第3章　大数据生态

学习导读

在当今数字化时代，企业和组织每天都会生成和处理海量的数据。这些数据不仅数量庞大，而且种类繁多，包括文本、图像、视频和传感器数据等。为了从这些数据中提取有价值的信息并做出更准确的决策，企业需要强大的数据处理能力和灵活的存储解决方案。Hadoop作为一种开源的大数据处理框架，正是在这样的背景下应运而生的。现在 Hadoop 不仅成为 Apache 的一个顶级项目，更成为分析大数据的领先平台。目前有很多企业开始提供基于 Hadoop 的商业软件。本章将会探讨 Hadoop 之所以这么流行的奥秘，并详细介绍 Hadoop 的组成部分及环境搭建。通过学习 Hadoop 的安装和配置，读者能够掌握大数据生态系统的重要组成部分，为处理和分析大规模数据集打下坚实的基础。

学习目标

1. 了解大数据处理框架 Hadoop 的特性及 Hadoop 的各个组件
2. 掌握部署 Hadoop 的方法
3. 认识分布式文件系统 HDFS、编程模型 MapReduce 和 ZooKeeper 集群

开篇案例

驱动数据智能的新引擎——阿里巴巴飞天大数据架构体系与 Hadoop 生态系统的深度融合

阿里巴巴飞天（Apsara）是由阿里云自主研发、服务全球的超大规模通用计算操作系统。它将遍布全球的百万级服务器连成一台超级计算机，以在线公共服务的方式为社会提供计算能力。飞天平台包括飞天内核和飞天开发服务两大部分，其中飞天内核负责统一管理数据中心内的通用服务器集群，调度集群的计算、存储资源，支撑分布式应用的部署和执行。

阿里巴巴作为全球领先的电商平台和云计算服务提供商，对大数据处理有着极高的需求。飞天大数据架构体系与 Hadoop 生态系统的深度融合，是阿里巴巴在数据处理领域的重要战略选择。这种融合不仅提升了数据处理能力，还降低了运维成本，加速了业务创新。

在架构融合过程中，阿里巴巴充分利用了飞天和 Hadoop 各自的技术优势。首先，在存储层面，飞天盘古分布式文件系统与 Hadoop HDFS 实现了无缝对接，共同为海量数据提供高可靠、高吞吐量的存储服务。其次，在计算层面，飞天伏羲任务调度系统与 Hadoop MapReduce 及 YARN 进行了深度整合，实现了计算资源的统一管理和调度。此外，阿里巴巴还基于 Hadoop 生态系统开发了多种定制化的数据处理工具和服务，如大数据计算服务 MaxCompute 等，进一步提升了数据处理效率和质量。

阿里巴巴飞天大数据架构体系与 Hadoop 生态系统的深度融合，被广泛应用于电商、金融、物流等多个领域。在电商领域，阿里巴巴通过实时分析用户行为数据、交易数据等海量信息，为商家提供精准的营销和推广服务；同时，通过智能预测和推荐算法优化用户体验和购物流

程。在金融领域，阿里巴巴利用大数据技术对信贷风险进行评估和预测，提高了金融服务的效率和安全性。

（资料来源：阿里巴巴飞天大数据架构体系与 Hadoop 生态系统的深度融合：构建高效、可扩展的数据处理平台）

思考：

大数据处理平台有哪些优势？这些优势是如何实现的？

3.1　认识 Hadoop

Hadoop 是一个存储和处理大规模数据的开源软件框架，可以在大量计算机组成的集群中对海量数据进行分布式存储计算。Hadoop 最初由 Doug Cutting 根据谷歌的 GFS 和 MapReduce 的思想，采用 Java 语言开发创建。

由于 Hadoop 采用了分布式存储方式及 Java 语言开发，所以可以部署在不同操作系统平台和通用的计算机集群中。Hadoop 中 HDFS 的数据管理能力，能够快速高效地读写文件，同时还采用了存储冗余数据的方式保证数据的安全性。Hadoop MapReduce 处理分布式任务时的简单方便及它的开源特性，使 Hadoop 在大数据领域被广泛使用。

