我们仔细看看Hadoop的配置目录：

```
[hadoop-user@master]$ cd $HADOOP_HOME
[hadoop-user@master]$ ls -l conf/
total 100
-rw-rw-r-- 1 hadoop-user hadoop 2065 Dec 1 10:07 capacity-scheduler.xml
-rw-rw-r-- 1 hadoop-user hadoop 535 Dec 1 10:07 configuration.xsl
-rw-rw-r-- 1 hadoop-user hadoop 49456 Dec 1 10:07 hadoop-default.xml
-rwxrwxr-x 1 hadoop-user hadoop 2314 Jan 8 17:01 hadoop-env.sh
-rw-rw-r-- 1 hadoop-user hadoop 2234 Jan 2 15:29 hadoop-site.xml
-rw-rw-r-- 1 hadoop-user hadoop 2815 Dec 1 10:07 log4j.properties
-rw-rw-r-- 1 hadoop-user hadoop 28 Jan 2 15:29 masters
-rw-rw-r-- 1 hadoop-user hadoop 84 Jan 2 15:29 slaves
-rw-rw-r-- 1 hadoop-user hadoop 401 Dec 1 10:07 sslinfo.xml.example
```

需要做的第一件事是指定包括主节点在内所有节点上Java的位置，即在hadoop-env.sh中定义JAVA_HOME环境变量使之指向Java安装目录。在服务器上，我们将其指定为

```
export JAVA_HOME=/usr/share/jdk
```

(如果你做过了第1章的例子，这一步其实已经完成。)

在hadoop-env.sh文件中还包含定义Hadoop环境的其他变量，但JAVA_HOME是唯一在开始时需要做修正的。其他变量在默认设置下通常都可以很好地工作。你可以在逐渐熟悉Hadoop之后，通过修改这个文件来做个性化的设置（如日志目录的位置、Java类所在的目录，等等）。

Hadoop的设置主要包含在XML配置文件中。在0.20版以前，它们是hadoop-default.xml和hadoop-site.xml。顾名思义，hadoop-default.xml中包含了Hadoop会使用的默认设置，除非这些设置在hadoop-site.xml中被显式地覆盖。因此，在实际操作中你只需要处理hadoop-site.xml。在版本0.20中，这个文件被分离成3个XML文件：core-site.xml，hdfs-site.xml与mapred-site.xml。这次重构更好地对应了它们所控制的Hadoop子系统。在本章的其余部分，我们通常会点明是3个文件中的哪一个被用于配置的调整。如果你使用的是Hadoop的早期版本，请记住，所有这些配置都在hadoop-site.xml中修改。

下面我们将进一步描述Hadoop的不同工作模式并给出其配置文件的示例。

2.3.1 本地（单机）模式

单机模式是Hadoop的默认模式。当首次解压Hadoop的源码包时，Hadoop无法了解硬件安装环境，便保守地选择了最小配置。在这种默认模式下所有3个XML文件（或者0.20版本之前的hadoop-site.xml）均为空。

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

<!-- Put site-specific property overrides in this file. -->

<configuration>

</configuration>
```

当配置文件为空时，Hadoop会完全运行在本地。因为不需要与其他节点交互，单机模式就不使用HDFS，也不加载任何Hadoop的守护进程。该模式主要用于开发调试MapReduce程序的应用

逻辑，而不会与守护进程交互，避免引起额外的复杂性。在运行第1章中的MapReduce程序时，采用的就是单机模式。

2.3.2　伪分布模式

伪分布模式在“单节点集群”上运行Hadoop，其中所有的守护进程都运行在同一台机器上。该模式在单机模式之上增加了代码调试功能，允许你检查内存使用情况、HDFS输入输出，以及其他的守护进程交互。代码清单2-1给出了该模式下用于配置单个服务器的简单XML文件。

代码清单2-1　伪分布模式下3个配置文件的示例

```
core-site.xml
<?xml version="1.0"?>
<?xml-stylesheet type="text/xsl" href="configuration.xsl"?>

<!-- Put site-specific property overrides in this file. -->

<configuration>

<property>
 <name>fs.default.name</name>
 <value>hdfs://localhost:9000</value>
 <description>The name of the default file system. A URI whose
 scheme and authority determine the FileSystem implementation.
 </description>
</property>

