

# 特别说明

此资料来自豆丁网(<http://www.docin.com/>)

您现在所看到的文档是使用下载器所生成的文档

此文档的原件位于

<http://www.docin.com/p-241830742.html>

感谢您的支持

抱米花

<http://blog.sina.com.cn/lotusbaob>



北京宽连十方数字技术有限公司

公开 内部公开 ✓  
机密 绝密

# Hadoop入门实战手册



北京宽连十方数字技术有限公司

技术研究部

(2011年7月)

# 目录

1 概述 .....	4
1.1 什么是Hadoop? .....	4
1.2 为什么要选择Hadoop? .....	4
1.2.1 系统特点 .....	4
1.2.2 使用场景 .....	5
2 术语 .....	5
3 Hadoop的单机部署 .....	6
3.1 目的 .....	6
3.2 先决条件 .....	6
3.2.1 支持平台 .....	6
3.2.2 所需软件 .....	6
3.2.3 安装软件 .....	6
3.3 下载 .....	7
3.4 运行Hadoop集群的准备工作 .....	7
3.5 单机模式的操作方法 .....	7
3.6 伪分布式模式的操作方法 .....	8
3.6.1 配置 .....	8
3.6.2 免密码ssh设置 .....	9
3.6.3 执行 .....	9
4 Hadoop集群搭建过程手记 .....	11
4.1 免密码SSH设置 .....	12
4.2 Hadoop软件安装 .....	12
4.3 Master(85)配置 .....	13
4.4 Slave(60,245上)配置 .....	14
4.5 初始化和启动hadoop集群 .....	15
4.5.1 初始化文件系统 .....	15
4.5.2 启动Hadoop .....	15
4.5.3 停止Hadoop .....	17
4.6 测试 .....	17
4.7 管理界面与命令 .....	19
4.7.1 hdfs运行状态界面 .....	19
4.7.2 Map-reduce的运行状态界面 .....	20
4.7.3 直接的命令行查看 .....	20
4.7.1 运行的进程查看 .....	21
5 架构分析 .....	23
5.1 HDFS .....	23
5.1.1 HDFS的三个重要角色 .....	24
5.1.2 HDFS设计特点 .....	25
5.2 MapReduce .....	26
5.2.1 算法介绍 .....	26

---

5.2.2 Hadoop框架下的mapreduce .....	28
5.3 综合架构分析 .....	29
6 Hadoop的应用 .....	38
7 系统维护 .....	39
7.1 Hadoop的系统监控 .....	39
7.2 Hadoop中的命令（Command）总结 .....	错误！未定义书签。 32
7.3 NameNode与JobTracker单点故障说明 .....	40
7.4 经验总结 .....	40
7.5 如何在一个hadoop集群新增或删除一些机器而不重启 .....	41
7.5.1 新增节点 .....	41
7.5.2 删除节点 .....	41
7.6 其它日常问题说明 .....	43
7.6.1 datanode启动失败，各slave节点的namespaceIDs与masters不同 .....	43
7.6.2 taskTracker和jobTracker 启动失败 .....	44
7.6.3 Shuffle Error: Exceeded MAX_FAILED_UNIQUE_FETCHES; bailing-out .....	45
7.6.4 Too many fetch-failures .....	45
7.6.5 能够启动datanode，但无法访问，也无法结束的错误 .....	45
7.6.6 java.io.IOException: Could not obtain block: .....	46
7.6.7 java.lang.OutOfMemoryError: Java heap space .....	46
7.6.8 解决hadoop OutOfMemoryError问题： .....	46
7.6.9 Hadoop java.io.IOException: .....	46
7.7 防火墙的端口开放要求 .....	47
7.7.1 与HDFS有关的地址及端口属性 .....	47
7.7.2 与MapReduce 有关的地址及端口属性 .....	48
8 附录 .....	49
8.1 hadoop历史 .....	49
8.2 Hadoop大记事 .....	50
8.3 Hadoop的几个主要子项目 .....	51
8.4 官方集群搭建参考 .....	51
8.4.1 配置文件 .....	51
8.4.2 集群配置说明 .....	52

# 1 概述

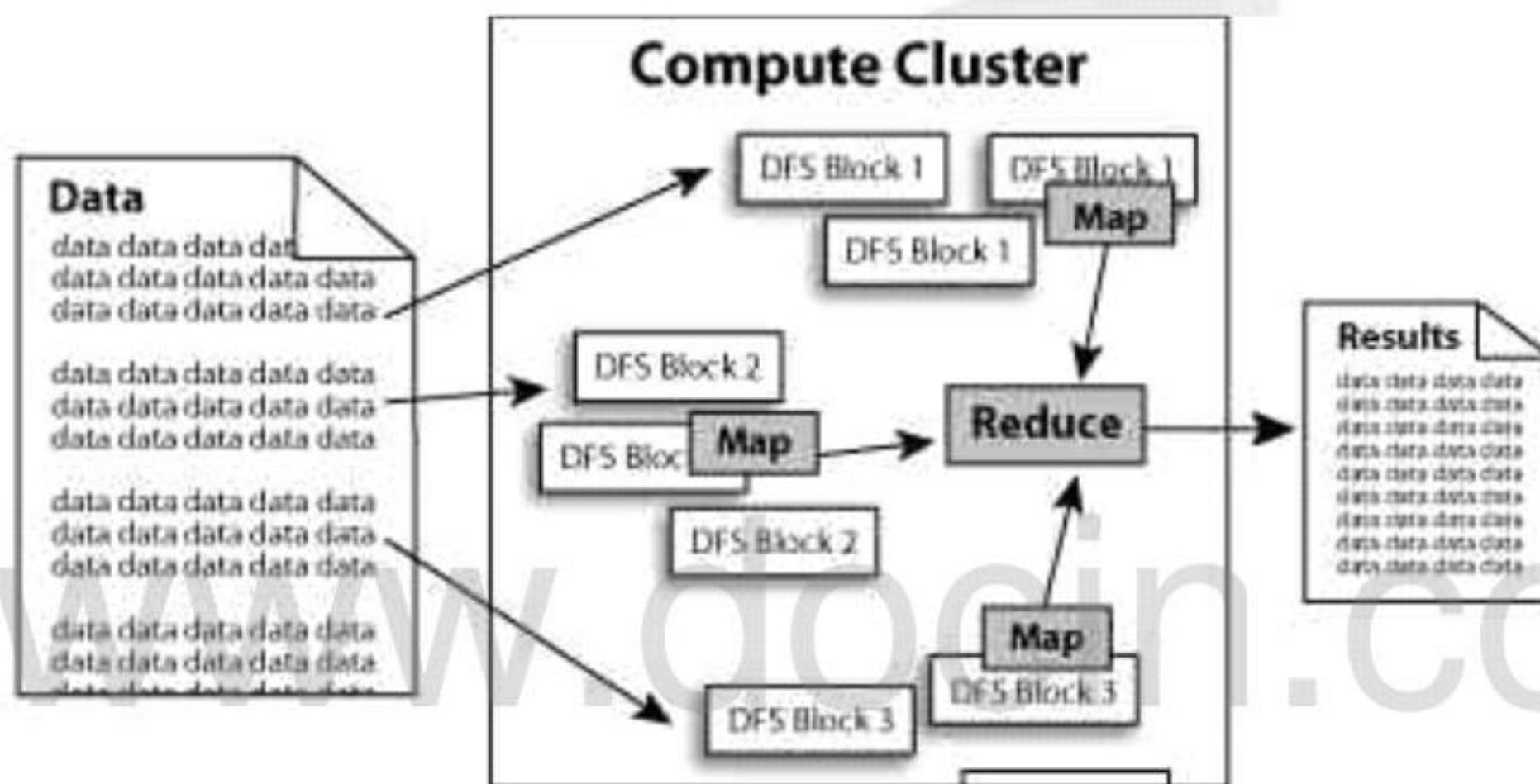
做什么事情之前，第一步是要知道**What**（是什么），然后是**Why**（为什么），最后才是**How**（怎么做）。避免将技术误用于不适合的场景，这一点非常重要。

## 1.1 什么是Hadoop？

Hadoop 由 Apache Software Foundation 公司于 2005 年秋天作为 Lucene 的子项目 Nutch 的一部分正式引入。它受到最先由 Google Lab 开发的 MapReduce 和 Google File System 的启发。2006 年 3 月份，MapReduce 和 Nutch Distributed File System (NDFS) 分别被纳入称为 Hadoop 的项目中。

Hadoop 并不仅仅是一个用于存储的分布式文件系统，而是设计用来**在由通用计算设备组成的大型集群上执行分布式应用的基础框架**。它由 Apache 基金会开发。用户可以在不了解分布式底层细节的情况下，开发分布式程序。充分利用集群的威力高速运算和存储。简单地说来，Hadoop 是一个可以更容易开发和运行处理大规模数据的软件平台。

下图是 Hadoop 的体系结构：



Hadoop 框架中最核心的设计就是：MapReduce 和 HDFS。

- 1) MapReduce 的思想是由 Google 的一篇论文所提及而被广为流传的，简单的一句话解释 MapReduce 就是“任务的分解与结果的汇总”。
- 2) HDFS 是 Hadoop 分布式文件系统 (Hadoop Distributed File System) 的缩写，为分布式计算存储提供了底层支持。

## 1.2 为什么要选择Hadoop？

### 1.2.1 系统特点

下面列举 hadoop 主要的一些特点：

- 1) 扩容能力 (Scalable)：能可靠地 (reliably) 存储和处理千兆字节 (PB) 数据。

- 2) 成本低 (Economical) : 可以通过普通机器组成的服务器群来分发以及处理数据。这些服务器群总计可达数千个节点。
- 3) 高效率 (Efficient) : 通过分发数据, hadoop 可以在数据所在的节点上并行地 (parallel) 处理它们, 这使得处理非常的快速。
- 4) 可靠性 (Reliable) : hadoop 能自动地维护数据的多份复制, 并且在任务失败后能自动地重新部署 (redeploy) 计算任务。

### 1.2.2 使用场景

个人觉得最适合的就是海量数据的分析, 其实 Google 最早提出 MapReduce 也就是为了海量数据分析。同时 HDFS 最早是为了搜索引擎实现而开发的, 后来才被用于分布式计算框架中。海量数据被分割于多个节点, 然后由每一个节点并行计算, 将得出的结果归并到输出。同时第一阶段的输出又可以作为下一阶段计算的输入, 因此可以想象到一个树状结构的分布式计算图, 在不同阶段都有不同产出, 同时并行和串行结合的计算也可以很好地在分布式集群的资源下得以高效的处理。

## 2 术语

- 1) Namenode: HDFS 采用 master/slave 架构。一个 HDFS 集群是由一个 Namenode 和一定数目的 Datanodes 组成。Namenode 是一个中心服务器, 负责管理文件系统的名字空间(namespace)以及客户端对文件的访问。Namenode 执行文件系统的名字空间操作, 比如打开、关闭、重命名文件或目录。它也负责确定数据块到具体 Datanode 节点的映射
- 2) Datanode: 集群中的 Datanode 一般是一个节点一个, 负责管理它所在节点上的存储。HDFS 暴露了文件系统的名字空间, 用户能够以文件的形式在上面存储数据。从内部看, 一个文件其实被分成一个或多个数据块, 这些块存储在一组 Datanode 上。Datanode 负责处理文件系统客户端的读写请求。在 Namenode 的统一调度下进行数据块的创建、删除和复制。
- 3) Secondnamenode: 光从字面上来理解, 很容易让一些初学者先入为主的认为: SecondaryNameNode (snn) 就是 NameNode (nn) 的热备进程。其实不是。snn 是 HDFS 架构中的一个组成部分, 但是经常由于名字而被人误解它真正的用途, 其实它真正的用途, 是用来保存 namenode 中对 HDFS metadata 的信息的备份, 并减少 namenode 重启的时间。
- 4) Jobtracker 和 Tasktracher: JobTracker 是 MapReduce 框架中最主要的类之一, 所有 job 的执行都由它来调度, 而且 Hadoop 系统中只配置一个 JobTracker 应用。它们都是由一个 master 服务 JobTracker 和多个运行于多个节点的 slaver 服务 TaskTracker 两个类提供的服务调度的。master 负责调度 job 的每一个子任务 task 运行于 slave 上, 并监控它们, 如果发现有失败的 task 就重新运行它, slave 则负责直接执行每一个 task。

TaskTracker 都需要运行在 HDFS 的 DataNode 上，而 JobTracker 则不需要，一般情况应该把 JobTracker 部署在单独的机器上。

### 3 Hadoop的单机部署

参考：

[http://hadoop.apache.org/common/docs/current/single\\_node\\_setup.html#Supported+Platforms](http://hadoop.apache.org/common/docs/current/single_node_setup.html#Supported+Platforms)

#### 3.1 目的

本章节的目的是帮助你快速完成单机上的 Hadoop 安装与使用以便你对 [Hadoop 分布式文件系统\(HDFS\)](#) 和 Map-Reduce 框架有所体会，比如在 HDFS 上运行示例程序或简单作业等。

#### 3.2 先决条件

##### 3.2.1 支持平台

- 1) GNU/Linux 是产品开发和运行的平台。 Hadoop 已在有 2000 个节点的 GNU/Linux 主机组的集群系统上得到验证。
- 2) Win32 平台是作为开发平台支持的。由于分布式操作尚未在 Win32 平台上充分测试，所以还不作为一个生产平台被支持。

##### 3.2.2 所需软件

Linux 和 Windows 所需软件包括：

1. Sun Java™1.6.x，必须安装。
2. **ssh** 必须安装并且保证 **sshd** 一直运行，以便用 Hadoop 脚本管理远端 Hadoop 守护进程。

##### 3.2.3 安装软件

如果你的集群尚未安装所需软件，你得首先安装它们。

以 Linux 为例：

```
$ sudo apt-get install ssh  
$ sudo apt-get install rsync
```

### 3.3 下载

为了获取 Hadoop 的发行版，从 Apache 的某个镜像服务器上下载最近的 [稳定发行版](#)。

下载地址：<http://mirror.bjtu.edu.cn/apache/hadoop/common/stable/>

### 3.4 运行Hadoop集群的准备工作

解压所下载的 Hadoop 发行版。编辑 `conf/hadoop-env.sh` 文件，至少需要将 `JAVA_HOME` 设置为 Java 安装根路径。

尝试如下命令：

```
$ bin/hadoop
```

将会显示 **hadoop** 脚本的使用文档。

现在你可以用以下三种支持的模式中的一种启动 Hadoop 集群：

- 单机模式
- 伪分布式模式
- 完全分布式模式

### 3.5 单机模式的操作方法

默认情况下，Hadoop 被配置成以非分布式模式运行的一个独立 Java 进程。这对调试非常有帮助。

下面的实例将已解压的 `conf` 目录拷贝作为输入，查找并显示匹配给定正则表达式的条目。输出写入到指定的 `output` 目录。

```
$ mkdir input
$ cp conf/*.xml input
$ bin/hadoop jar hadoop-examples-0.20.203.0.jar grep input output 'dfs[a-z.]+'
$ cat output/*
```

注：语法不理解没关系看下面进一步说明

显示结果

```
1      dfsadmin
```

## 3.6 伪分布式模式的操作方法

Hadoop 可以在单节点上以所谓的伪分布式模式运行，此时每一个 Hadoop 守护进程都作为一个独立的 Java 进程运行。

### 3.6.1 配置

注：以前的版本是 `hadoop-site.xml`, 可 hadoop 在 0.20 版本，配置文件由以前的 `hadoop-site.xml` 文件变成三个配置文件 `core-site.xml`, `hdfs-site.xml`, `mapred-site.xml`. 内在的原因是因为 hadoop 代码量越来越大，拆解成三个大的分支进行独立开发，配置文件也独立了

`conf/core-site.xml`:

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

`conf/hdfs-site.xml`:

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

`conf/mapred-site.xml`:

```
<configuration>
  <property>
    <name>mapred.job.tracker</name>
    <value>localhost:9001</value>
  </property>
</configuration>
```

### 3.6.2 免密码ssh设置

现在确认能否不输入口令就用 ssh 登录 localhost:

```
$ ssh localhost
```

如果不输入口令就无法用 ssh 登陆 localhost，执行下面的命令：

```
$ ssh-keygen -t dsa -P '' -f ~/.ssh/id_dsa
$ cat ~/.ssh/id_dsa.pub >> ~/.ssh/authorized_keys
```