用户可以轻松地在 Hadoop 上开发和运行处理海量数据的分布式应用程序。Hadoop 主要有以下几个优点。

（1）高扩展性。Hadoop 支持集群中的节点动态地增加，方便集群的扩展，可以根据需要扩展到上千节点。

（2）高容错性。Hadoop 能够自动保存数据的多个副本，当有集群节点数据丢失时，能够自动实现数据的备份。

（3）高效性。Hadoop 可以充分地利用网络、磁盘 I/O 资源和备份数据，实现文件的快速读写。

（4）高可靠性。正在执行的子任务由于某种原因终止的话，Hadoop 可以自动让其他节点继续执行该子任务，不影响整个任务的执行。

（5）低成本。Hadoop 集群不用部署在专用服务器中，只需部署在通用的计算机中即可；而且 Hadoop 生态系统绝大部分都是开源的，易于用户开发使用。

Hadoop 的两大核心是 HDFS 和 MapReduce：HDFS 为海量数据提供分布式存储，MapReduce 为海量数据的规则提取提供并行计算。

3.1.1　HDFS

HDFS 是一个分布式文件系统，采用了主从（Master-Slave）架构模型。一个 HDFS 集群是由一个 NameNode（Master）节点和若干个 DataNode（Slave）节点组成的。其中 NameNode 节点作为主服务器，管理整个文件系统的命名空间和客户端对文件的访问操作，如打开、关闭、重命名文件或目录等，同时它也负责文件块（在 HDFS 中，一个文件被分成若干个文件块，这些文件块存放在 DataNode 上）到具体 DataNode 的映射。DataNode 管理存储的数据，处理客户端的文件读写请求，并在 NameNode 的统一调度下进行文件块的创建、删除和复制工作。

为了避免数据的丢失，HDFS 默认采用了三个冗余备份，并将这些数据放在不同的 DataNode 上。

3.1.2　MapReduce

MapReduce 是一个简单易用的并行计算模型，它将运行于大规模集群上的复杂的并行计算过程高度地抽象为两个函数：Map 和 Reduce。MapReduce 应用程序能够以一种可靠容错的方式并行处理大数据，实现了在 Hadoop 集群上数据和任务的并行计算与处理。

MapReduce 应用通常会把输入文件切分为若干个文件块，每个文件块会启动一个相应的 Map 来并行处理，然后把 Map 处理后的结果输送给 Reduce 来进行汇总。整个 MapReduce 框架负责任务的调度和监控，以及重新执行已经失败的任务。

通常，MapReduce 框架和分布式文件系统是运行在一组相同的节点上的；也就是说，计算节点和存储节点在一起。这种做法可以在存储节点上运行 MapReduce 程序，减少集群内部的数据传输量，提高运行效率。

3.1.3　YARN

YARN 是 Hadoop 2.0 版本中的一个新增特性，主要负责集群的资源管理和调度。YARN 不仅支持 MapReduce 计算，还支持 Hive、HBase、Pig、Spark 等应用，这样就可以方便地从系统层面对集群进行统一的管理。

YARN 仍然采用 Master-Slave 结构，在整个资源管理框架中，ResourceManager 为 Master，NodeManager 为 Slave。其中 ResourceManager 负责整个系统的资源管理和分配，NodeManager 负责每个 DataNode 上的资源和任务管理。

3.2　部署 Hadoop

针对 Hadoop 学习，本书采用 VMware16 + Ubuntu18 64 位方案，其中主机内存应不小于 8G 且操作系统为 64 位，并且一个 Linux 主节点分配内存 4G，两个从 Linux 节点分配内存 1G。

3.2.1　单节点伪分布模式安装

1．创建 Hadoop 用户（在 root 用户下创建）

```
useradd -m hadoop -s /bin/bash
```

这条命令创建了可以登录的 Hadoop 用户，并使用/bin/bash 作为 shell。

接着使用如下命令设置密码：

```
passwd hadoop
```

按提示输入两次密码，并将其加入 sudo 组，如图 3-1 所示。

```
root@ubuntu1:/# useradd -m hadoop -s /bin/bash
root@ubuntu1:/# passwd hadoop
输入新的 UNIX 密码:
重新输入新的 UNIX 密码:
passwd: 已成功更新密码
root@ubuntu1:/# adduser hadoop sudo
正在添加用户"hadoop"到"sudo"组...
正在将用户“hadoop”加入到“sudo”组中
完成。
```

图 3-1　创建 Hadoop 用户

可为 Hadoop 用户增加管理员权限，方便部署。最后注销当前用户（见图 3-2），返回登录界面。在登录界面中选择刚创建的 Hadoop 用户进行登录，如图 3-3 所示。