</configuration>

mapred-site.xml
<?xml version="1.0"?>
<?xml-stylesheet type="text/xsl" href="configuration.xsl"?>

<!-- Put site-specific property overrides in this file. -->

<configuration>

<property>
 <name>mapred.job.tracker</name>
 <value>localhost:9001</value>
 <description>The host and port that the MapReduce job tracker runs
 at.</description>
</property>

</configuration>

hdfs-site.xml
<?xml version="1.0"?>
<?xml-stylesheet type="text/xsl" href="configuration.xsl"?>

<!-- Put site-specific property overrides in this file. -->

<configuration>

<property>
 <name>dfs.replication</name>
 <value>1</value>
```

```
  <description>The actual number of replications can be specified when the
  file is created.</description>
 </property>

 </configuration>
```

我们在core-site.xml和mapred-site.xml中分别指定了NameNode和JobTracker的主机名与端口。在hdfs-site.xml中指定了HDFS的默认副本数，因为仅运行在一个节点上，这里副本数为1。我们还需要在文件masters中指定SNN的位置，并在文件slaves中指定从节点的位置。

```
[hadoop-user@master]$ cat masters
localhost
[hadoop-user@master]$ cat slaves
localhost
```

虽然所有的守护进程都运行在同一节点上，它们仍然像分布在集群中一样，彼此通过相同的SSH协议进行通信。在2.2节中已经详细说明了SSH通道的安装，但对于单节点的操作，仅需检查一下机器是否允许对自己运行ssh。

```
[hadoop-user@master]$ ssh localhost
```

如果允许，说明一切正常。否则需要用两行命令来安装。

```
[hadoop-user@master]$ ssh-keygen -t dsa -P '' -f ~/.ssh/id_dsa
[hadoop-user@master]$ cat ~/.ssh/id_dsa.pub >> ~/.ssh/authorized_keys
```

现在几乎已经准备好了启动Hadoop。但还是先要输入一个命令来格式化HDFS：

```
[hadoop-user@master]$ bin/hadoop namenode -format
```

我们现在可以使用start-all.sh脚本装载守护进程，然后用Java的jps命令列出所有守护进程来验证安装成功。

```
[hadoop-user@master]$ bin/start-all.sh
[hadoop-user@master]$ jps
26893 Jps
26832 TaskTracker
26620 SecondaryNameNode
26333 NameNode
26484 DataNode
26703 JobTracker
```

当结束Hadoop时，可以通过stop-all.sh脚本来关闭Hadoop的守护进程。

```
[hadoop-user@master]$ bin/stop-all.sh
```

单机模式和伪分布模式均用于开发与调试的目的。真实Hadoop集群的运行采用的是第三种模式，即全分布模式。

2.3.3 全分布模式

在不断强调分布式存储和分布式计算的好处之后，是时候来建立一个完全的集群了。下面的讨论将使用如下的服务器名称。

- master——集群的主节点，驻留NameNode和JobTracker守护进程
- backup——驻留SNN守护进程的节点

- hadoop1, hadoop2, hadoop3, ...——集群的从节点，驻留DataNode和TaskTracker守护进程

沿用先前的命名习惯，在伪分布模式配置文件（代码清单2-1）的基础上修改得到代码清单2-2。

代码清单2-2　全分布式模式的配置文件样例

core-site.xml

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

<!-- Put site-specific property overrides in this file. -->

<configuration>

<property>
 <name>fs.default.name</name>
 <value>hdfs://master:9000</value>
 <description>The name of the default file system. A URI whose
 scheme and authority determine the FileSystem implementation.
 </description>
</property>

</configuration>
```

❶ 定位文件系统的NameNode

mapred-site.xml

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

<!-- Put site-specific property overrides in this file. -->

<configuration>
<property>
 <name>mapred.job.tracker</name>
 <value>master:9001</value>
 <description>The host and port that the MapReduce job tracker runs
 at.</description>
</property>

</configuration>
```

❷ 定位JobTracker所在主节点

hdfs-site.xml

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

<!-- Put site-specific property overrides in this file. -->

<configuration>

<property>
 <name>dfs.replication</name>
 <value>3</value>
 <description>The actual number of replications can be specified when the
 file is created.</description>
</property>

</configuration>
```

❸ 增大HDFS备份参数

主要不同在于以下两点。

- 明确地声明了NameNode❶和JobTracker❷守护进程所在的主机名。
- 增大了HDFS的备份参数以发挥分布式存储的优势❸。回忆一下，数据通过在HDFS上复