### 3.6.3 执行

首先使用 hadoop 命令对 Hadoop File System (HDFS) 进行格式化。

首先，请求 namenode 对 DFS 文件系统进行格式化。在安装过程中完成了这个步骤，但是了解是否需要生成干净的文件系统是有用的。

```
[hadoop@TEST085 hadoop-0.20.203.0]$ bin/hadoop namenode -format
```

注：在确认请求之后，文件系统进行格式化并返回一些信息：

```
11/07/12 17:47:12 INFO namenode.NameNode: STARTUP_MSG:
```

```
*****
```

```
STARTUP_MSG: Starting NameNode
```

```
STARTUP_MSG: host = TEST085/202.102.110.206
```

```
STARTUP_MSG: args = [-format]
```

```
STARTUP_MSG: version = 0.20.203.0
```

```
STARTUP_MSG: build =
```

```
http://svn.apache.org/repos/asf/hadoop/common/branches/branch-0.20-security-203 -r
1099333; compiled by 'oom' on Wed May 4 07:57:50 PDT 2011
```

```
*****/
```

```
11/07/12 17:47:12 INFO util.GSet: VM type      = 32-bit
```

```
11/07/12 17:47:12 INFO util.GSet: 2% max memory = 19.33375 MB
```

```
11/07/12 17:47:12 INFO util.GSet: capacity      = 2^22 = 4194304 entries
```

```
11/07/12 17:47:12 INFO util.GSet: recommended=4194304, actual=4194304
```

```
11/07/12 17:47:13 INFO namenode.FSNamesystem: fsOwner=hadoop
11/07/12 17:47:13 INFO namenode.FSNamesystem: supergroup=supergroup
11/07/12 17:47:13 INFO namenode.FSNamesystem: isPermissionEnabled=true
11/07/12 17:47:13 INFO namenode.FSNamesystem: dfs.block.invalidate.limit=100
11/07/12 17:47:13 INFO namenode.FSNamesystem: isAccessTokenEnabled=false
accessKeyUpdateInterval=0 min(s), accessTokenLifetime=0 min(s)
11/07/12 17:47:13 INFO namenode.NameNode: Caching file names occurring more than
10 times
11/07/12 17:47:13 INFO common.Storage: Image file of size 112 saved in 0 seconds.
11/07/12 17:47:13 INFO common.Storage: Storage directory
/tmp/hadoop-hadoop/dfs/name has been successfully formatted.
11/07/12 17:47:13 INFO namenode.NameNode: SHUTDOWN_MSG:
*****
SHUTDOWN_MSG: Shutting down NameNode at TEST085/202.102.110.206
*****
```

接下来，启动 Hadoop 守护进程。

启动 Hadoop 守护进程：

```
[hadoop@TEST085 hadoop-0.20.203.0]$ bin/start-all.sh
```

注：1) Hadoop 守护进程的日志写入到 \${HADOOP\_LOG\_DIR} 目录（默认是 \${HADOOP\_HOME}/logs）

2) 启动 hadoop，但 ssh 端口不是默认的 22 怎么样？好在它可以配置。在 conf/hadoop-env.sh 里改下。如：

```
export HADOOP_SSH_OPTS="-p 1234"
```

浏览 NameNode 和 JobTracker 的网络接口，它们的地址默认为：

- NameNode - <http://localhost:50070/>
- JobTracker - <http://localhost:50030/>

将输入文件拷贝到分布式文件系统：

```
$ bin/hadoop fs -put conf input
```

运行发行版提供的示例程序：

```
$ bin/hadoop jar hadoop-examples-0.20.203.0.jar grep input output 'dfs[a-z.]+'
```

查看输出文件：

将输出文件从分布式文件系统拷贝到本地文件系统查看：

```
$ bin/hadoop fs -get output output  
$ cat output/*
```

或者

在分布式文件系统上查看输出文件：

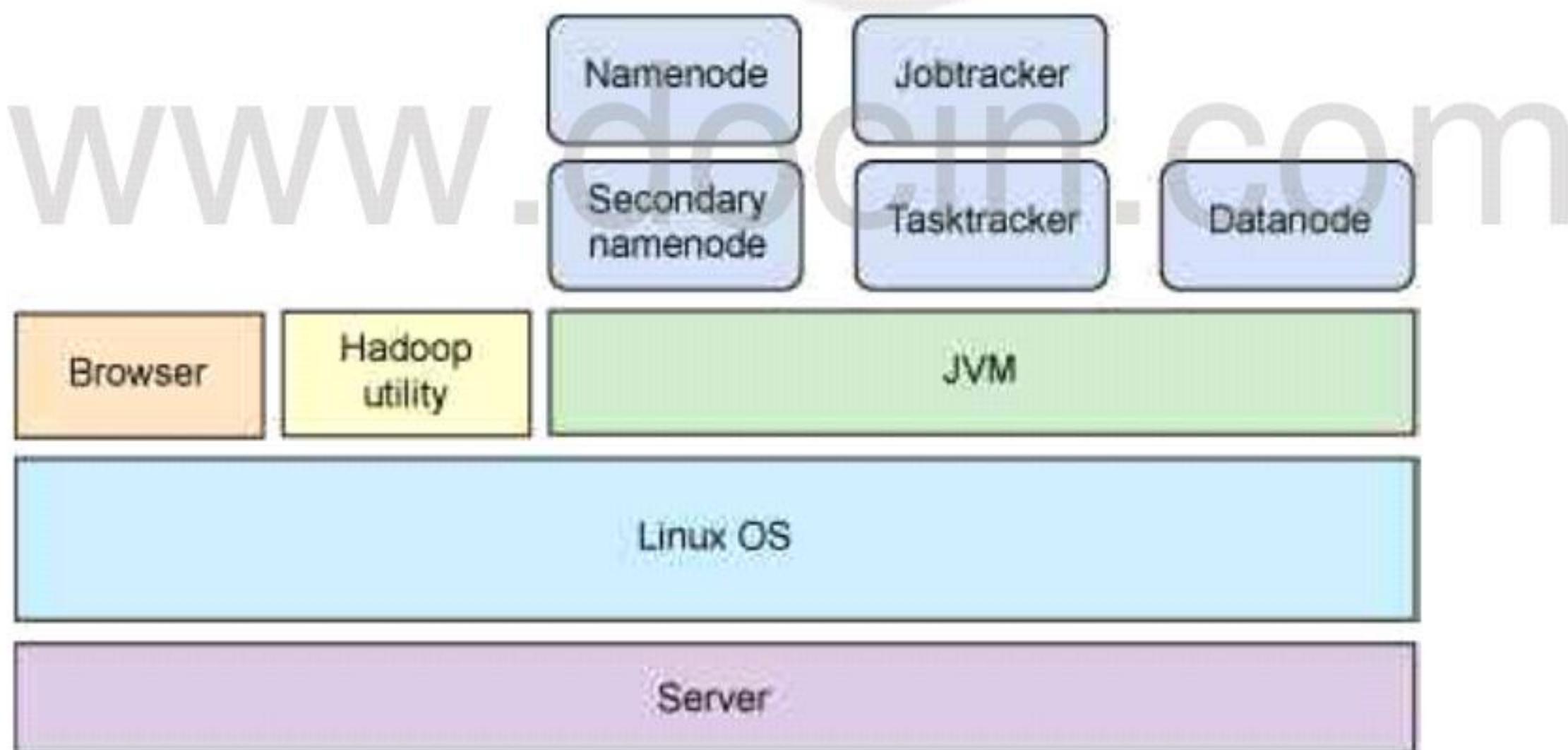
```
$ bin/hadoop fs -cat output/*
```

完成全部操作后，停止守护进程：

```
$ bin/stop-all.sh
```

Hadoop 在这个伪分布式配置中启动 5 个守护进程：namenode、secondarynamenode、datanode、jobtracker 和 tasktracker。在启动每个守护进程时，会看到一些相关信息（指出存储日志的位置）。每个守护进程都在后台运行。图 1 说明完成启动之后伪分布式配置的架构。

图 1. 伪分布式 Hadoop 配置



## 4 Hadoop集群搭建过程手记

参考：[http://hadoop.apache.org/common/docs/r0.19.2/cn/cluster\\_setup.html](http://hadoop.apache.org/common/docs/r0.19.2/cn/cluster_setup.html)

先用了三台服务器作了 hadoop 集群的部署测试，服务器有 192.168.10.85（下面简称 85），192.168.10.160（下面简称 160），192.168.10.254（下面简称 254），架构规划如下：

- 1) 85 作为 NameNode, SecondaryNameNode, JobTracker;
- 2) 160 和 254 作为 DataNode, TaskTracker

## 4.1 免密码SSH设置

打通 ssh，让 85 免登陆到 160, 254。打通过程如下：

- 1) 名称节点 85 和数据节点（160, 254）各自创建用户 hadoop，使用相同的密码。
- 2) 以 hadoop 用户名登陆名称节点（85）执行 `ssh-keygen -t rsa` 然后一路回车，完毕后生成文件 `.ssh/id_rsa.pub`，把这个文件复制到当前位置，命名为 `authorized_keys`；然后执行命令 `ssh 127.0.0.1`，如果不需要密码则直接登陆进去的话，就达到要求；否则需检查 `authorized_keys` 的权限，看是否为 644（`-rw-r--r--`）。
- 3) 接下来，同样也 hadoop 用户登陆数据节点服务器（160, 254），创建 `.ssh` 目录，并给与 600 权限（`chmod 600 .ssh`）；再把名称服务器上的 `authorized_keys` 复制到目录数据节点（160, 254）`./ssh`，注意权限和目录结构跟名称节点保持一致，然后再从名称节点用 ssh 登陆数据节点，如果不需要密码能登陆成功，则 ssh 的配置结束。

## 4.2 Hadoop软件安装

以 **hadoop** 用户登陆，将安装软件解压到集群内的所有机器上，编辑 `conf/hadoop-env.sh` 文件，至少需要将 `JAVA_HOME` 设置为 Java 安装根路径（安装过程参考“[3 hadoop 的单机部署](#)”）。

我们用 `HADOOP_HOME` 指定安装的根路径。通常集群里的所有机器的 `HADOOP_HOME` 路径相同，安装路径定为：`/home/hadoop/hadoop-0.20.203.0`

- 1) 进行 JDK 和内存占用配置：

`conf/hadoop-env.sh` 需要修改的内容：

```
# The java implementation to use. Required.  
export JAVA_HOME=/usr/local/java --改成你自己 jdk 安装的目录  
# The maximum amount of heap to use, in MB. Default is 1000.  
export HADOOP_HEAPSIZE=200 --根据你的内存大小调整
```

- 2) 修改 `masters` 和 `slaves` 配置

修改文件 /usr/local/hadoop/conf/slaves 及 /usr/local/hadoop/conf/masters, 把数据节点的主机名加到 slaves、名称节点主机名加到 masters。可以加多个，每行一个。注意主机名需要在每个服务器的 /etc/hosts 映射好。

```
[hadoop@TEST085 hadoop-0.20.203.0]$ vi conf/slaves
```

```
192.168.10.160  
192.168.10.245
```

```
[hadoop@TEST085 hadoop-0.20.203.0]$ vi conf/master
```

```
192.168.10.85
```

### 4.3 Master(85)配置

85 为 master 结点，则 85 的配置文件如下：

hadoop 在 0.20 版本，配置文件由以前的 **hadoop-site.xml** 文件变成三个配置文件 **core-site.xml**, **hdfs-site.xml**, **mapred-site.xml**。内在的原因是因为 **hadoop** 代码量越来越庞大，拆解成三个大的分支进行独立开发，配置文件也独立了。

下面是三个配置文件示例：

```
[root@192.168.10.85 conf]# cat 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://192.168.10.85:9000</value>  
</property>  
</configuration>
```

```
[root@192.168.10.85 conf]# cat 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>Default block replication.
The actual number of replications can be specified when the file is created.
The default is used if replication is not specified in create time.
</description>
</property>
</configuration>
```

[root@192.168.10.85 conf]# cat **mapred-site.xml**

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

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

<configuration>
<property>
<name>mapred.job.tracker</name>
<value>192.168.168.85:9001</value>
</property>
</configuration>
```

设置系统环境，以方便执行**hadoop**命令

在/home/hadoop/.bashrc加入

```
export HADOOP_HOME=/root/hadoop/hadoop
export HADOOP_CONF_DIR=$HADOOP_HOME/conf
export PATH=/root/hadoop/hadoop/bin:$PATH
```

#### 4.4 Slave(60,245上)配置

在Slave(60,245上)上的配置文件如下(**hdfs-site.xml**不需要配置):

[root@192.168.10.160 conf]# cat **core-site.xml**

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

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

<configuration>
<property>
```

```
<name>fs.default.name</name>
<value>hdfs://192.168.10.85:9000</value>
</property>
</configuration>
```

[root@192.168.10.160 conf]# **cat mapred-site.xml**

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

<!-- Put site-specific property overrides in this file. --&gt;

&lt;configuration&gt;
&lt;property&gt;
&lt;name&gt;mapred.job.tracker&lt;/name&gt;
&lt;value&gt;192.168.10.85:9001&lt;/value&gt;
&lt;/property&gt;
&lt;/configuration&gt;</pre>
```

## 4.5 初始化和启动hadoop集群

### 4.5.1 初始化文件系统

初始化 namenode,为 HDFS 作第一次运行的准备。

```
$ bin/hadoop namenode -format
```

注：一看到 **format** 就和磁盘格式化联想到一起，然后这个 **format** 是对 **hdfs** 来说的，所以有些人害怕真的是格式化自己的文件系统了，其实大可不必担心，**namenode format** 只是初始化一些目录和文件而已。

### 4.5.2 启动Hadoop

在 **master** 结点配置用户环境变量,在 **master** 结点 **192.168.10.85** 启动 **hadoop** 集群程序,

执行 **bin** 目录下的 **start-all.sh**

```
[hadoop@TEST085 hadoop-0.20.203.0]$ bin/start-all.sh
starting namenode, logging to
/home/hadoop/hadoop-0.20.203.0/bin/../logs/hadoop-hadoop-namenode-TEST085.out
192.168.10.160: starting datanode, logging to
/home/hadoop/hadoop-0.20.203.0/bin/../logs/hadoop-hadoop-datanode-DBSERVER.out
192.168.10.245: starting datanode, logging to
/home/hadoop/hadoop-0.20.203.0/bin/../logs/hadoop-hadoop-datanode-localhost.localdomain.out
```

```
192.168.10.85: starting secondarynamenode, logging to
/home/hadoop/hadoop-0.20.203.0/bin/../logs/hadoop-hadoop-secondarynamenode-TEST085.out
starting jobtracker, logging to
/home/hadoop/hadoop-0.20.203.0/bin/../logs/hadoop-hadoop-jobtracker-TEST085.out
192.168.10.160: starting tasktracker, logging to
/home/hadoop/hadoop-0.20.203.0/bin/../logs/hadoop-hadoop-tasktracker-DBSERVER.out
192.168.10.245: starting tasktracker, logging to
/home/hadoop/hadoop-0.20.203.0/bin/../logs/hadoop-hadoop-tasktracker-localhost.localdomain.out
```

另也可以分步执行：

第一步启动 **hdfs**:

```
[hadoop@TEST085 hadoop-0.20.203.0]$ bin/start-dfs.sh
starting namenode, logging to
/home/hadoop/hadoop-0.20.203.0/bin/../logs/hadoop-hadoop-namenode-TEST085.out
192.168.10.160: starting datanode, logging to
/home/hadoop/hadoop-0.20.203.0/bin/../logs/hadoop-hadoop-datanode-DBSERVER.out
192.168.10.245: starting datanode, logging to
/home/hadoop/hadoop-0.20.203.0/bin/../logs/hadoop-hadoop-datanode-localhost.localdomain.out
192.168.10.85: starting secondarynamenode, logging to
/home/hadoop/hadoop-0.20.203.0/bin/../logs/hadoop-hadoop-secondarynamenode-TEST085.out
```

第二步启动 **map-reduce**:

```
[hadoop@TEST085 hadoop-0.20.203.0]$ bin/start-mapred.sh
starting jobtracker, logging to
/home/hadoop/hadoop-0.20.203.0/bin/../logs/hadoop-hadoop-jobtracker-TEST085.out
192.168.10.160: starting tasktracker, logging to
/home/hadoop/hadoop-0.20.203.0/bin/../logs/hadoop-hadoop-tasktracker-DBSERVER.out
192.168.10.245: starting tasktracker, logging to
/home/hadoop/hadoop-0.20.203.0/bin/../logs/hadoop-hadoop-tasktracker-localhost.localdomain.out
```

注：实际应用中**NameNode**和**Jobtracker**不在同一台服务器上，则需要按下面方式进行启动

在分配的**NameNode**上，运行下面的命令启动**HDFS**：

**\$ bin/start-dfs.sh**

**bin/start-dfs.sh**脚本会参照**NameNode**上

**\${HADOOP\_CONF\_DIR}/slaves**文件的内容，在所有列出的**slave**上

启动**DataNode**守护进程。

在分配的**JobTracker**上，运行下面的命令启动**Map/Reduce**:

\$ bin/start-mapred.sh

*bin/start-mapred.sh*脚本会参照**JobTracker**上

*\${HADOOP\_CONF\_DIR}/slaves*文件的内容，在所有列出的**slave**上  
启动**TaskTracker**守护进程。

### 4.5.3 停止Hadoop

在分配的**NameNode**上，执行下面的命令停止 HDFS:

\$ bin/stop-dfs.sh

*bin/stop-dfs.sh*脚本会参照 NameNode 上 *\${HADOOP\_CONF\_DIR}/slaves*文件  
的内容，在所有列出的 **slave** 上停止 DataNode 守护进程。

在分配的**JobTracker**上，运行下面的命令停止 Map/Reduce:

\$ bin/stop-mapred.sh

*bin/stop-mapred.sh*脚本会参照 JobTracker 上 *\${HADOOP\_CONF\_DIR}/slaves*文件  
的内容，在所有列出的 **slave** 上停止 TaskTracker 守护进程。

## 4.6 测试

在 **hdfs** 上创建 **test1** 文件夹，上传文件到此目录下

```
[hadoop@TEST085 hadoop-0.20.203.0]$ bin/hadoop fs -mkdir test1
[hadoop@TEST085 hadoop-0.20.203.0]$ bin/hadoop fs -put ./README.txt test1
[hadoop@TEST085 hadoop-0.20.203.0]$ bin/hadoop fs -ls
Found 1 items
drwxr-xr-x - hadoop supergroup          0 2011-07-21 19:58 /user/hadoop/test1
```

运行一个 **map-reduce** 示例程序 **wordcount**，运行结果如下：

```
[hadoop@TEST085 hadoop-0.20.203.0]$ hadoop jar hadoop-examples-0.20.203.0.jar wordcount
/user/hadoop/test1/README.txt output1
```

结果如下：

```
11/07/22 15:21:29 INFO input.FileInputFormat: Total input paths to process : 1
11/07/22 15:21:30 INFO mapred.JobClient: Running job: job_201107221440_0001
11/07/22 15:21:31 INFO mapred.JobClient: map 0% reduce 0%
11/07/22 15:21:51 INFO mapred.JobClient: map 100% reduce 0%
```



```
11/07/22 15:22:09 INFO mapred.JobClient: map 100% reduce 100%
11/07/22 15:22:15 INFO mapred.JobClient: Job complete: job_201107221440_0001
11/07/22 15:22:15 INFO mapred.JobClient: Counters: 25
11/07/22 15:22:15 INFO mapred.JobClient: Job Counters
11/07/22 15:22:15 INFO mapred.JobClient: Launched reduce tasks=1
11/07/22 15:22:15 INFO mapred.JobClient: SLOTS_MILLIS_MAPS=18252
11/07/22 15:22:15 INFO mapred.JobClient: Total time spent by all reduces waiting after reserving
slots (ms)=0
11/07/22 15:22:15 INFO mapred.JobClient: Total time spent by all maps waiting after reserving slots
(ms)=0
11/07/22 15:22:15 INFO mapred.JobClient: Launched map tasks=1
11/07/22 15:22:15 INFO mapred.JobClient: Data-local map tasks=1
11/07/22 15:22:15 INFO mapred.JobClient: SLOTS_MILLIS_REDUCES=15479
11/07/22 15:22:15 INFO mapred.JobClient: File Output Format Counters
11/07/22 15:22:15 INFO mapred.JobClient: Bytes Written=1306
11/07/22 15:22:15 INFO mapred.JobClient: FileSystemCounters
11/07/22 15:22:15 INFO mapred.JobClient: FILE_BYTES_READ=1836
11/07/22 15:22:15 INFO mapred.JobClient: HDFS_BYTES_READ=1485
11/07/22 15:22:15 INFO mapred.JobClient: FILE_BYTES_WRITTEN=45989
11/07/22 15:22:15 INFO mapred.JobClient: HDFS_BYTES_WRITTEN=1306
11/07/22 15:22:15 INFO mapred.JobClient: File Input Format Counters
11/07/22 15:22:15 INFO mapred.JobClient: Bytes Read=1366
11/07/22 15:22:15 INFO mapred.JobClient: Map-Reduce Framework
11/07/22 15:22:15 INFO mapred.JobClient: Reduce input groups=131
11/07/22 15:22:15 INFO mapred.JobClient: Map output materialized bytes=1836
11/07/22 15:22:15 INFO mapred.JobClient: Combine output records=131
11/07/22 15:22:15 INFO mapred.JobClient: Map input records=31
11/07/22 15:22:15 INFO mapred.JobClient: Reduce shuffle bytes=1836
11/07/22 15:22:15 INFO mapred.JobClient: Reduce output records=131
11/07/22 15:22:15 INFO mapred.JobClient: Spilled Records=262
11/07/22 15:22:15 INFO mapred.JobClient: Map output bytes=2055
11/07/22 15:22:15 INFO mapred.JobClient: Combine input records=179
11/07/22 15:22:15 INFO mapred.JobClient: Map output records=179
11/07/22 15:22:15 INFO mapred.JobClient: SPLIT_RAW_BYTES=119
11/07/22 15:22:15 INFO mapred.JobClient: Reduce input records=131
```

[hadoop@TEST085 hadoop-0.20.203.0]\$ bin/hadoop fs -ls output1

查看输出结果文件，这个文件在 **hdfs** 上：

```
[hadoop@TEST085 hadoop-0.20.203.0]$ hadoop fs -ls output1
Found 3 items
-rw-r--r-- 3 hadoop supergroup          0 2011-07-22 15:22 /user/hadoop/output1/_SUCCESS
drwxr-xr-x - hadoop supergroup        0 2011-07-22 15:21 /user/hadoop/output1/_logs
-rw-r--r-- 3 hadoop supergroup    1306 2011-07-22 15:22 /user/hadoop/output1/part-r-00000
```

```
[hadoop@TEST085 hadoop-0.20.203.0]$ hadoop fs -cat output1/part-r-00000
```

(BIS),	1
(ECCN)	1
(TSU)	1
(see	1
5D002. C. 1,	1
740. 13) 1	
<http://www.wassenaar.org/>	1
Administration	1
Apache	1
BEFORE	1
BIS	1
Bureau	1
Commerce,	1
..... 省略	

## 4.7 管理界面与命令

### 4.7.1 hdfs运行状态界面

查看 **hdfs** 运行状态,可以通过 **web** 界面来访问

<http://192.168.10.85:50070/dfshealth.jsp>

## NameNode '192.168.10.85:9000'

```
Started: Thu Jul 21 19:17:58 CST 2011
Version: 0.20.203.0, r1099333
Compiled: Wed May 4 07:57:50 PDT 2011 by com
Upgrades: There are no upgrades in progress.
```

[Browse the filesystem](#)  
[NameNode Logs](#)

### Cluster Summary

```
17 files and directories, 4 blocks = 21 total. Heap Size is 31.13 MB / 966.69 MB (3%)
Configured Capacity : 34.62 GB
DFS Used           : 604 KB
Non DFS Used       : 28.03 GB
DFS Remaining      : 6.59 GB
DFS Used%          : 0 %
DFS Remaining%     : 19.03 %
Live Nodes          : 1
Dead Nodes          : 0
Decommissioning Nodes: 0
Number of Under-Replicated Blocks: 4
```

### NameNode Storage:

Storage Directory	Type	State
/tmp/hadoop-hadoop/dfs/name	IMAGE_AND_EDITS	Active

This is Apache Hadoop release 0.20.203.0

### 4.7.2 Map-reduce的运行状态界面

查看 map-reduce 信息，可以通过 web 界面来访问：

<http://192.168.10.85:50030/jobtracker.jsp>

### 4.7.3 直接的命令行查看

下面是直接命令行看到的结果。

```
[root@192.168.10.85 ~]# hadoop dfsadmin -report
Configured Capacity: 291104653312 (271.11 GB)
Present Capacity: 74432905216 (69.32 GB)
DFS Remaining: 74432823296 (69.32 GB)
DFS Used: 81920 (80 KB)
DFS Used%: 0%
Under replicated blocks: 1
Blocks with corrupt replicas: 0
Missing blocks: 0
```

Datanodes available: 2 (2 total, 0 dead)

Name: 192.168.10.160:50010

Decommission Status : Normal

Configured Capacity: 37169479680 (34.62 GB)

DFS Used: 36864 (36 KB)

Non DFS Used: 30097534976 (28.03 GB)

DFS Remaining: 7071907840 (6.59 GB)

DFS Used%: 0%

DFS Remaining%: 19.03%

Last contact: Fri Jul 22 15:16:36 CST 2011

Name: 192.168.10.245:50010

Decommission Status : Normal

Configured Capacity: 253935173632 (236.5 GB)

DFS Used: 45056 (44 KB)

Non DFS Used: 186574213120 (173.76 GB)

DFS Remaining: 67360915456 (62.73 GB)

DFS Used%: 0%

DFS Remaining%: 26.53%

Last contact: Fri Jul 22 15:16:37 CST 2011

#### 4.7.1 运行的进程查看

在 master 机器上通过 ps 命令查看, 可以看到 namenode/secondarynamenode/jobtracker 在运行, 如下:

```
[hadoop@TEST085 hadoop-0.20.203.0]$ ps uax |grep ha
```

```
hadoop 27440 0.9 5.8 1195576 59428 pts/0 S 14:40 0:05
/usr/local/java/bin/java -Dproc_namenode -Xmx1000m
-Dcom.sun.management.jmxremote -Dcom.sun.management.jmxremote
-Dhadoop.log.dir=/home/hadoop/hadoop-0.20.203.0/bin/../logs
-Dhadoop.log.file=hadoop-hadoop-namenode-TEST085.log
-Dhadoop.home.dir=/home/hadoop/
```

```
hadoop 17840 0.5 3.5 1185092 36196 ? S 14:40 0:03
/usr/local/java/bin/java -Dproc_secondarynamenode -Xmx1000m
-Dcom.sun.management.jmxremote -Dcom.sun.management.jmxremote
-Dhadoop.log.dir=/home/hadoop/hadoop-0.20.203.0/bin/../logs
-Dhadoop.log.file=hadoop-hadoop-secondarynamenode-TEST085.log -Dhadoop.home
```

```
hadoop 18435 0.9 4.2 1199836 42744 pts/0 S 14:40 0:05
/usr/local/java/bin/java -Dproc_jobtracker -Xmx1000m
-Dcom.sun.management.jmxremote -Dcom.sun.management.jmxremote
-Dhadoop.log.dir=/home/hadoop/hadoop-0.20.203.0/bin/../logs
-Dhadoop.log.file=hadoop-hadoop-jobtracker-TEST085.log
-Dhadoop.home.dir=/home/had

[hadoop@TEST085 hadoop-0.20.203.0]$ netstat -ant | grep 900
tcp      0      0 192.168.10.85:9000          0.0.0.0:*
LISTEN
tcp      0      0 192.168.10.85:9001          0.0.0.0:*
LISTEN
tcp      0      0 192.168.10.85:9000          192.168.10.245:64074
ESTABLISHED
tcp      0      0 192.168.10.85:9001          192.168.10.245:5956
ESTABLISHED
tcp      0      0 192.168.10.85:9000          192.168.10.85:49534
ESTABLISHED
tcp      0      0 192.168.10.85:49534         192.168.10.85:9000
ESTABLISHED
tcp      0      0 192.168.10.85:9000          192.168.10.160:34921
ESTABLISHED
tcp      0      0 192.168.10.85:9001          192.168.10.160:34926
ESTABLISHED
```

在 slaves 机器上通过 ps 命令可以看到 datanode 和 tasktracher 的进程在运行。

```
[hadoop@DBSERVER /]$ ps aux |grep ha
hadoop 11127 1.3 6.4 1179584 32656 ? S 14:41 0:04
/usr/local/java/bin/java -Dproc_datanode -Xmx1000m -server
-Dcom.sun.management.jmxremote -Dcom.sun.management.jmxremote
-Dhadoop.log.dir=/home/hadoop/hadoop-0.20.203.0/bin/../logs
-Dhadoop.log.file=hadoop-hadoop-datanode-DBSERVER.log -Dhadoop.home.dir=/hom

hadoop 11215 0.8 7.0 1188132 35660 ? S 14:42 0:02
/usr/local/java/bin/java -Dproc_tasktracker -Xmx1000m
-Dhadoop.log.dir=/home/hadoop/hadoop-0.20.203.0/bin/../logs
-Dhadoop.log.file=hadoop-hadoop-tasktracker-DBSERVER.log
```

-Dhadoop.home.dir=/home/hadoop/hadoop-0.20.203.0/bin/.. -Dhadoop.id.str=hadoop  
-Dhadoop