图 3-2　注销用户

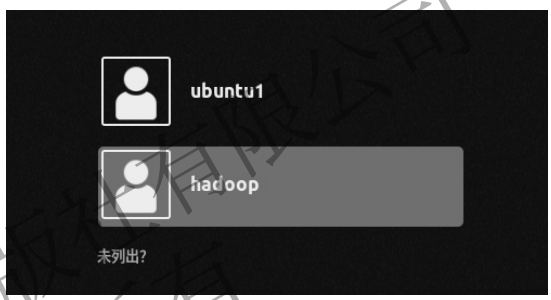

图 3-3　用新创建的 Hadoop 用户登录

2．安装 SSH、配置 SSH 无密码登录

集群、单节点模式都需要用到 SSH 登录（类似于远程登录）。Ubuntu 默认已安装了 SSH client，还需要安装 SSH server。

同时，为了方便下载，更改 Ubuntu 的默认安装源为阿里云。

（1）将 sources.list 中的源文件备份到 sources.list.bak 中。

```
sudo cp /etc/apt/sources.list /etc/apt/sources.list.bak
```

（2）命令行打开 sources.list 文件。

```
sudo vi /etc/apt/sources.list
```

（3）修改 sources.list 文件，将内容更换为阿里云源，在文件末尾添加如下内容。

```
deb http://mirrors.aliyun.com/ubuntu/ focal main restricted universe multiverse
deb-src http://mirrors.aliyun.com/ubuntu/ focal main restricted universe multiverse
deb http://mirrors.aliyun.com/ubuntu/ focal-security main restricted universe multiverse
deb-src http://mirrors.aliyun.com/ubuntu/ focal-security main restricted universe multiverse
deb http://mirrors.aliyun.com/ubuntu/ focal-updates main restricted universe multiverse
deb-src http://mirrors.aliyun.com/ubuntu/ focal-updates main restricted universe multiverse
deb http://mirrors.aliyun.com/ubuntu/ focal-proposed main restricted universe multiverse
deb-src http://mirrors.aliyun.com/ubuntu/ focal-proposed main restricted universe multiverse
deb http://mirrors.aliyun.com/ubuntu/ focal-backports main restricted universe multiverse
deb-src http://mirrors.aliyun.com/ubuntu/ focal-backports main restricted universe multiverse
```

（4）安装 SSH server。

```
sudo apt update      #更新源
sudo apt install openssh-server
```

此时会有 SSH 首次登录提示，输入 yes，然后按提示输入 Hadoop 用户密码，这样就登录本机了。

（5）配置免 SSH 密钥登录。

```
cd ~/.ssh/                              #若没有该目录，请先执行一次 ssh localhost
ssh-keygen -t rsa                       #会有提示，都按回车就可以
cat ./id_rsa.pub >> ./authorized_keys   #加入授权
```

3．安装 Java 环境

Hadoop 3.1.3 需要 JDK 版本在 1.8 及以上。下载 JDK 页面 http://www.oracle.com/technetwork/java/javase/downloads/index.html。为了方便 Scala 和 Spark 的学习，本书采用 JDK1.8 安装。

下载完毕后，即可将其传到虚拟机中，并将其解压到/hadoop 目录（该目录需要手动创建）。使用 vim 编辑器（先使用 sudo apt-get install vim 安装该软件）打开 Hadoop 这个用户的环境变量配置文件（~/.bashrc 文件），在文件末尾添加如下几行内容。

```
export JAVA_HOME=/hadoop/jdk1.8.0_162
export JRE_HOME=${JAVA_HOME}/jre
export CLASSPATH=.:${JAVA_HOME}/lib:${JRE_HOME}/lib
export PATH=${JAVA_HOME}/bin:$PATH
```

保存.bashrc 文件并退出 vim 编辑器。然后，继续执行如下命令让.bashrc 文件的配置立即生效。

```
source ~/.bashrc
```

这时，可以使用 java -version 命令查看是否安装成功，如果能够在屏幕上显示如图 3-4 所示信息，则说明安装成功。

```
hadoop@ubuntu1:/hadoop$ java -version
java version "1.8.0_162"
Java(TM) SE Runtime Environment (build 1.8.0_162-b12)
Java HotSpot(TM) 64-Bit Server VM (build 25.162-b12, mixed mode)
```

图 3-4　Java 环境测试

4．安装 Hadoop 3.1.3

Hadoop 安装文件，可以到 Hadoop 官网下载 hadoop-3.1.3.tar.gz（https://archive.apache.org/dist/hadoop/common/hadoop-3.1.3/）。将 Hadoop 上传至/soft/中，并将其解压到/hadoop 目录。

```
sudo tar -zxvf /soft/hadoop-3.1.3.tar.gz -C /hadoop
sudo chown-R hadoop /hadoop    #修改文件权限
```