制可以提高可用性与可靠性。

我们还需要更新masters和slaves文件来指定其他守护进程的位置。

```
[hadoop-user@master]$ cat masters
backup
[hadoop-user@master]$ cat slaves
hadoop1
hadoop2
hadoop3
...
```

在将这些文件复制到集群上的所有节点之后，一定要格式化HDFS以准备好存储数据：

```
[hadoop-user@master]$ bin/hadoop namenode-format
```

现在可以启动Hadoop的守护进程：

```
[hadoop-user@master]$ bin/start-all.sh
```

并验证节点正在运行被指派的任务。

```
[hadoop-user@master]$ jps
30879 JobTracker
30717 NameNode
30965 Jps
[hadoop-user@backup]$ jps
2099 Jps
1679 SecondaryNameNode
[hadoop-user@hadoop1]$ jps
7101 TaskTracker
7617 Jps
6988 DataNode
```

一个可用的集群终于建成了！

模式之间的切换

我发现有一个技巧在开始使用Hadoop时是很有用的，就是使用符号链接而不是不断编辑XML文件来切换Hadoop的模式。为了做到这一点， 需要为每种模式分别生成一个配置目录，并相应地放入恰当版本的XML文件。下面是目录列表的示例：

```
[hadoop@hadoop_master hadoop]$ ls -l
total 4884
drwxr-xr-x 2 hadoop-user hadoop 4096 Nov 26 17:36 bin
-rw-rw-r-- 1 hadoop-user hadoop 57430 Nov 13 19:09 build.xml
drwxr-xr-x 4 hadoop-user hadoop 4096 Nov 13 19:14 c++
-rw-rw-r-- 1 hadoop-user hadoop 287046 Nov 13 19:09 CHANGES.txt
lrwxrwxrwx 1 hadoop-user hadoop 12 Jan 5 16:06 conf -> conf.cluster
drwxr-xr-x 2 hadoop-user hadoop 4096 Jan 8 17:05 conf.cluster
drwxr-xr-x 2 hadoop-user hadoop 4096 Jan 2 15:07 conf.pseudo
drwxr-xr-x 2 hadoop-user hadoop 4096 Dec 1 10:10 conf.standalone
drwxr-xr-x 12 hadoop-user hadoop 4096 Nov 13 19:09 contrib
drwxrwxr-x 5 hadoop-user hadoop 4096 Jan 2 09:28 datastore
drwxr-xr-x 6 hadoop-user hadoop 4096 Nov 26 17:36 docs
...
```

然后，就可以使用Linux的ln命令（例如 ln -s conf.cluster conf）在不同配置之间切换。这个技巧还有助于临时把一个节点从集群中分离出来，从而通过伪分布模式来调试一个MapReduce程序，但需要确保这些模式在HDFS上有不同的文件存储位置，并且在改变配置之前还应该停止所有的守护进程。

既然我们已经通览了建立并运行一个Hadoop集群的所有配置，下面将介绍用于集群状态基本监控的Web用户界面。

2.4　基于Web的集群用户界面

在讨论了Hadoop的工作模式之后，我们现在介绍Hadoop中用于监控集群健康状态的Web界面。与搜寻日志和目录相比，通过浏览器的界面，我们可以更快地获得所需的信息。

NameNode通过端口50070提供常规报告，描绘集群上HDFS的状态视图。图2-4显示了一个以两节点集群为例的报告。通过这个界面，可以通览文件系统，检查集群每个DataNode的状态，并详细查看Hadoop守护进程的日志来判断集群当前运行是否正确。