```
[hadoop@DBSERVER /]$ netstat -ant |grep 900
```

tcp	0	0	192.168.10.160:34921	192.168.10.85:9000	ESTABLISHED
tcp	0	0	192.168.10.160:34926	192.168.10.85:9001	ESTABLISHED

## 5 架构分析

【架构详情请参考：[http://hadoop.apache.org/common/docs/r0.19.2/cn/hdfs\\_design.html](http://hadoop.apache.org/common/docs/r0.19.2/cn/hdfs_design.html)】

Hadoop 有许多元素构成。最底部是 Hadoop Distributed File System (**HDFS**)，它存储 Hadoop 集群中所有存储节点上的文件，与 HDFS 相关的服务有 **NameNode**、**SecondaryNameNode** 及 **DataNode**；HDFS (对于本文) 的上一层是 **MapReduce** 引擎，该引擎由 **JobTrackers** 和 **TaskTrackers** 组成 (所以 MapReduce 相关的服务有 **JobTracker** 和 **TaskTracker** 两种)。

Hadoop 集群中有两种角色：master 与 slave，master 又分为主 master 与次 master。其中：

- 1) 主 master 同时提供 NameNode、SecondaryNameNode 及 JobTracker 三种服务；
- 2) 次 master 只提供 SecondaryNameNode 服务；
- 3) 所有 slave 可以提供 DataNode 或 TaskTracker 两种服务。

### 5.1 HDFS

对外部客户机而言，HDFS 就像一个传统的分级文件系统。可以创建、删除、移动或重命名文件，等等。但是 HDFS 的架构是基于一组特定的节点构建的 (参见图 5-1)，这是由它自身的特点决定的。这些节点包括 NameNode (仅一个)，它在 HDFS 内部提供元数据服务；DataNode，它为 HDFS 提供存储块。

下图为 hadoop 集群的简化视图

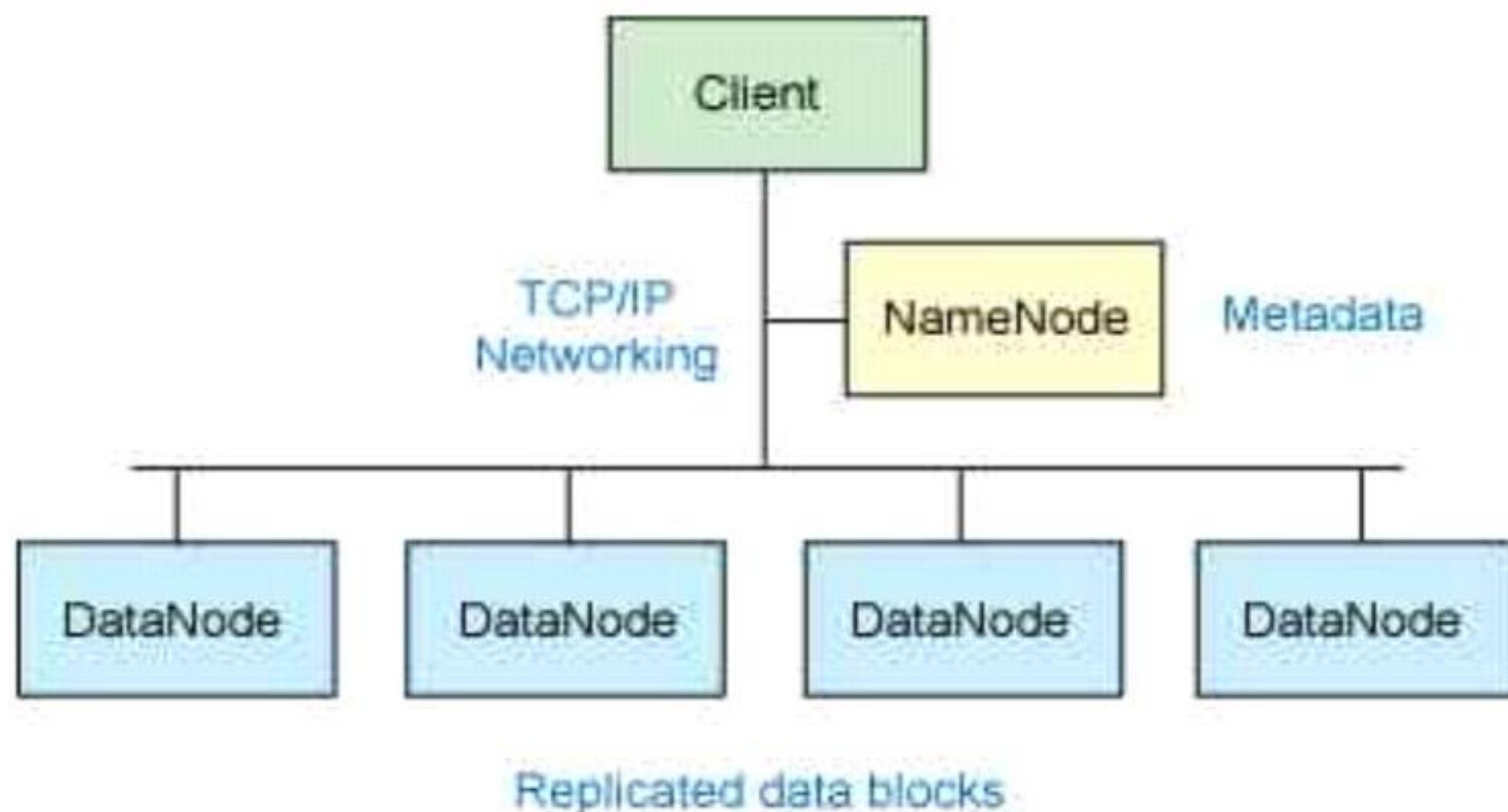


图 5-1. Hadoop 集群的简化视图

存储在 HDFS 中的文件被分成块，然后将这些块复制到多个计算机中 (DataNode)。这与传统的 RAID 架构大不相同。块的大小（通常为 64MB）和复制的块数量在创建文件时由客户机决定。NameNode 可以控制所有文件操作。HDFS 内部的所有通信都基于标准的 TCP/IP 协议。

### 5.1.1 HDFS的三个重要角色

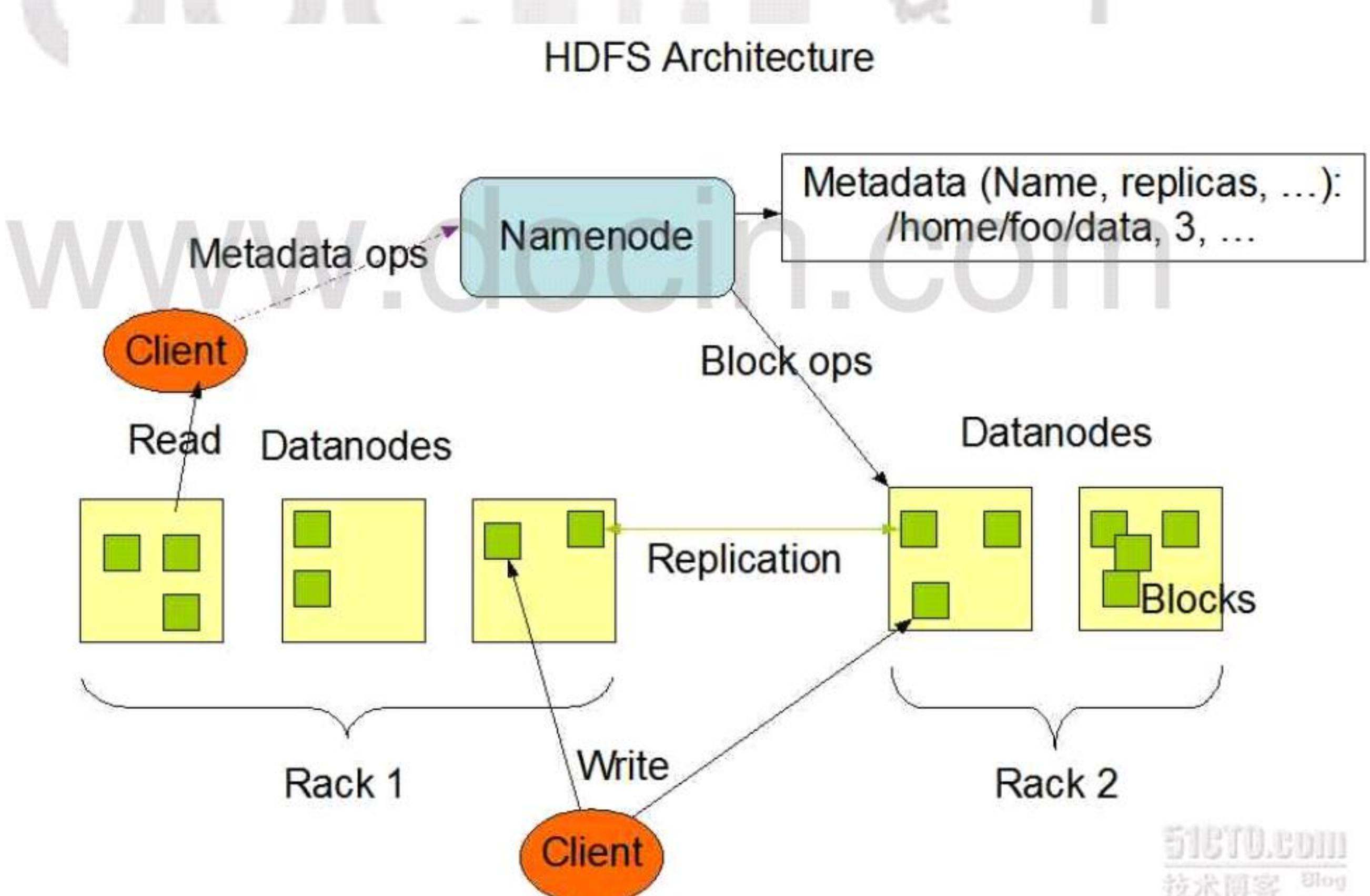


图5-2: HDFS结构示意图

上面这个图很经典，图中展现了整个 HDFS 三个重要角色：NameNode、DataNode 和 Client。

- **NameNode** 可以看作是分布式文件系统中的**管理者**，主要负责管理文件系统的命名空间、集群配置信息和存储块的复制等。NameNode会将文件系统的**Meta-data**存储在内存中，这些信息主要包括了文件信息、每一个文件对应的文件块的信息和每一个文件块在DataNode的信息等。
- **DataNode** 是文件存储的基本单元，它将**Block**存储在本地文件系统中，保存了**Block**的**Meta-data**，同时周期性地将所有存在的**Block**信息发送给**NameNode**。
- **Client** 就是需要获取分布式文件系统文件的应用程序。

这里通过三个操作来说明他们之间的交互关系。

#### 1) 文件写入

- a) Client向NameNode发起文件写入的请求。
- b) NameNode根据文件大小和文件块配置情况，返回给Client它所管理部分 DataNode的信息。
- c) Client将文件划分为多个Block，根据DataNode的地址信息，按顺序写入到每一个 DataNode块中。

#### 2) 文件读取

- a) Client向NameNode发起文件读取的请求。
- b) NameNode返回文件存储的DataNode的信息。
- c) Client读取文件信息。

#### 3) 文件Block复制

- a) NameNode发现部分文件的Block不符合最小复制数或者部分DataNode失效。
- b) 通知DataNode相互复制Block。
- c) DataNode开始直接相互复制。

### 5.1.2 HDFS设计特点

下面说说HDFS的几个设计特点（对于框架设计值得借鉴）：

#### 1. Block的放置

默认不配置。一个Block会有三份备份，一份放在NameNode指定的DataNode，另一份放在与指定DataNode非同一Rack上的DataNode，最后一份放在与指定DataNode同一Rack上的DataNode上。备份无非就是为了数据安全，考虑同一Rack的失败情况以及不同Rack之间数据拷贝性能问题就采用这种配置方式。

#### 2. 心跳检测

心跳检测DataNode的健康状况，如果发现问题就采取数据备份的方式来保证数据的安全性。

#### 3. 数据复制

数据复制（场景为DataNode失败、需要平衡DataNode的存储利用率和需要平衡DataNode数据交互压力等情况）：这里先说一下，使用HDFS的balancer命令，可以配置一个Threshold来平衡每一个DataNode磁盘利用率。例如设置了Threshold为10%，那么执行balancer命令的时候，首先统计所有DataNode的磁盘利用率的均值，然后判断如果某一个DataNode的磁盘利用率超过这个均值Threshold以上，那么将会把这个DataNode的block转移到磁盘利用

率低的DataNode，这对于新节点的加入来说十分有用。

#### 4. 数据校验：

采用CRC32作数据交验。在文件Block写入的时候除了写入数据还会写入交验信息，在读取的时候需要交验后再读入。

#### 5. NameNode是单点

如果失败的话，任务处理信息将会记录在本地文件系统和远端的文件系统中。

#### 6. 数据管道性的写入

当客户端要写入文件到DataNode上，首先客户端读取一个Block然后写到第一个DataNode上，然后由第一个DataNode传递到备份的DataNode上，一直到所有需要写入这个Block的DataNode都成功写入，客户端才会继续开始写下一个Block。

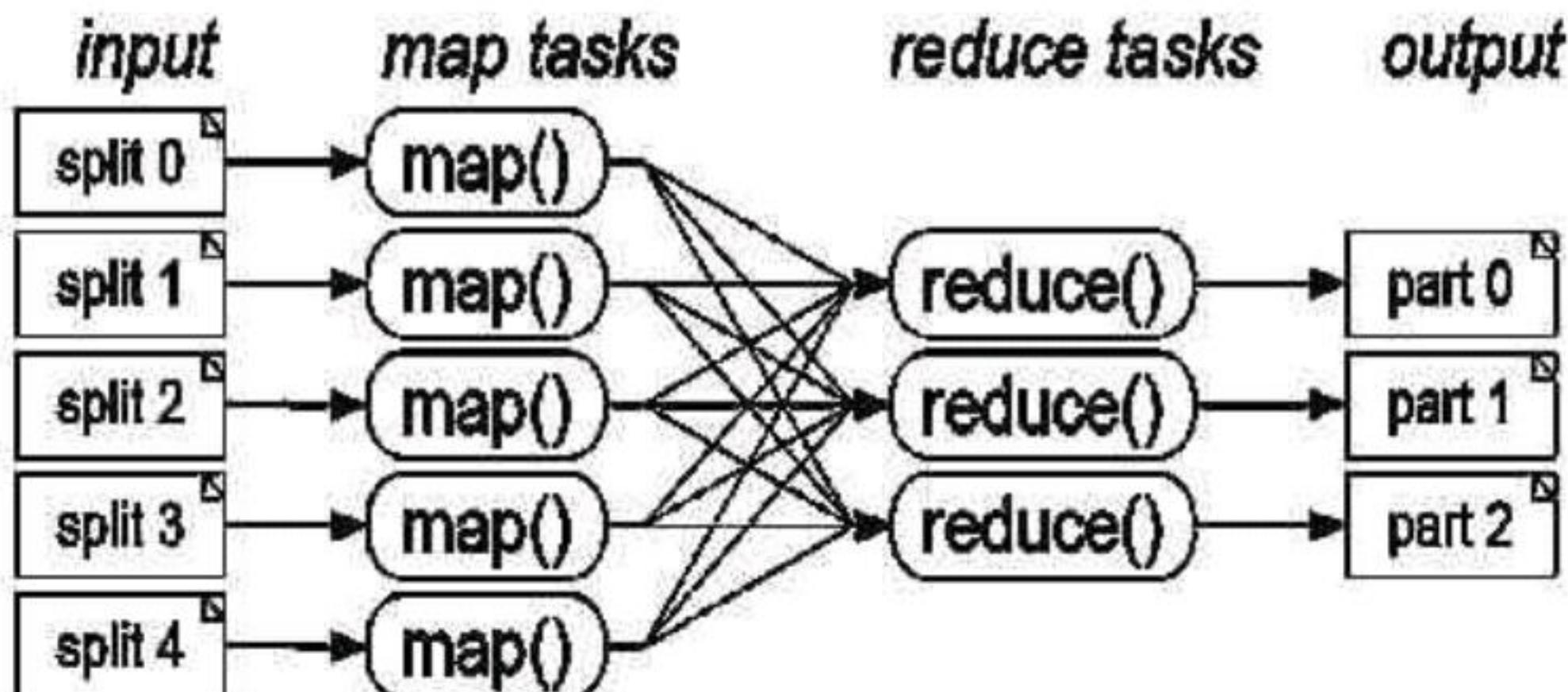
#### 7. 安全模式

安全模式主要是为了系统启动的时候检查各个DataNode上数据块的有效性，同时根据策略必要的复制或者删除部分数据块。在分布式文件系统启动的时候，开始的时候会有安全模式，当分布式文件系统处于安全模式的情况下，文件系统中的内容不允许修改也不允许删除，直到安全模式结束。运行期通过命令也可以进入安全模式。在实践过程中，系统启动的时候去修改和删除文件也会有安全模式不允许修改的出错提示，只需要等待一会儿即可。

### 5.1.3 MapReduce

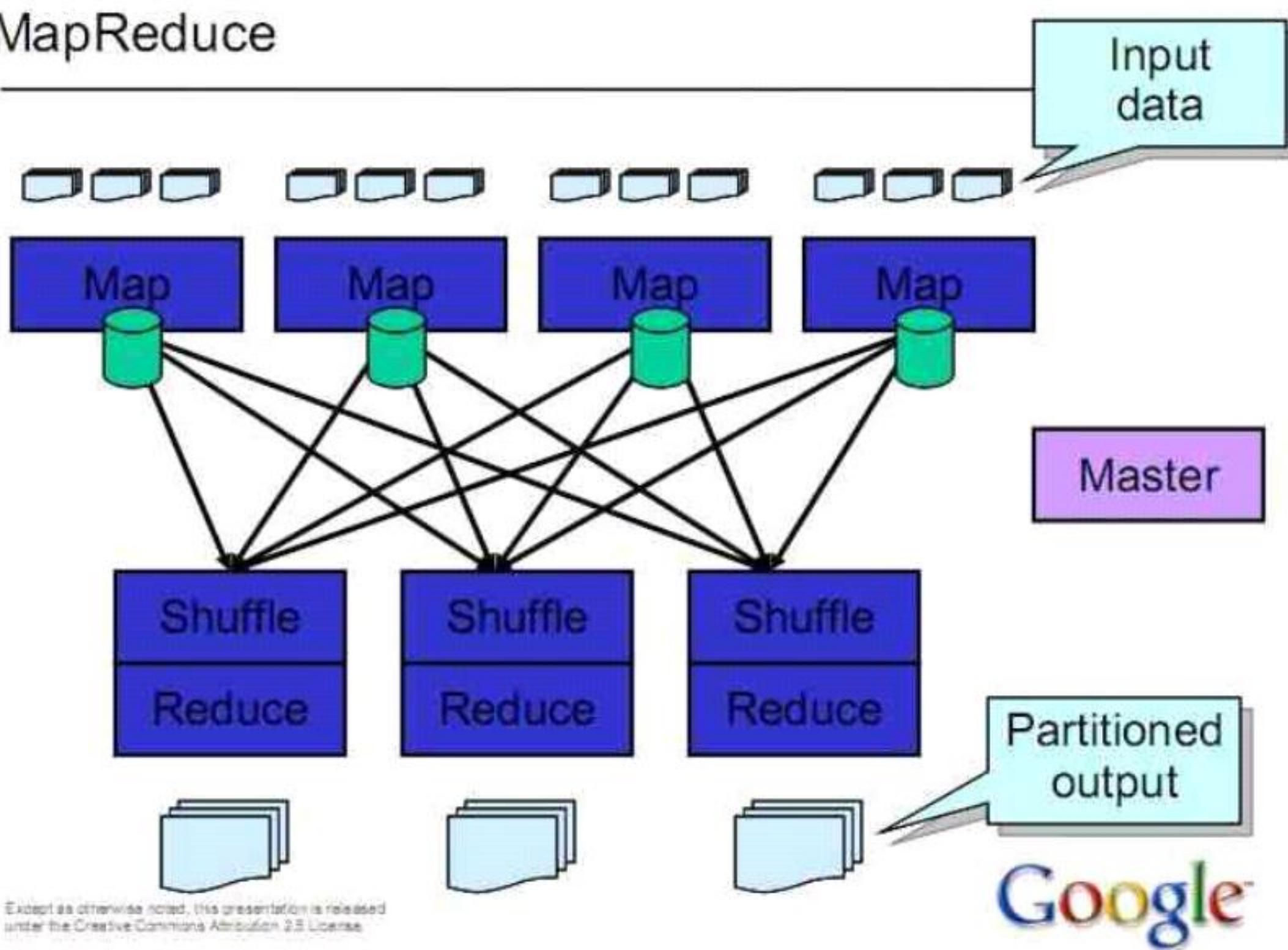
#### 5.1.4 算法介绍

2004年，Google发表了论文，向全世界介绍了MapReduce。2005年初，Nutch的开发者在Nutch上有了一个可工作的MapReduce应用。



5-3 mapreduce结构示意图一

## MapReduce



5-4 mapreduce结构示意图二

MapReduce从它名字上来看就大致可以看出个缘由，两个动词Map和Reduce，“Map（展开）”就是将一个任务分解成为多个任务，“Reduce”就是将分解后多任务处理的结果汇总起来，得出最后的分析结果。

在分布式系统中，机器集群就可以看作硬件资源池，将并行的任务拆分，然后交由每一个空闲机器资源去处理，能够极大地提高计算效率，同时这种资源无关性，对于计算集群的扩展无疑提供了最好的设计保证。（廉价的机器群可以匹敌任何高性能的计算机，纵向扩展的曲线始终敌不过横向扩展的斜线）。任务分解处理以后，那就需要将处理以后的结果再汇总起来，这就是Reduce要做的工作。

具体程序如下：

1) Input输入

从文件中读取原始数据

原始数据  $\langle \text{InputKey}, \text{InputValue} \rangle$

2) Map映射

将原始数据映射成用于Reduce的数据

$\langle \text{InputKey}, \text{InputValue} \rangle \rightarrow \text{List}[\langle \text{MapKey}, \text{MapValue} \rangle]$

3) Reduce合并

将相同Key值的中间数据合并成最终数据

$\langle \text{MapKey}, \text{List}[\text{MapValue}] \rangle \rightarrow \langle \text{OutputKey}, \text{OutputValue} \rangle$

4) Output输出

将最终处理结果输出到文件

$\langle \text{OutputKey}, \text{OutputValue} \rangle \rightarrow \text{结果文件}$

上述就是MapReduce大致处理过程，在Map前还可能会对输入的数据有**Split（分割）**的过程，保证任务并行效率，在Map之后还会有**Shuffle（混合）**的过程，对于提高Reduce的效率以及减小数据传输的压力有很大的帮助。后面会具体提及这些部分的细节。

### 5.1.5 Hadoop框架下的mapreduce

最简单的 MapReduce 应用程序至少包含 3 个部分：一个 Map 函数、一个 Reduce 函数和一个 main 函数。main 函数将作业控制和文件输入/输出结合起来。在这点上，Hadoop 提供了大量的接口和抽象类，从而为 Hadoop 应用程序开发人员提供许多工具，可用于调试和性能度量等。

MapReduce 本身就是用于并行处理大数据集的软件框架。MapReduce 的根源是函数性编程中的 map 和 reduce 函数。它由两个可能包含有许多实例（许多 Map 和 Reduce）的操作组成。Map 函数接受一组数据并将其转换为一个键/值对列表，输入域中的每个元素对应一个键/值对。Reduce 函数接受 Map 函数生成的列表，然后根据它们的键（为每个键生成一个键/值对）缩小键/值对列表。

#### 1. 示例1

假设输入域是 one small step for man, one giant leap for mankind。在这个域上运行 Map 函数将得出以下的键/值对列表：