Hadoop 解压后即可使用。输入如下命令来检查 Hadoop 是否可用，成功则会显示 Hadoop 版本信息，如图 3-5 所示。

```
cd /hadoop/hadoop-3.1.3
./bin/hadoop version
```

```
hadoop@ubuntu1:/hadoop/hadoop-3.1.3$ ./bin/hadoop version
Hadoop 3.1.3
Source code repository https://gitbox.apache.org/repos/asf/hadoop.git -r ba631c4
36b806728f8ec2f54ab1e289526c90579
Compiled by ztang on 2019-09-12T02:47Z
Compiled with protoc 2.5.0
From source with checksum ec785077c385118ac91aadde5ec9799
This command was run using /hadoop/hadoop-3.1.3/share/hadoop/common/hadoop-commo
n-3.1.3.jar
```

图 3-5　Hadoop 版本测试

也可以通过在 Hadoop 的安装目录下运行如下测试命令查看 Hadoop 自带的一个例子，./bin/hadoop jar ./share/hadoop/mapreduce/hadoop-mapreduce-examples-3.1.3.jar grep ./input ./output 'hadoop'，运行结果如图 3-6 和图 3-7 所示。

```
                Reduce input groups=1
                Reduce shuffle bytes=23
                Reduce input records=1
                Reduce output records=1
                Spilled Records=2
                Shuffled Maps =1
                Failed Shuffles=0
                Merged Map outputs=1
                GC time elapsed (ms)=0
                Total committed heap usage (bytes)=656408576
        Shuffle Errors
                BAD_ID=0
                CONNECTION=0
                IO_ERROR=0
                WRONG_LENGTH=0
                WRONG_MAP=0
                WRONG_REDUCE=0
        File Input Format Counters
                Bytes Read=121
        File Output Format Counters
                Bytes Written=22
```

图 3-6　Hadoop 测试案例 1

```
hadoop@ubuntu1:/hadoop/hadoop-3.1.3$ cat output/*
10      hadoop
```

图 3-7　Hadoop 测试案例 2

5．Hadoop 伪分布式配置

Hadoop 可以在单节点上以伪分布式的方式运行，Hadoop 进程以分离的 Java 进程来运行，节点既作为 NameNode 也作为 DataNode，同时，读取的是 HDFS 中的文件。Hadoop 的配置文件位于/hadoop/hadoop-3.1.3/etc/hadoop/中，伪分布式需要修改 2 个配置文件 core-site.xml 和 hdfs-site.xml。Hadoop 的配置文件是 xml 格式，每个配置以声明 property 的 name 和 value 的方式来实现。

（1）修改配置文件 core-site.xml。

修改为下面配置（注：/hadoop/hadoop-3.1.3 是 Hadoop 的安装目录）。

```
<configuration>
    <property>
        <name>hadoop.tmp.dir</name>
        <value>file:/hadoop/hadoop-3.1.3/tmp</value>
    </property>
    <property>
        <name>fs.defaultFS</name>
        <value>hdfs://localhost:9000</value>
```

```
    </property>
</configuration>
```

（2）修改配置文件 hdfs-site.xml。

```
<configuration>
    <property>
        <name>dfs.replication</name>
        <value>1</value>
    </property>
    <property>
        <name>dfs.namenode.name.dir</name>
        <value>file:/hadoop/hadoop-3.1.3/tmp/dfs/name</value>
    </property>
    <property>
        <name>dfs.datanode.data.dir</name>
        <value>file:/hadoop/hadoop-3.1.3/tmp/dfs/data</value>
    </property>
</configuration>
```

（3）配置完成后，执行 NameNode 的格式化。

```
cd /hadoop/hadoop-3.1.3
./bin/hdfs namenode -format
```