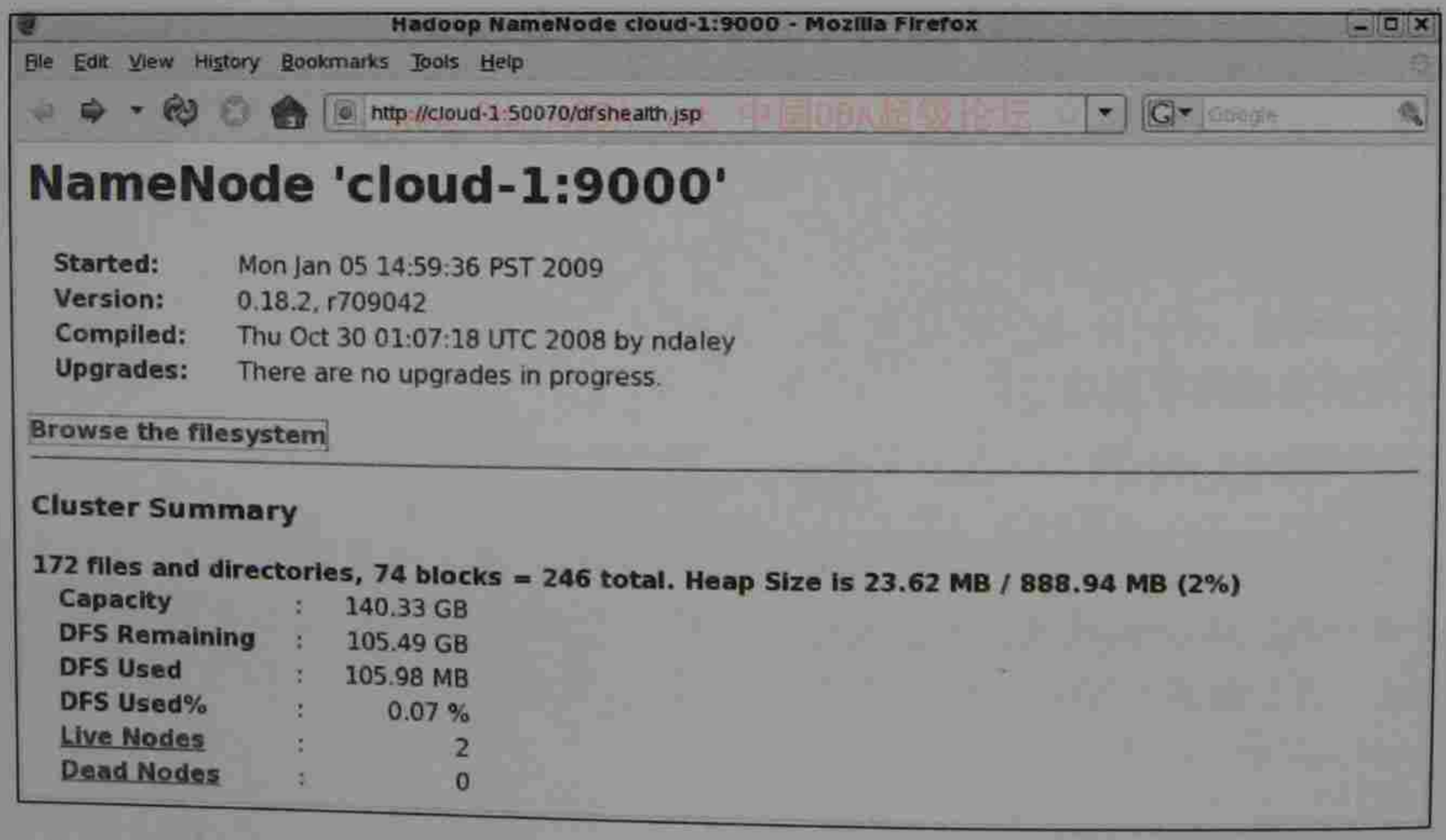

图2-4　HDFS的web界面快照。从这个界面可以通览HDFS文件系统，掌握每个独立节点上可获得的存储资源，并监控集群的整体健康状态

Hadoop提供一个MapReduce作业运行时状态的近似视图。图2-5显示了一个JobTracker通过端口50030给出的视图。

通过这个报告同样可以获得大量的信息，包括MapReduce中任务的运行时状态，以及整个作业的详细报告。后者尤为重要——这些日志描述了哪个节点执行了哪个任务，以及需要完成每个任务所需的时间或资源比。最后，还可以获得Hadoop对每个作业的配置，如图2-6所示。通过所