```
(one, 1) (small, 1) (step, 1) (for, 1) (man, 1)  
(one, 1) (giant, 1) (leap, 1) (for, 1) (mankind, 1)
```

如果对这个键/值对列表应用 Reduce 函数，将得到以下一组键/值对：

```
(one, 2) (small, 1) (step, 1) (for, 2) (man, 1)  
(giant, 1) (leap, 1) (mankind, 1)
```

结果是对输入域中的单词进行计数，这无疑对处理索引十分有用。但是，现在假设有两个输入域，第一个是 one small step for man，第二个是 one giant leap for mankind。您可以在每个域上执行 Map 函数和 Reduce 函数，然后将这两个键/值对列表应用到另一个 Reduce 函数，这时得到与前面一样的结果。换句话说，可以在输入域并行使用相同的操作，得到的结果是一样的，但速度更快。这便是 MapReduce 的威力；它的并行功能可在任意数量的系统上使用。

#### 2. 示例2

Hadoop 提供的范例 Wordcount（计算网页中各个单词的数量）：

- 1) Input: 文本内容 → <行号, 文本内容>
- 2) Map: <行号, 文本内容> → List<<单词, 数量1>>
- 3) Reduce: <单词, List<数量1>> → <单词, 数量合计>
- 4) Output: List<<单词, 数量>> → 文本文件

现在回到 Hadoop 上，它是如何实现这个功能的？

一个代表客户机在单个主系统上启动的 MapReduce 应用程序称为 **JobTracker**。类似于 NameNode，它是 Hadoop 集群中惟一负责控制 MapReduce 应用程序的系统。在应用程

序提交之后，将提供包含在 HDFS 中的输入和输出目录。JobTracker 使用文件块信息（物理量和位置）确定如何创建其他 TaskTracker 从属任务。MapReduce 应用程序被复制到每个出现输入文件块的节点。将为特定节点上的每个文件块创建一个惟一的从属任务。每个 TaskTracker 将状态和完成信息报告给 JobTracker。图 5-5 显示一个示例集群中的工作分布。

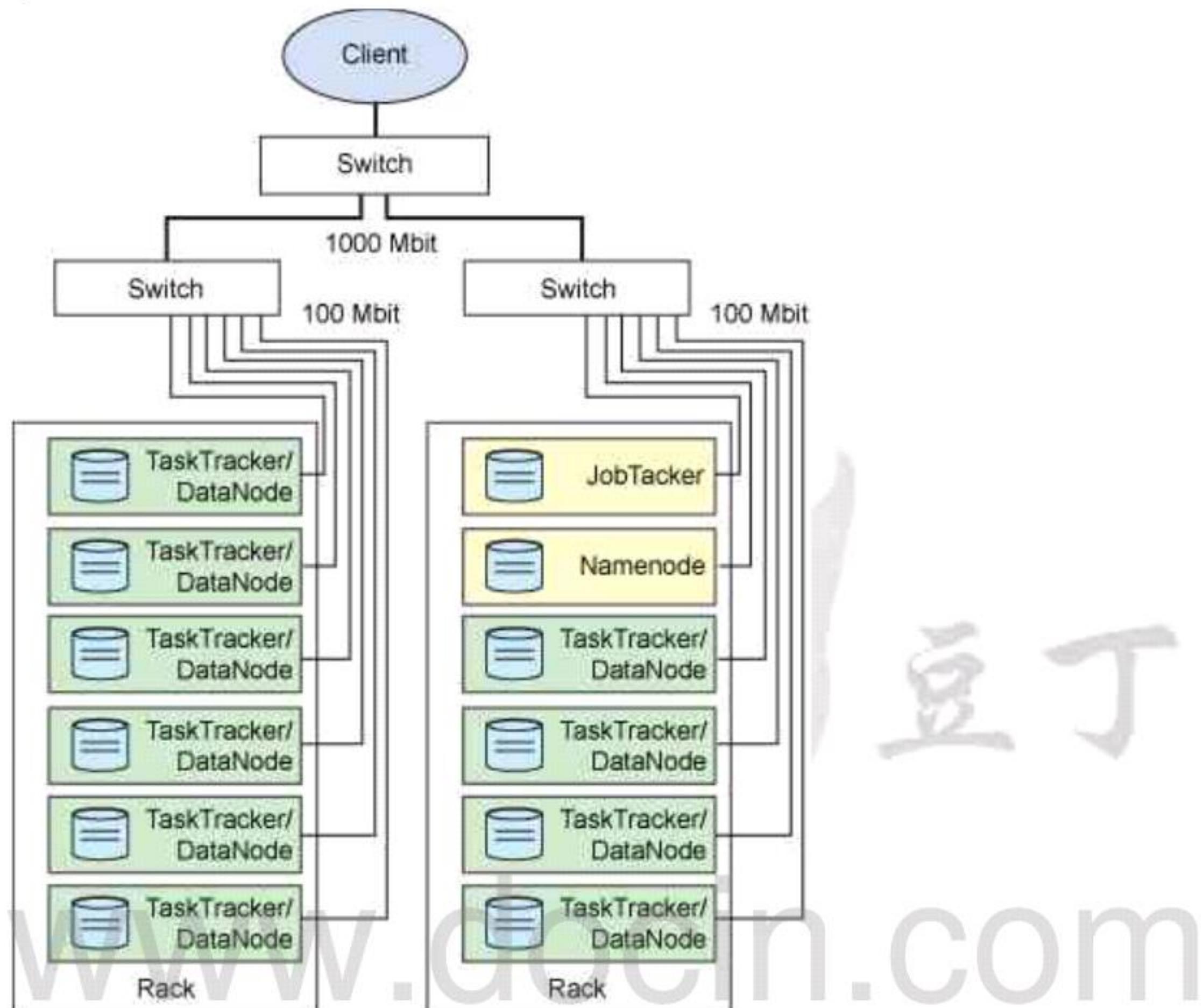


图 5-5. 显示处理和存储的物理分布的 Hadoop 集群

注：

在真实的应用环境中需要做到：

- 1) Namenode 与 JobTacker 要部署在不同的服务器上

## 5.2 综合架构分析

下面综合 MapReduce 和 HDFS 来看 Hadoop 的结构：

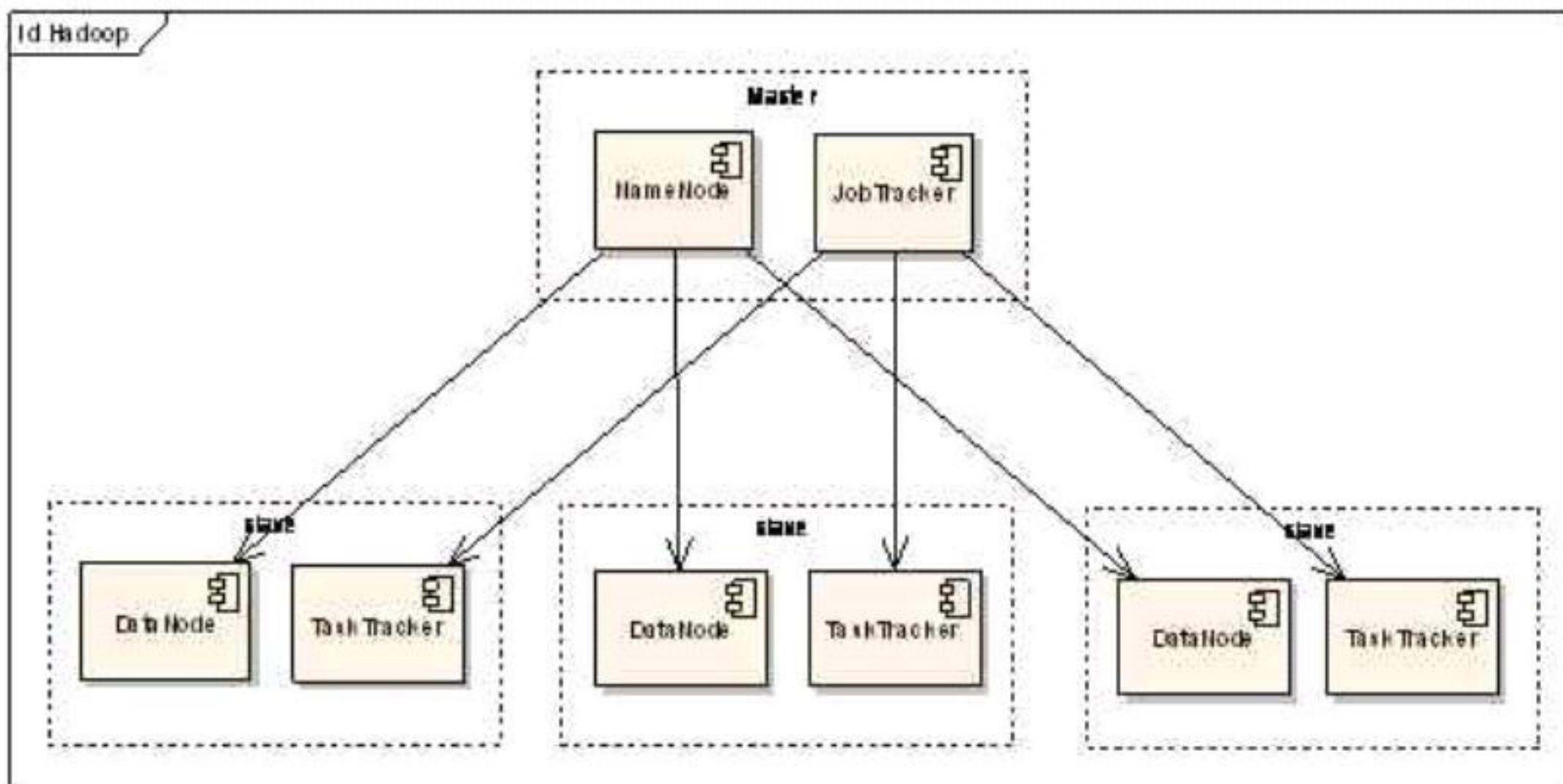


图3：Hadoop结构示意图

在Hadoop的系统中，会有一台**Master**，主要负责**NameNode**的工作以及**JobTracker**的工作。**JobTracker**的主要职责就是启动、跟踪和调度各个**Slave**的任务执行。还会有多台**Slave**，每一台**Slave**通常具有**DataNode**的功能并负责**TaskTracker**的工作。**TaskTracker**根据应用要求来结合本地数据执行**Map**任务以及**Reduce**任务。

说到这里，就要提到分布式计算最重要的一个设计点：**Moving Computation is Cheaper than Moving Data**。就是在分布式处理中，移动数据的代价总是高于转移计算的代价。简单来说就是分而治之的工作，需要将数据也分而存储，本地任务处理本地数据然后归总，这样才会保证分布式计算的高效性。

对外部客户机而言，**HDFS** 就像一个传统的分级文件系统。可以创建、删除、移动或重命名文件，等等。但是 **HDFS** 的架构是基于一组特定的节点构建的（参见图 1），这是由它自身的特点决定的。这些节点包括 **NameNode**（仅一个），它在 **HDFS** 内部提供元数据服务；**DataNode**，它为 **HDFS** 提供存储块。由于仅存在一个 **NameNode**，因此这是 **HDFS** 的一个缺点（单点失败）。

**HDFS**是分布式计算的存储基石，**Hadoop**的分布式文件系统和其他分布式文件系统有很多类似的特质。分布式文件系统基本的几个特点：

- 1) 对于整个集群有单一的命名空间。
- 2) 数据一致性。适合一次写入多次读取的模型，客户端在文件没有被成功创建之前无法看到文件存在。
- 3) 文件会被分割成多个文件块，每个文件块被分配存储到数据节点上，而且根据配置会由复制文件块来保证数据的安全性。

## 6 Hadoop的命令

所有的 hadoop 命令均由 bin/hadoop 脚本引发。不指定参数运行 hadoop 脚本会打印所有命令的描述。

用法: hadoop[--configconfdir][COMMAND][GENERIC\_OPTIONS][COMMAND\_OPTIONS]  
Hadoop 有一个选项解析框架用于解析一般的选项和运行类。

命令选项描述

--configconfdir 覆盖缺省配置目录。缺省是 \${HADOOP\_HOME}/conf。

GENERIC\_OPTIONS 多个命令都支持的通用选项。

COMMAND

命令选项 S 各种各样的命令和它们的选项会在下面提到。这些命令被分为用户命令管理命令两组。

### Hadoop 命令常规选项

下面的选项被 dfsadmin, fs, fsck 和 job 支持。应用程序要实现 Tool 来支持常规选项。

GENERIC\_OPTION 描述

GENERIC_OPTION	描述
-conf <configuration file>	指定应用程序的配置文件。
-D <property=value>	为指定 property 指定值 value。
-fs <local namenode:port>	指定 namenode。
-jt <local jobtracker:port>	指定 job tracker。只适用于 <a href="#">job</a> 。
-files <逗号分隔的文件列表>	指定要拷贝到 map reduce 集群的文件的逗号分隔的列表。 只适用于 <a href="#">job</a> 。
-libjars <逗号分隔的jar列表>	指定要包含到 classpath 中的 jar 文件的逗号分隔的列表。 只适用于 <a href="#">job</a> 。
-archives <逗号分隔的archive列表>	指定要被解压到计算节点上的档案文件的逗号分割的列表。 只适用于 <a href="#">job</a> 。

### 6.1 用户命令

hadoop 集群用户的常用命令。

### 6.1.1 archive

创建一个 hadoop 档案文件。参考 HadoopArchives.

用法: hadoop archive-archiveName<src>\*<dest>

命令选项描述

-archiveNameNAME 要创建的档案的名字。

src 文件系统的路径名，和通常含正则表达式的一样。

dest 保存档案文件的目标目录。

### 6.1.2 distcp

Hadoop 命令 distcp 用于递归地拷贝文件或目录。参考 DistCp 指南以获取等多信息。

用法: hadoop distcp<srcurl><desturl>

命令选项描述

srcurl 源 Url

desturl 目标 Url

### 6.1.3 fs(FSShell命令)

用法: hadoop fs[**GENERIC\_OPTIONS**][**COMMAND\_OPTIONS**]

运行一个常规的文件系统客户端。

各种命令选项可以参考下面的 Hadoop Shell 命令指南

FSShell 命令指南

调用文件系统(FS)Shell 命令应使用 bin/hadoop fs<args>的形式。所有的的 FSshell 命令使用 URI 路径作为参数。URI 格式是 scheme://authority/path。对 HDFS 文件系统，scheme 是 hdfs，对本地文件系统，scheme 是 file。其中 scheme 和 authority 参数都是可选的，如果未加指定，就会使用配置中指定的默认 scheme。一个 HDFS 文件或目录比如 /parent/child 可以表示成 hdfs://namenode:namenodeport/parent/child，或者更简单的 /parent/child（假设你配置文件中的默认值是 namenode:namenodeport）。大多数 FSShell 命令的行为和对应的 UnixShell 命令类似，不同之处会在下面介绍各命令使用详情时指出。出错信息会输出到 stderr，其他信息输出到 stdout。

1) cat

使用方法: hadoop fs -catURI[URI…]

将路径指定文件的内容输出到 stdout。

示例:

```
hadoop fs-cat hdfs://host1:port1/file1hdfs://host2:port2/file2
```

```
hadoop fs-cat file:///file3/user/hadoop/file4
```

返回值:

成功返回 0, 失败返回-1。

#### 2) copyFromLocal

使用方法: hadoop fs -copyFromLocal<localsrc>URI 除了限定源路径是一个本地文件外, 和 put 命令相似。

#### 3) copyToLocal

使用方法: hadoop fs -copyToLocal[-ignorecrc][-crc]URI<localdst>

除了限定目标路径是一个本地文件外, 和 get 命令类似。

#### 4) cp

使用方法: hadoopfs-cpURI[URI…]<dest>

将文件从源路径复制到目标路径。这个 Hadoop Shell 命令允许有多个源路径, 此时目标路径必须是一个目录。

示例:

```
Hadoopfs - cp /user/hadoop/file1/user/hadoop/file2
```

```
hadoopfs - cp /user/hadoop/file1/user/hadoop/file2/user/hadoop/dir
```

返回值:

成功返回 0, 失败返回-1。

#### 5) du

使用方法: hadoop fs - du URI[URI…]

此 Hadoop Shell 命令显示目录中所有文件的大小, 或者当只指定一个文件时, 显示此文件的大小。

示例：

```
Hadoop fs - du  
/user/hadoop/dir1/user/hadoop/file1hdfs://host:port/user/hadoop/dir1
```

返回值：

成功返回 0，失败返回-1。

6) dus

使用方法：hadoop fs -dus<args>

显示文件的大小。

7) expunge

使用方法：hadoop fs -expunge

清空回收站。请参考 HDFS 设计文档以获取更多关于回收站特性的信息。

8) get

使用方法：hadoop fs -get[-ignorecrc][-crc]<src><localdst>

复制文件到本地文件系统。可用-ignorecrc 选项复制 CRC 校验失败的文件。使用-crc 选项复制文件以及 CRC 信息。

示例：

```
hadoop fs - get /user/hadoop/filelocalfile
```

```
hadoop fs - get hdfs://host:port/user/hadoop/filelocalfile
```

返回值：

成功返回 0，失败返回-1。Hadoop Shell 命令还有很多，这里只介绍了其中的一部分。

#### 6.1.4 Fsck

Hadoop 命令主要用来运行 HDFS 文件系统检查工具。参考 Fsck 了解更多。

用法：hadoop

```
fsck[GENERIC_OPTIONS]<path>[-move|-delete|-openforwrite][-files[-blocks[-locations|-racks]]]
```

### 命令选项描述

<path>检查的起始目录。

-move 移动受损文件到/lost+found

-delete 删除受损文件。

-openforwrite 打印出写打开的文件。

-files 打印出正被检查的文件。

-blocks 打印出块信息报告。

-locations 打印出每个块的位置信息。

-racks 打印出 data-node 的网络拓扑结构。

### 6.1.5 jar

Hadoop 命令主要用来运行 jar 文件。用户可以把他们的 MapReduce 代码捆绑到 jar 文件中，使用这个命令执行。

用法：hadoop jar<jar>[mainClass]args...

streaming 作业是通过这个命令执行的。参考 Streamingexamples 中的例子。

Wordcount 例子也是通过 jar 命令运行的。参考 Wordcountexample。

### 6.1.6 job

用于和 MapReduce 作业交互和命令。

用法：

Hadoop

```
job[GENERIC_OPTIONS] |[-submit<job-file>] |[-status<job-id>] |[-counter<job-id><group-name><counter-name>] |[-kill<job-id>] |[-events<job-id><from-event-#><#-of-events>] |[-history[all]<jobOutputDir>] |[-list[all]] |[-kill-task<task-id>] |[-fail-task<task-id>]
```

### 命令选项描述

-submit<job-file>提交作业

-status<job-id>打印 map 和 reduce 完成百分比和所有计数器。

-counter<job-id><group-name><counter-name>打印计数器的值。

-kill<job-id>杀死指定作业。

-events<job-id><from-event-#><#-of-events>打印给定范围内 jobtracker 接收到的事件细节。

-history[all]<jobOutputDir>-history<jobOutputDir>打印作业的细节、失败及被杀死原因的细节。更多的关于一个作业的细节比如成功的任务，做过的任务尝试等信息可以通过指定[all]选项查看。

-list[all]-listall 显示所有作业。-list 只显示将要完成的作业。

-kill-task<task-id>杀死任务。被杀死的任务不会不利于失败尝试。

-fail-task<task-id>使任务失败。被失败的任务会对失败尝试不利。本节有关 Hadoop 命令简单介绍到这里。

## 6.2 管理命令

- [balancer](#)
- [daemonlog](#)
- [datanode](#)
- [dfsadmin](#)
- [jobtracker](#)
- [namenode](#)
- [secondarynamenode](#)
- [tasktracker](#)

详情点击以上链接参考：

### 6.2.1 balancer

运行集群平衡工具。管理员可以简单的按 Ctrl-C 来停止平衡过程。参考 [Rebalancer](#) 了解更多。

用法：hadoop balancer [-threshold <threshold>]

命令选项	描述
-threshold <threshold>	磁盘容量的百分比。这会覆盖缺省的阀值。

### 6.2.2 datanode

运行一个 HDFS 的 datanode。

用法：hadoop datanode [-rollback]

命令选项	描述
-rollback	将datanode回滚到前一个版本。这需要在停止datanode，分发老的hadoop版本之后使用。

### 6.2.3 dfsadmin命令

'bin/hadoop dfsadmin' 命令支持一些和 **HDFS** 管理相关的操作。bin/hadoop dfsadmin -help 命令能列出所有当前支持的命令。比如：

- **-report**: 报告 **HDFS** 的基本统计信息。有些信息也可以在 **NameNode Web** 服务首页看到。
- **-safemode**: 虽然通常并不需要，但是管理员的确可以手动让 **NameNode** 进入或离开安全模式。
- **-finalizeUpgrade**: 删除上一次升级时制作的集群备份。

### 6.2.4 jobtracker

运行 **MapReduce job Tracker** 节点。

用法: hadoop jobtracker

### 6.2.5 namenode

运行 **namenode**。有关升级，回滚，升级终结的更多信息请参考[升级和回滚](#)。

用法: hadoop namenode [-format] | [-upgrade] | [-rollback] | [-finalize] | [-importCheckpoint]

命令选项	描述
-format	格式化namenode。它启动namenode，格式化namenode，之后关闭namenode。
-upgrade	分发新版本的hadoop后，namenode应以upgrade选项启动。
-rollback	将namenode回滚到前一版本。这个选项要在停止集群，分发老的hadoop版本后使用。
-finalize	finalize会删除文件系统的前一状态。最近的升级会被持久化， rollback选项将再不可用，升级终结操作之后，它会停掉namenode。
-importCheckpoint	从检查点目录装载镜像并保存到当前检查点目录，检查点目录由fs.checkpoint.dir指定。

### 6.2.6 secondarynamenode

运行 HDFS 的 secondary namenode。

用 法 : hadoop secondarynamenode [-checkpoint [force]] | [-geteditsize]

命令选项	描述
-checkpoint [force]	如果 EditLog 的大小 $\geq$ fs.checkpoint.size, 启动 Secondary namenode 的检查点过程。如果使用了-force, 将不考虑 EditLog 的大小。
-geteditsize	打印 EditLog 大小。

### 6.2.7 tasktracker

运行 MapReduce 的 task Tracker 节点。

用法: hadoop tasktracker

## 6.3 常用命令

这部分内容其实可以通过命令的 Help 以及介绍了解, 这里主要侧重于介绍一下我用的比较多的几个命令。Hadoop dfs 这个命令后面加参数就是对于 HDFS 的操作, 和 Linux 操作系统的命令很类似, 例如:

- Hadoop dfs -ls 就是查看/usr/root 目录下的内容, 默认如果不填路径这就是当前用户路径;
- Hadoop dfs -rmr xxx 就是删除目录, 还有很多命令看看就很容易上手;
- Hadoop dfsadmin -report 这个命令可以全局的查看 DataNode 的情况;
- Hadoop job 后面增加参数是对于当前运行的 Job 的操作, 例如 list,kill 等;
- Hadoop balancer 就是前面提到的均衡磁盘负载的命令。

## 7 Hadoop的应用

Hadoop 是一个用于开发分布式应用程序的多功能框架; 从不同的角度看待问题是充分利用 Hadoop 的好方法。

Hadoop 一直帮助解决各种问题, 包括超大型数据集的排序和大文件的搜索。它还是各种搜索引擎的核心, 比如 Amazon 的 A9 和用于查找酒信息的 Able Grape 垂直搜索引擎。

Hadoop 的最常见用法之一是 Web 搜索。虽然它不是惟一的软件框架应用程序，但作为一个并行数据处理引擎，它的表现非常突出。Hadoop 最有趣的方面之一是 Map and Reduce 流程。这个流程称为创建索引，它将 Web 爬行器检索到的文本 Web 页面作为输入，并且将这些页面上的单词的频率报告作为结果。然后可以在整个 Web 搜索过程中使用这个结果从已定义的搜索参数中识别内容。

当前，Yahoo! 拥有最大的 Hadoop Linux 生产架构，共由 **10,000** 多个内核组成，有超过 **5PB** 字节的储存分布到各个 DataNode。在它们的 Web 索引内部差不多有一万亿个链接。不过您可能不需要那么大型的系统，如果是这样的话，您可以使用 Amazon Elastic Compute Cloud (EC2) 构建一个包含 20 个节点的虚拟集群。事实上，纽约时报 使用 Hadoop 和 EC2 在 36 个小时内将 4TB 的 TIFF 图像（包括 405K 大 TIFF 图像，3.3M SGML 文章和 405K XML 文件）转换为 800K 适合在 Web 上使用的 PNG 图像。这种处理称为云计算，它是一种展示 Hadoop 的威力的独特方式。

## 8 系统维护

### 8.1 Hadoop的系统监控

通过从 hadoop 的 hadoop-metrics 文件中就可以知道 hadoop 对 Ganglia 是有支持的，而且很多运维 hadoop 集群的都是使用的这个作为监控管理工具，不过也有其他解决方案，例如 hadoop 自家的 Chukwa 。以下收集了一些比较好的资料以作参考：

**Chukwa** 在百度的应用实践

<http://hi.baidu.com/ops%5Fbd/blog/item/7dd0d6374675e08aa8018e31.html>

**hadoop** 状态分析系统 **chukwa**

<http://hi.baidu.com/ops%5Fbd/blog/item/5f39abde48a10f3f10df9b12.html>

**hadoop ganglia configuration**

<http://hi.baidu.com/hovlj%5F1130/blog/item/e8fe89c3e9a67e160ff47755.html>

**Ganglia** 和 **Nagios**，第 1 部分：用 **Ganglia** 监视企业集群

<http://www.ibm.com/developerworks/cn/linux/l-ganglia-nagios-1/>

**Ganglia** 和 **Nagios**，第 2 部分：使用 **Nagios** 监视企业集群

<http://www.ibm.com/developerworks/cn/linux/l-ganglia-nagios-2/index.html>

## hadoop wiki ganglia

<http://wiki.apache.org/hadoop/GangliaMetrics>

## monitoring-hadoop-clusters-using-ganglia

<http://www.ryangreenhall.com/2010/10/monitoring-hadoop-clusters-using-ganglia/>

### 8.2 NameNode与JobTracker单点故障说明

NameNode与JobTracker在 Hadoop集群中都是唯一的。一旦NameNode与JobTracker有一个瘫痪或者说主master瘫痪，整个Hadoop也就瘫痪了。对于主master的维护工作（如备份、重启等）应该说十分重要。这属于单点故障，我们这在这里主要要解决两个问题：一是数据的备份，二是故障时节点的切换。

- 1) 目前JobTracker在单点故障方面没有做任何工作，所以当发生时，必须人工去切换，并重执行未完成的Job。
- 2) NameNode目前只做了数据备份，也就是配置的SecondaryNameNode，自动切换也没有做，需要自己去实现，当发现NameNode故障后，停止它并启动SecondaryNameNode，然后告诉所有slave连接到新的Master。可以直接使用slave中配置的master的IP，或者使用域名等手段。

### 8.3 经验总结

经验总结和注意事项（这部分是我在使用过程中花了一些时间走的弯路）：

- 1) Master 和 Slave 上的几个 conf 配置文件不需要全部同步，如果确定都是通过 Master 去启动和关闭，那么 Slave 机器上的配置不需要去维护。但如果希望在任意一台机器都可以启动和关闭 Hadoop，那么就需要全部保持一致了。
- 2) Master 和 Slave 机器上的/etc/hosts 中必须把集群中机器都配置上去，就算在各个配置文件中使用的是 IP。这个吃过不少苦头，原来以为如果配成 IP 就不需要去配置 Host，结果发现在执行 Reduce 的时候总是卡住，在拷贝的时候就无法继续下去，不断重试。另外如果集群中如果有两台机器的机器名如果重复也会出现问题。
- 3) 如果在新增了节点或者删除节点的时候出现了问题，首先就去删除 Slave 的 hadoop.tmp.dir，然后重新启动试试看，如果还是不行那就干脆把 Master 的 hadoop.tmp.dir 删除（意味着 dfs 上的数据也会丢失），如果删除了 Master 的 hadoop.tmp.dir，那么就需要重新 namenode -format。
- 4) Map 任务个数以及 Reduce 任务个数配置。前面分布式文件系统设计提到一个文件被放入到分布式文件系统中，会被分割成多个 block 放置到每一个的 DataNode 上，默认 dfs.block.size 应该是 64M，也就是说如果你放置到 HDFS 上的数据小于 64，那么将只有一个 Block，此时会被放置到某一个 DataNode 中，这个可以通过使用命令：hadoop fsadmin -report 就可以看到各个节点存储的情况。也可以直接去某一个 DataNode 查看目录：hadoop.tmp.dir/dfs/data/current 就可以看到那些 block 了。Block 的数量将会直接影响到 Map 的个数。当然可以通过配置来设定 Map 和 Reduce 的任务个数。Map 的个数通常默认和 HDFS 需要处理的

**blocks** 相同。也可以通过配置 **Map** 的数量或者配置 **minimum split size** 来设定，实际的个数为： $\max(\min(\text{block\_size}, \text{data}/\#\text{maps}), \text{min\_split\_size})$ 。**Reduce** 可以通过这个公式计算：

`0.95*num_nodes*mapred.tasktracker.tasks.maximum.`

总的来说出了问题或者启动的时候最好去看看日志，这样心里有底。

## 8.4 如何在一个hadoop集群新增或删除一些机器而不重启

### 8.4.1 新增节点

操作如下：

首先，把新节点的 IP或主机名 加入主节点（master）的 conf/slaves 文件。

然后登录新的从节点，执行以下命令：