如果成功运行，会出现如图 3-8 所示内容。

```
2024-02-28 09:59:31,694 INFO namenode.FSDirectory: POSIX ACL inheritance enabled? true
2024-02-28 09:59:31,694 INFO namenode.FSDirectory: XAttrs enabled? true
2024-02-28 09:59:31,695 INFO namenode.NameNode: Caching file names occurring more than 10 times
2024-02-28 09:59:31,699 INFO snapshot.SnapshotManager: Loaded config captureOpenFiles: false, skipCaptureAccessTimeOnlyChange: f
alse, snapshotDiffAllowSnapRootDescendant: true, maxSnapshotLimit: 65536
2024-02-28 09:59:31,701 INFO snapshot.SnapshotManager: SkipList is disabled
2024-02-28 09:59:31,704 INFO util.GSet: Computing capacity for map cachedBlocks
2024-02-28 09:59:31,705 INFO util.GSet: VM type       = 64-bit
2024-02-28 09:59:31,705 INFO util.GSet: 0.25% max memory 868 MB = 2.2 MB
2024-02-28 09:59:31,705 INFO util.GSet: capacity      = 2^18 = 262144 entries
2024-02-28 09:59:31,716 INFO metrics.TopMetrics: NNTop conf: dfs.namenode.top.window.num.buckets = 10
2024-02-28 09:59:31,716 INFO metrics.TopMetrics: NNTop conf: dfs.namenode.top.num.users = 10
2024-02-28 09:59:31,716 INFO metrics.TopMetrics: NNTop conf: dfs.namenode.top.windows.minutes = 1,5,25
2024-02-28 09:59:31,719 INFO namenode.FSNamesystem: Retry cache on namenode is enabled
2024-02-28 09:59:31,720 INFO namenode.FSNamesystem: Retry cache will use 0.03 of total heap and retry cache entry expiry time is
 600000 millis
2024-02-28 09:59:31,722 INFO util.GSet: Computing capacity for map NameNodeRetryCache
2024-02-28 09:59:31,722 INFO util.GSet: VM type       = 64-bit
2024-02-28 09:59:31,722 INFO util.GSet: 0.029999999329447746% max memory 868 MB = 266.6 KB
2024-02-28 09:59:31,722 INFO util.GSet: capacity      = 2^15 = 32768 entries
2024-02-28 09:59:31,748 INFO namenode.FSImage: Allocated new BlockPoolId: BP-1760947488-127.0.1.1-1709085571738
2024-02-28 09:59:31,760 INFO common.Storage: Storage directory /hadoop/hadoop-3.1.3/tmp/dfs/name has been successfully formatted
.
2024-02-28 09:59:31,803 INFO namenode.FSImageFormatProtobuf: Saving image file /hadoop/hadoop-3.1.3/tmp/dfs/name/current/fsimage
.ckpt_0000000000000000000 using no compression
2024-02-28 09:59:31,932 INFO namenode.FSImageFormatProtobuf: Image file /hadoop/hadoop-3.1.3/tmp/dfs/name/current/fsimage.ckpt_0
0000000000000000000 of size 393 bytes saved in 0 seconds .
2024-02-28 09:59:31,946 INFO namenode.NNStorageRetentionManager: Going to retain 1 images with txid >= 0
2024-02-28 09:59:31,954 INFO namenode.FSImage: FSImageSaver clean checkpoint: txid = 0 when meet shutdown.
2024-02-28 09:59:31,955 INFO namenode.NameNode: SHUTDOWN_MSG:
/************************************************************
SHUTDOWN_MSG: Shutting down NameNode at ubuntu1/127.0.1.1
************************************************************/
hadoop@ubuntu1:/hadoop/hadoop-3.1.3$
```

图 3-8　NameNode 格式化

注意：在运行的时候可能会出现错误，如提示权限不够，则需要增加相应权限。

```
sudo chmod 777 tmp/     #tmp 是 core-site.xml 中配置的 hadoop.tmp.dir
```

接着启动 HDFS，命令如下所示。

```
cd /hadoop/hadoop-3.1.3
./sbin/start-dfs.sh
```

此时 Hadoop 系统会有如图 3-9 所示的提示，并可通过 jps 命令查看 Hadoop 相关进程，如图 3-10 所示。

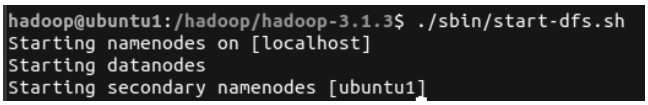

图 3-9　启动 HDFS

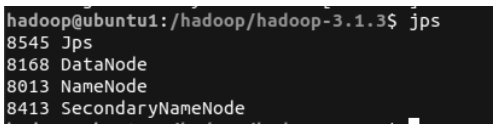

图 3-10　查看 Hadoop 相关进程

启动后，可以访问 Web 界面 http://localhost:9870 查看 NameNode 和 Datanode 信息，还可以在线查看 HDFS 中的文件，如图 3-11 所示。