有这些信息，可以合理地管理MapReduce程序，更好地利用集群的资源。

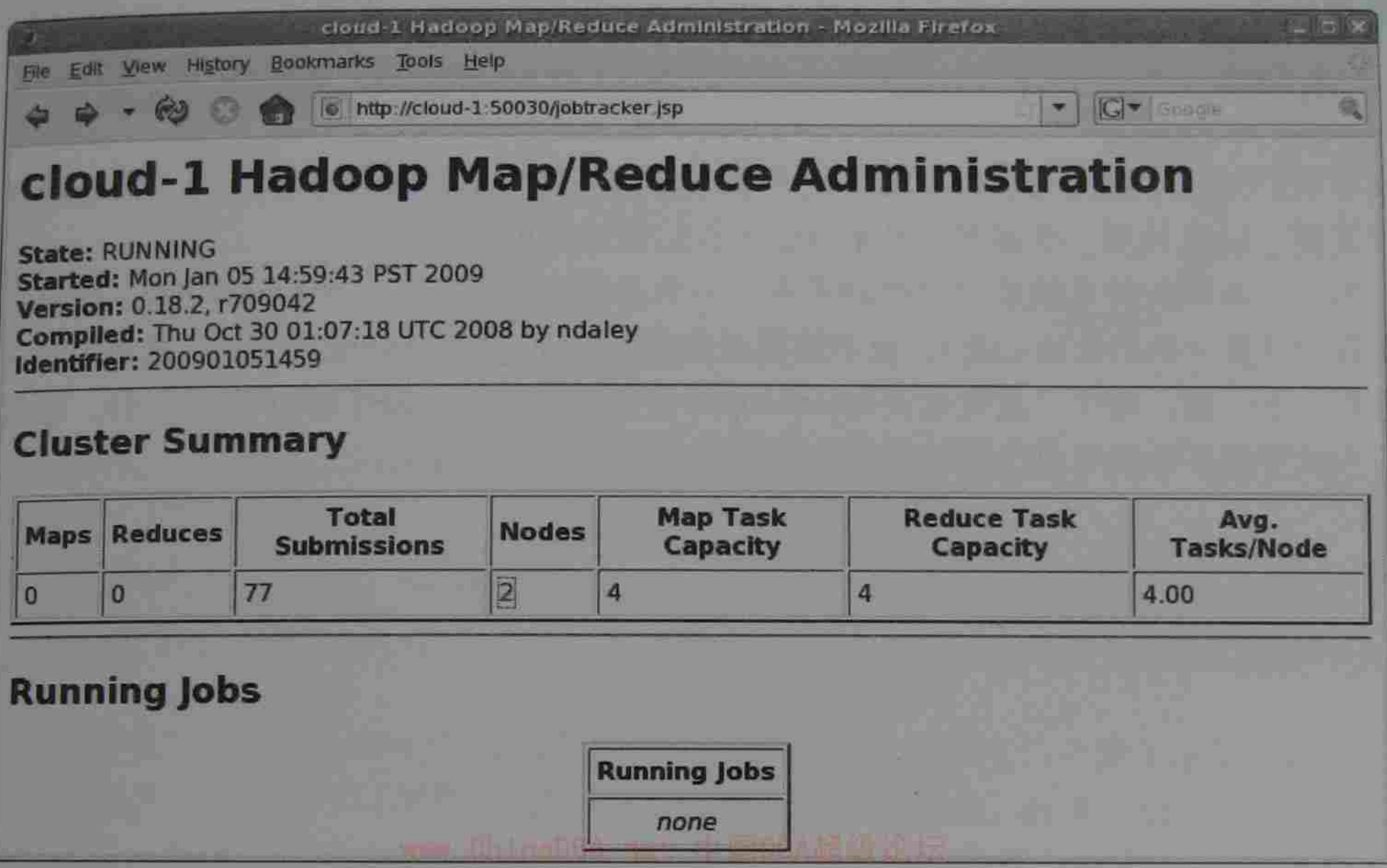

图2-5 MapReduce的web界面快照。这个工具可以监控活跃的MapReduce作业，并访问每个map和reduce任务的日志。还可以获得以前提交作业的日志，以辅助程序的调试