```
$ cd $HADOOP_HOME  
$ bin/hadoop-daemon.sh start datanode  
$ bin/hadoop-daemon.sh start tasktracker
```

然后就可以在namenode机器上运行balancer，执行负载均衡

```
$bin/hadoop balancer  
(或者: bin/start-balancer.sh)
```

运行bin/start-balancer.sh，这个会很耗时间

备注：

1. 如果不balance，那么cluster会把新的数据都存放在新的node上，这样会降低mr的工作效率；
2. 也可调用bin/start-balancer.sh 命令执行，也可加参数 `-threshold 5`  
`threshold` 是平衡阈值，默认是10%，值越低各节点越平衡，但消耗时间也更长。
3. balancer也可以在有mr job的cluster上运行，默认dfs.balance.bandwidthPerSec 很低，为1M/s。在没有mr job时，可以提高该设置加快负载均衡时间。

其他备注：

1. 必须确保slave的firewall已关闭；
2. 确保新的slave的ip已经添加到master及其他slaves的/etc/hosts中，反之也要将 master及其他slave的ip添加到新的slave的/etc/hosts中

注：在0.21中执行bin/hadoop-daemon.sh start datanode 会提示该命令已废除，建议使用./hdfs datanode 命令，但是用后者反而会抛异常。

### 8.4.2 删除节点

比如我原来有 10 个节点的集群。现在我想停掉 2 个，但数据不能丢失，只能让它们的数据转移到其它 8 台上。这道过程就是 decommission。我们不能直接把那 2 台停掉，要在停掉之前把数据转移出去。

### 1) 确定要下架的机器

首先在 namenode 服务器, 在 HADOOP\_HOME 目录下建立一个 excludes 文件. 它是一个文本, 里面每行就是想要停掉的主机名或 IP. 这里 excludes 放在 \$HADOOP\_HOME 下

例如

```
192.168.10.160
```

这样就表明要把 1 台机器给停了.

### 2) 修改 **conf/hdfs-site.xml** 文件

然后在 conf/ core-site.xml 添加这样的属性:

```
<property>
<name>dfs.hosts.exclude</name>
<value>excludes</value>
</property>
```

### 3) 强制重新加载配置

做完这步, 用 bin/hadoop dfsadmin -refreshNodes 命令更新结点以及 hadoop 配置  
然后你可以通过 bin/hadoop dfsadmin -report 就可以查到类似这样的信息:

```
Datanodes available: 2 (2 total, 0 dead)

Name: 192.168.10.160:50010
Decommission Status : Decommission in progress
Configured Capacity: 37169479680 (34.62 GB)
DFS Used: 94208 (92 KB)
Non DFS Used: 30112800768 (28.04 GB)
DFS Remaining: 7056584704 (6.57 GB)
DFS Used%: 0%
DFS Remaining%: 18.98%
Last contact: Tue Jul 26 15:31:17 CST 2011

Name: 192.168.10.245:50010
Decommission Status : Normal
Configured Capacity: 253935173632 (236.5 GB)
DFS Used: 94208 (92 KB)
Non DFS Used: 188234915840 (175.31 GB)
DFS Remaining: 65700163584 (61.19 GB)
DFS Used%: 0%
DFS Remaining%: 25.87%
```

Last contact: Tue Jul 26 15:31:16 CST 2011

可以看到 160 在停止中 (Decommission Status : Decommission in progress ) , 执行完成会显示 (Decommission Status : Decommissioned) 。

注：在没停止之前，mapreduce 会拒绝操作。会出异常的

#### 4) 再次编辑**excludes**文件

一旦完成了机器下架，它们就可以从**excludes**文件移除了

登录要下架的机器，会发现**DataNode**进程没有了，但是**TaskTracker**依然存在，需要手工终止 (kill) 一下。

注：附带说一下 **-refreshNodes** 命令的另外三种用途：

2. 添加允许的节点到列表中（添加主机名到 **dfs.hosts** 里来）
3. 直接去掉节点，不做数据副本备份（在 **dfs.hosts** 里去掉主机名）
4. 退服的逆操作——停止 **exclude** 里面和 **dfs.hosts** 里面都有的，正在进行 **decommission** 的节点的退服，也就是把 **Decommission in progress** 的节点重新变为 **Normal**（在 web 界面叫 **in service**）

## 8.5 其它日常问题说明

### 8.5.1 datanode启动失败，各slave节点的namespaceIDs与masters不同

```
... ERROR org.apache.hadoop.dfs.DataNode: java.io.IOException: Incompatible namespaceIDs in
/app/hadoop/tmp/dfs/data: namenode namespaceID = 308967713; datanode namespaceID = 113030094
    at org.apache.hadoop.dfs.DataStorage.doTransition(DataStorage.java:281)
    at org.apache.hadoop.dfs.DataStorage.recoverTransitionRead(DataStorage.java:121)
    at org.apache.hadoop.dfs.DataNode.startDataNode(DataNode.java:230)
    at org.apache.hadoop.dfs.DataNode.(DataNode.java:199)
    at org.apache.hadoop.dfs.DataNode.makeInstance(DataNode.java:1202)
    at org.apache.hadoop.dfs.DataNode.run(DataNode.java:1146)
    at org.apache.hadoop.dfs.DataNode.createDataNode(DataNode.java:1167)
    at org.apache.hadoop.dfs.DataNode.main(DataNode.java:1326)
```

Workaround 1: Start from scratch

I can testify that the following steps solve this error, but the side effects won't make you happy (me neither). The crude workaround I have found is to:

Stop the cluster

Delete the data directory on the problematic DataNode: the directory is specified by **dfs.data.dir** in **conf/hdfs-site.xml**; if you followed this tutorial, the relevant directory is **/app/hadoop/tmp/dfs/data**

Reformat the NameNode (NOTE: all HDFS data is lost during this process!)

Restart the cluster

When deleting all the HDFS data and starting from scratch does not sound like a good idea (it might be ok during the initial setup/testing), you might give the second approach a try.

#### Workaround 2: Updating namespaceID of problematic DataNodes

Big thanks to Jared Stehler for the following suggestion. I have not tested it myself yet, but feel free to try it out and send me your feedback. This workaround is “minimally invasive” as you only have to edit one file on the problematic DataNodes:

Stop the DataNode

Edit the value of namespaceID in /current/VERSION to match the value of the current NameNode

Restart the DataNode

If you followed the instructions in my tutorials, the full path of the relevant files are:

NameNode: </app/hadoop/tmp/dfs/name/current/VERSION>

DataNode: </app/hadoop/tmp/dfs/data/current/VERSION>

(background: dfs.data.dir is by default set to \${hadoop.tmp.dir}/dfs/data, and we set hadoop.tmp.dir in this tutorial to /app/hadoop/tmp).

If you wonder how the contents of VERSION look like, here's one of mine:

# contents of /current/VERSION

```
namespaceID=393514426
storageID=DS-1706792599-10.10.10.1-50010-1204306713481
cTime=1215607609074
storageType=DATA_NODE
layoutVersion=-13
```

### 8.5.2 taskTracker和jobTracker 启动失败

Can not start task tracker because java.lang.RuntimeException: Not a host:port pair: local

需配置mapred.job.tracker属性，在mapred-site.xml的configuration节点中配置(job-tracker host为jobTracker服务器的ip或域名)

```
<property>
    <name>mapred.job.tracker</name>
    <value>[job-tracker host]:9001</value>
</property>
```

### 8.5.3 Shuffle Error: Exceeded MAX\_FAILED\_UNIQUE\_FETCHES; bailing-out

Answer:

程序里面需要打开多个文件，进行分析，系统一般默认数量是1024，（用ulimit -a可以看到）对于正常使用是够了，但是对于程序来讲，就太少了。

修改办法：

修改2个文件。

```
/etc/security/limits.conf  
vi /etc/security/limits.conf  
加上:  
* soft nofile 102400  
* hard nofile 409600  
  
$cd /etc/pam.d/  
$sudo vi login  
添加 session required /lib/security/pam_limits.so
```

针对第一个问题我纠正下答案：

这是reduce预处理阶段shuffle时获取已完成的map的输出失败次数超过上限造成的，上限默认为5。引起此问题的方式可能会有很多种，比如网络连接不正常，连接超时，带宽较差以及端口阻塞等。。。通常框架内网络情况较好是不会出现此错误的。

### 8.5.4 Too many fetch-failures

Answer:

出现这个问题主要是结点间的连通不够全面。

1) 检查 /etc/hosts

要求本机ip 对应 服务器名

要求要包含所有的服务器ip + 服务器名

2) 检查 .ssh/authorized\_keys

要求包含所有服务器（包括其自身）的public key

3: 处理速度特别的慢 出现map很快 但是reduce很慢 而且反复出现 reduce=0%

Answer:

结合第二点，然后

修改 conf/hadoop-env.sh 中的export HADOOP\_HEAPSIZE=4000

### 8.5.5 能够启动datanode，但无法访问，也无法结束的错误

在重新格式化一个新的分布式文件时，需要将你NameNode上所配置的dfs.name.dir这一namenode用来存放NameNode 持久存储名字空间及事务日志的本地文件系统路径删除，同时将各DataNode上的dfs.data.dir的路径 DataNode 存放块数据的本地文件系统

路径的目录也删除。如本此配置就是在**NameNode**上删除/home/hadoop/NameData，在**DataNode**上删除/home/hadoop/DataNode1和/home/hadoop/DataNode2。这是因为**Hadoop**在格式化一个新的分布式文件系统时，每个存储的名字空间都对应了建立时间的那个版本（可以查看/home/hadoop/NameData/current目录下的VERSION文件，上面记录了版本信息），在重新格式化新的分布式系统文件时，最好先删除**NameData**目录。必须删除各**DataNode**的dfs.data.dir。这样才可以使**namedode**和**datanode**记录的信息版本对应。

注意：删除是个很危险的动作，不能确认的情况下不能删除！！做好删除的文件等通通备份！！

#### 8.5.6 java.io.IOException: Could not obtain block:

blk\_194219614024901469\_1100

file=/user/hive/warehouse/src\_20090724\_log/src\_20090724\_log

出现这种情况大多是结点断了，没有连接上。

#### 8.5.7 java.lang.OutOfMemoryError: Java heap space

出现这种异常，明显是jvm内存不够得原因，要修改所有的**datanode**的jvm内存大小。

Java -Xms1024m -Xmx4096m

一般jvm的最大内存使用应该为总内存大小的一半，我们使用的8G内存，所以设置为4096m，这一值可能依旧不是最优的值

#### 8.5.8 解决hadoop OutOfMemoryError问题：

```
<property>
  <name>mapred.child.java.opts</name>
  <value>-Xmx800M -server</value>
</property>
```

With the right JVM size in your hadoop-site.xml , you will have to copy this to all mapred nodes and restart the cluster.

或者：hadoop jar jarfile [main class] -D mapred.child.java.opts=-Xmx800M

#### 8.5.9 Hadoop java.io.IOException:

Job failed! at org.apache.hadoop.mapred.JobClient.runJob(JobClient.java:1232)  
while indexing.

when i use nutch1.0,get this error:

Hadoop java.io.IOException: Job failed! at  
org.apache.hadoop.mapred.JobClient.runJob(JobClient.java:1232) while  
indexing.

这个也很好解决：

可以删除conf/log4j.properties，然后可以看到详细的错误报告

我这儿出现的是out of memory

解决办法是在给运行主类org.apache.nutch.crawl.Crawl加上参数: -Xms64m  
-Xmx512m

你的或许不是这个问题，但是能看到详细的错误报告问题就好解决了

## 8.6 防火墙的端口开放要求

部署 Hadoop 集群时，master 与 slave 的防火墙均要关闭。关闭防火墙的根本目的也是为了图省事儿，因为在使用 HDFS 与 MapReduce 时，Hadoop 会打开许多监听端口。它们分别是：

### 8.6.1 与HDFS有关的地址及端口属性

#### 1. **fs.default.name**

位 置：conf/core-site.xml

必 须 项：是

常 用 值：hdfs:// [域名或IP地 址] :9000

说 明：NameNode 主服务器的地址

- 必须在所有 master 及 slave 上的 conf/core-site.xml 中设置此项。并且因为 Hadoop 架构是主 master 模式，所以在一个集群中的所有 master 及 slave 上设置的 fs.default.name 值应该是唯一一个 NameNode 主服务器的地址。

#### 2. **dfs.datanode.address**

位 置：conf/hdfs-site.xml

必 须 项：否

默 认 值：0.0.0.0:50010

说 明：DataNode 服务的地址

#### 3. **dfs.datanode.ipc.address**

位 置：conf/hdfs-site.xml

必 须 项：否

默 认 值：0.0.0.0:50020

说 明：DataNode IPC服务的地址

#### 4. **dfs.http.address**

位 置：conf/hdfs-site.xml

必 须 项：否

默认值: 0.0.0.0:**50070**

说明: NameNode HTTP状态监视地址

### 5. **dfs.secondary.http.address**

位 置: conf/hdfs-site.xml

必须项: 否

默认值: 0.0.0.0:**50090**

说明: SecondaryNameNode HTTP状态监视地址

### 6. **dfs.datanode.http.address**

位 置: conf/hdfs-site.xml

必须项: 否

默认值: 0.0.0.0:**50075**

说明: DataNode HTTP状态监视地址

## 8.6.2 与MapReduce 有关的地址及端口属性

### 1. **mapred.job.tracker**

位 置: conf/mapred-site.xml

必须项: 是

常用值: 【域名或IP地址】:**9001**

说 明: JobTracker 主服务器地址及端口

- 必须在所有 master 及 slave 上的 conf/mapred-site.xml 中设置此项。并且因为 Hadoop 架构是主 master 模式，所以在一个集群中的所有 master 及 slave 上设置的 mapred.job.tracker 的值应该是唯一一个 JobTracker 主服务器的地址。

### 2. **mapred.task.tracker.report.address**

位 置: conf/mapred-site.xml

必须项: 否

默认值: 127.0.0.1:0

说明: 提交报告用TaskTracker 服务地址

### 3. **mapred.job.tracker.http.address**

位 置: conf/mapred-site.xml

必须项: 否

默认值: 0.0.0.0:**50030**

说明：JobTracker HTTP 状态监视地址

#### 4. mapred.task.tracker.http.address

位置：conf/mapred-site.xml

必须项：否

默认值：0.0.0.0:50060

说明：TaskTracker HTTP 状态监视地址

## 9 附录

### 9.1 Hadoop历史

Hadoop 这个名字不是一个缩写，它是一个虚构的名字。该项目的创建者，Doug Cutting 如此解释 Hadoop 的得名：“这个名字是我孩子给一头吃饱了的棕黄色大象命名的。我的命名标准就是简短，容易发音和拼写，没有太多的意义，并且不会被用于别处。小孩子是这方面的高手。Googol 就是由小孩命名的。”

Hadoop 及其子项目和后继模块所使用的名字往往也与其功能不相关，经常用一头大象或其他动物主题(例如："Pig")。较小的各个组成部分给与更多描述性(因此也更俗)的名称。这是一个很好的原则，因为它意味着可以大致从其名字猜测其功能，例如，jobtracker 的任务就是跟踪 MapReduce 作业。

从头开始构建一个网络搜索引擎是一个雄心勃勃的目标，不只是要编写一个复杂的、能够抓取和索引网站的软件，还需要面临着没有专有运行团队支持运行它的挑战，因为它有那么多独立部件。同样昂贵的还有：据 Mike Cafarella 和 Doug Cutting 估计，一个支持此 10 亿页的索引需要价值约 50 万美元的硬件投入，每月运行费用还需要 3 万美元。不过，他们相信这是一个有价值的目标，因为这会开放并最终使搜索引擎算法普及化。

Nutch 项目开始于 2002 年，一个可工作的抓取工具和搜索系统很快浮出水面。但他们意识到，他们的架构将无法扩展到拥有数十亿网页的网络。在 2003 年发表的一篇描述 Google 分布式文件系统(简称 GFS)的论文为他们提供了及时的帮助，文中称 Google 正在使用此文件系统。GFS 或类似的东西，可以解决他们在网络抓取和索引过程中产生的大量的文件的存储需求。具体而言，GFS 会省掉管理所花的时间，如管理存储节点。在 2004 年，他们开始写一个开放源码的应用，即 Nutch 的分布式文件系统(NDFS)。

2004 年，Google 发表了论文，向全世界介绍了 MapReduce。2005 年初，Nutch 的开发者在 Nutch 上有了一个可工作的 MapReduce 应用，到当年年中，所有主要的 Nutch 算法被移植到使用 MapReduce 和 NDFS 来运行。

Nutch 中的 NDFS 和 MapReduce 实现的应用远不只是搜索领域，在 2006 年 2 月，他们从 Nutch 转移出来成为一个独立的 Lucene 子项目，称为 Hadoop。大约在同一时间，Doug Cutting 加入雅虎，Yahoo 提供一个专门的团队和资源将 Hadoop 发展成一个可在网络上运行的系统(见后文的补充材料)。在 2008 年 2 月，雅虎宣布其搜索引擎产品部署在一个拥有 1 万个内核的 Hadoop 集群上。

2008 年 1 月，Hadoop 已成为 Apache 顶级项目，证明它是成功的，是一个多样化、活跃的社区。通过这次机会，Hadoop 成功地被雅虎之外的很多公司应用，如 Last.fm、Facebook 和《纽约时报》。(一些应用



在第 14 章的案例研究和 Hadoop 维基有介绍，Hadoop 维基的网址为  
<http://wiki.apache.org/hadoop/PoweredBy>。)

有一个良好的宣传范例，《纽约时报》使用亚马逊的 EC2 云计算将 4 TB 的报纸扫描文档压缩，转换为用于 Web 的 PDF 文件。这个过程历时不到 24 小时，使用 100 台机器运行，如果不结合亚马逊的按小时付费的模式(即允许《纽约时报》在很短的一段时间内访问大量机器)和 Hadoop 易于使用的并行程序设计模型，该项目很可能不会这么快开始启动。

2008 年 4 月，Hadoop 打破世界纪录，成为最快排序 1TB 数据的系统。运行在一个 910 节点的群集，Hadoop 在 209 秒内排序了 1 TB 的数据(还不到三分半钟)，击败了前一年的 297 秒冠军。同年 11 月，谷歌在报告中声称，它的 MapReduce 实现执行 1TB 数据的排序只用了 68 秒。在 2009 年 5 月，有报道宣称 Yahoo 的团队使用 Hadoop 对 1 TB 的数据进行排序只花了 62 秒时间。

构建互联网规模的搜索引擎需要大量的数据，因此需要大量的机器来进行处理。Yahoo! Search 包括四个主要组成部分：Crawler，从因特网下载网页；WebMap，构建一个网络地图；Indexer，为最佳页面构建一个反向索引；Runtime(运行时)，回答用户的查询。WebMap 是一幅图，大约包括一万亿条边(每条代表一个网络链接)和一千亿个节点(每个节点代表不同的网址)。创建和分析此类大图需要大量计算机运行若干天。在 2005 年初，WebMap 所用的基础设施名为 Dreadnaught，需要重新设计以适应更多节点的需求。Dreadnaught 成功地从 20 个节点扩展到 600 个，但需要一个完全重新的设计，以进一步扩大。Dreadnaught 与 MapReduce 有许多相似的地方，但灵活性更强，结构更少。具体说来，每一个分段(fragment)，Dreadnaught 作业可以将输出发送到此作业下一阶段中的每一个分段，但排序是在库函数中完成的。在实际情形中，大多数 WebMap 阶段都是成对存在的，对应于 MapReduce。因此，WebMap 应用并不需要为了适应 MapReduce 而进行大量重构。

Eric Baldeschwieler(Eric14)组建了一个小团队，我们开始设计并原型化一个新的框架(原型为 GFS 和 MapReduce，用 C++语言编写)，打算用它来替换 Dreadnaught。尽管当务之急是我们需要一个 WebMap 新框架，但显然，标准化对于整个 Yahoo! Search 平台至关重要，并且通过使这个框架泛化，足以支持其他用户，我们才能够充分运用对整个平台的投资。

与此同时，我们在关注 Hadoop(当时还是 Nutch 的一部分)及其进展情况。2006 年 1 月，雅虎聘请了 Doug Cutting，一个月后，我们决定放弃我们的原型，转而使用 Hadoop。相较于我们的原型和设计，Hadoop 的优势在于它已经在 20 个节点上实际应用过。这样一来，我们便能在两个月内搭建一个研究集群，并着手帮助真正的客户使用这个新的框架，速度比原来预计的快许多。另一个明显的优点是 Hadoop 已经开源，较容易(虽然远没有那么容易！)从雅虎法务部门获得许可在开源方面进行工作。因此，我们在 2006 年初设立了一个 200 个节点的研究集群，我们将 WebMap 的计划暂时搁置，转而为研究用户支持和发展 Hadoop。

## 9.2 Hadoop大记事

2004 年-- 最初的版本(现在称为 HDFS 和 MapReduce)由 Doug Cutting 和 Mike Cafarella 开始实施。

2005 年 12 月– Nutch 移植到新的框架，Hadoop 在 20 个节点上稳定运行。

2006 年 1 月– Doug Cutting 加入雅虎。

2006 年 2 月– Apache Hadoop 项目正式启动以支持 MapReduce 和 HDFS 的独立发展。

2006 年 2 月– 雅虎的网格计算团队采用 Hadoop。

2006 年 4 月– 标准排序(10 GB 每个节点)在 188 个节点上运行 47.9 个小时。

2006 年 5 月– 雅虎建立了一个 300 个节点的 Hadoop 研究集群。

2006 年 5 月– 标准排序在 500 个节点上运行 42 个小时(硬件配置比 4 月的更好)。

06年11月- 研究集群增加到600个节点。

06年12月- 标准排序在20个节点上运行1.8个小时，100个节点3.3小时，500个节点5.2小时，900个节点7.8个小时。

07年1月-- 研究集群到达900个节点。

07年4月-- 研究集群达到两个1000个节点的集群。

08年4月-- 赢得世界最快1TB数据排序在900个节点上用时209秒。

08年10月- 研究集群每天装载10TB的数据。

09年3月-- 17个集群总共24000台机器。

09年4月-- 赢得每分钟排序，59秒内排序500GB(在1400个节点上)和173分钟内排序100TB数据(在3400个节点上)。

在很多大型网站上都已经得到了应用，如亚马逊、Facebook、Yahoo、淘宝和百度等等

### 9.3 Hadoop的几个主要子项目

- 1) Hadoop Common: 在0.20及以前的版本中，包含HDFS、MapReduce和其他项目公共内容，从0.21开始HDFS和MapReduce被分离为独立的子项目，其余内容为Hadoop Common
- 2) HDFS: Hadoop 分佈式文件系統 (Distributed File System) — HDFS (Hadoop Distributed File System)
- 3) MapReduce: 并行计算框架，0.20前使用org.apache.hadoop.mapred旧接口，0.20版本开始引入org.apache.hadoop.mapreduce的新API
- 4) HBase: 类似Google BigTable的分布式NoSQL列数据库。(HBase和Avro已经于2010年5月成为顶级Apache项目[1])
- 5) Hive: 数据仓库工具，由Facebook贡献。
- 6) Zookeeper: 分布式锁设施，提供类似Google Chubby的功能，由Facebook贡献。
- 7) Avro: 新的数据序列化格式与传输工具，将逐步取代Hadoop原有的IPC机制。

### 9.4 官方集群搭建参考

参考：[http://hadoop.apache.org/common/docs/r0.19.2/cn/cluster\\_setup.html](http://hadoop.apache.org/common/docs/r0.19.2/cn/cluster_setup.html)

#### 9.4.1 配置文件

对Hadoop的配置通过conf/目录下的两个重要配置文件完成：

1. <src/core/core-default.xml>, <src/hdfs/hdfs-default.xml> and <src/mapred/mapred-default.xml> - 只读的默认配置。

2. *conf/core-site.xml*, *conf/hdfs-site.xml* and *conf/mapred-site.xml* - 集群特有的配置。

要了解更多关于这些配置文件如何影响 Hadoop 框架的细节, 请看[这里](#)。

此外, 通过设置 *conf/hadoop-env.sh* 中的变量为集群特有的值, 你可以对 *bin/* 目录下的 Hadoop 脚本进行控制。

#### 9.4.2 集群配置说明

要配置 Hadoop 集群, 你需要设置 Hadoop 守护进程的运行环境和 Hadoop 守护进程的运行参数。

Hadoop 守护进程指 NameNode/DataNode 和 JobTracker/TaskTracker。

##### 1. 配置Hadoop守护进程的运行环境

管理员可在 *conf/hadoop-env.sh* 脚本内对 Hadoop 守护进程的运行环境做特别指定。

至少, 你得设定 *JAVA\_HOME* 使之在每一远端节点上都被正确设置。

管理员可以通过配置选项 *HADOOP\_\*\_OPTS* 来分别配置各个守护进程。下表是可以配置的选项。

守护进程	配置选项
NameNode	<i>HADOOP_NAMENODE_OPTS</i>
DataNode	<i>HADOOP_DATANODE_OPTS</i>
SecondaryNamenode	<i>HADOOP_SECONDARYNAMENODE_OPTS</i>
JobTracker	<i>HADOOP_JOBTRACKER_OPTS</i>
TaskTracker	<i>HADOOP_TASKTRACKER_OPTS</i>

例如, 配置 **Namenode** 时, 为了使其能够并行回收垃圾 (**parallelGC**) , 要把下面的代码加入到 *hadoop-env.sh* :

```
export HADOOP_NAMENODE_OPTS="-XX:+UseParallelGC  
${HADOOP_NAMENODE_OPTS}"
```

其它可定制的常用参数还包括:

- *HADOOP\_LOG\_DIR* - 守护进程日志文件的存放目录。如果不存在会被自动创建。
- *HADOOP\_HEAPSIZE* - 最大可用的堆大小, 单位为 **MB**。比如, 1000MB。这个参数用于设置 **hadoop** 守护进程的堆大小。缺省大小是 1000MB。

## 2. 配置Hadoop守护进程的运行参数

这部分涉及 Hadoop 集群的重要参数，这些参数在下面配置文件中指定。

### 1) conf/core-site.xml:

参数	取值	备注
fs.default.name	URI of NameNode.	hdfs://hostname/

### 2) conf/hdfs-site.xml:

参数	取值	备注
dfs.name.dir	NameNode持久存储名字空间及事务日志的本地文件系统路径。	当这个值是一个逗号分割的目录列表时，nametable数据将会被复制到所有目录中做冗余备份。
dfs.data.dir	DataNode存放块数据的本地文件系统路径，逗号分割的列表。	当这个值是逗号分割的目录列表时，数据将被存储在所有目录下，通常分布在不同设备上。

### 3) conf/mapred-site.xml:



参数	取值	备注
mapred.job.tracker	JobTracker的主机（或者IP）和端口。	主机:端口。
mapred.system.dir	Map/Reduce框架存储系统文件的HDFS路径。比如 /hadoop/mapred/system/。	这个路径是默认文件系统（HDFS）下的路径，须从服务器和客户端上均可访问。
mapred.local.dir	本地文件系统下逗号分割的路径列表，Map/Reduce临时数据存放的地方。	多路径有助于利用磁盘i/o。
mapred.tasktracker.{map reduce}.tasks.maximum	某一TaskTracker上可运行的最大Map/Reduce任务数，这些任务将同时各自运行。	默认为2（2个map和2个reduce），可依据硬件情况更改。
dfs.hosts/dfs.hosts.exclude	许可/拒绝DataNode列表。	如有必要，用这个文件控制许可的datanode列表。
mapred.hosts/mapred.hosts.exclude	许可/拒绝TaskTracker列表。	如有必要，用这个文件控制许可的TaskTracker列表。
mapred.queue.names	Comma separated list of queues to which jobs can be submitted.	The MapReduce system always supports atleast one queue with the name as <i>default</i> . Hence, this parameter's value should always contain the string <i>default</i> . Some job schedulers supported in Hadoop, like the <a href="#">Capacity Scheduler</a> , support multiple queues. If such a scheduler is being used, the list of configured queue names must be specified here. Once queues are defined, users can submit jobs to a queue using the property name <i>mapred.job.queue.name</i> in the job configuration. There could be a separate configuration file for configuring properties of these queues that is managed by the scheduler. Refer to the documentation of the scheduler for information on the same.
mapred.acls.enabled	Boolean, specifying whether checks for queue ACLs and job ACLs are to be done for authorizing users for doing queue operations and job operations.	If <i>true</i> , queue ACLs are checked while submitting and administering jobs and job ACLs are checked for authorizing view and modification of jobs. Queue ACLs are specified using the configuration parameters of the form <i>mapred.queue.queue-name.acl-name</i> , defined below under <i>mapred-queue-acls.xml</i> . Job ACLs are described at <a href="#">Job Authorization</a>

## 4) conf/mapred-queue-acls.xml

Parameter	Value	Notes
mapred.queue. <i>queue-name</i> .acl-submit-job	List of users and groups that can submit jobs to the specified <i>queue-name</i> .	The list of users and groups are both comma separated list of names. The two lists are separated by a blank. Example: <i>user1, user2 group1, group2</i> . If you wish to define only a list of groups, provide a blank at the beginning of the value.
mapred.queue. <i>queue-name</i> .acl-administer-jobs	List of users and groups that can view job details, change the priority or kill jobs that have been submitted to the specified <i>queue-name</i> .	The list of users and groups are both comma separated list of names. The two lists are separated by a blank. Example: <i>user1, user2 group1, group2</i> . If you wish to define only a list of groups, provide a blank at the beginning of the value. Note that the owner of a job can always change the priority or kill his/her own job, irrespective of the ACLs.

通常，上述参数被标记为 [final](#) 以确保它们不被用户应用更改。