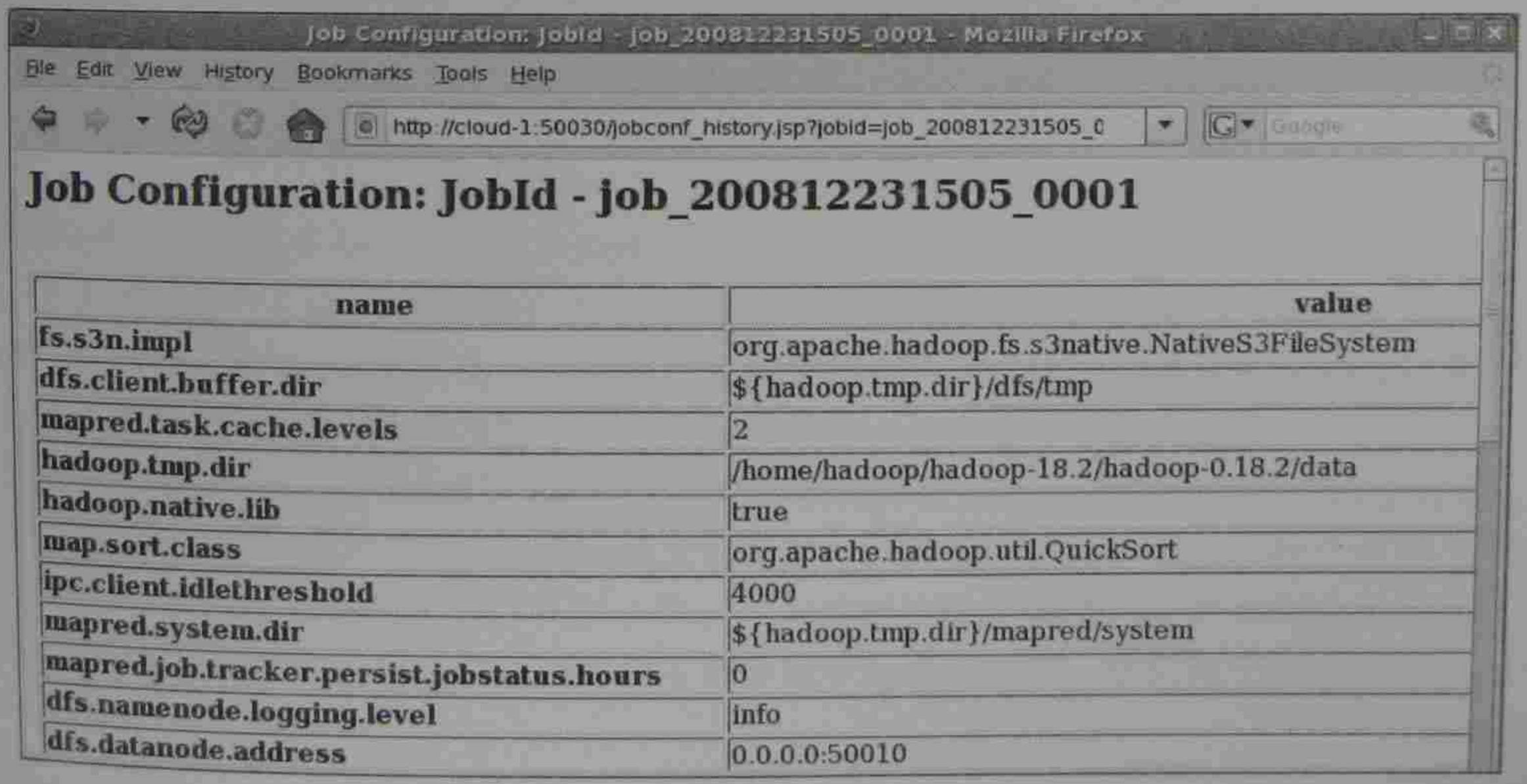

图2-6 一个特定MapReduce作业的详细配置，这些信息在通过调整参数来优化程序性能时可能有用

虽然在现阶段这些工具的用处也许不会马上显现，但当在集群上运行更为复杂的任务时，它们的易用性将逐渐展露头脚。随着深入学习Hadoop，我们会逐渐认识到这些工具的重要性。

2.5　小结

本章我们讨论了关键的节点以及它们在Hadoop体系结构中所扮演的角色。学习内容涵盖了如何配置集群，以及利用一些基本工具来监控集群的整体健康状态。

总体上，本章所讨论的是一次性的任务。一旦为集群配置好了一个NameNode，你将永远不需要再管它（至少我们希望如此），也不需要总去变更hadoop-site.xml配置文件，或者为节点分配守护进程。下一章，将学习需要每天和Hadoop打交道的部分，比如管理HDFS的文件。通过这些知识，你就可以开始撰写自己的MapReduce应用，并发掘Hadoop的巨大潜能。