图 3-11　通过访问 Web 界面查看 HDFS 中的文件

也可以在查看虚拟机 IP（hostname -I）后，通过 Windows 本机浏览器访问，如图 3-12 所示。

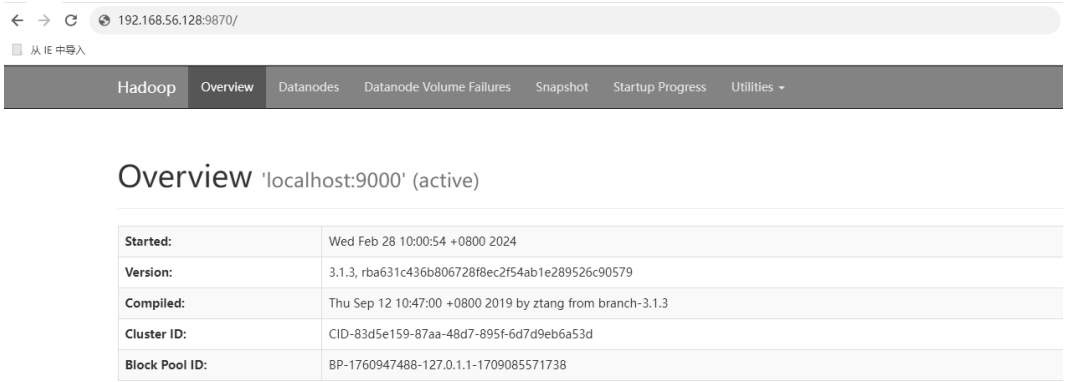

图 3-12　浏览器访问

3.2.2　多节点分布模式安装

1．配置静态 IP

Ubuntu 18.04 版本没有 ifconfig 等命令，可以安装 net-tools。

（1）安装 net-tools。

```
sudo apt install net-tools
```

（2）配置静态 IP，可以通过配置文件配置，也可以通过桌面菜单方式配置，如图 3-13 所示。

图 3-13　配置静态 IP

首先通过 ifconig 查看系统获得的动态 IP，然后将其配置在静态 IP 当中，以避免 IP 地址冲突。网关的设置需要查看 VMWare16 的相应配置，如图 3-14 和图 3-15 所示。

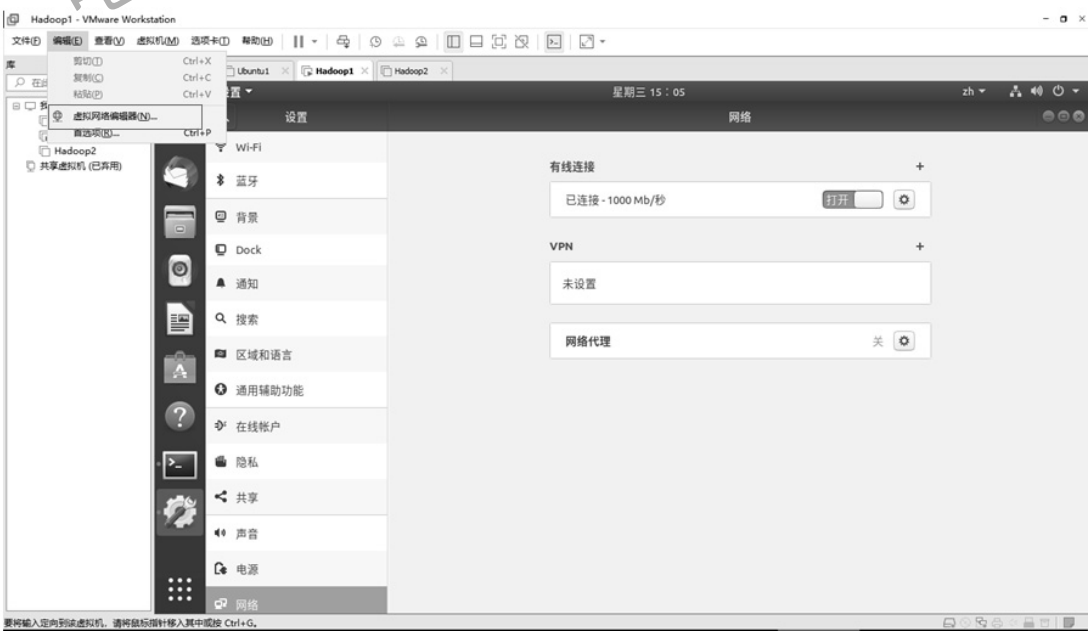

图 3-14　虚拟机网关查看 1

图 3-15　虚拟机网关查看 2

IP 地址设置完毕后，将用户注销或重启，即可生效。再次登录后，可通过 ping 命令查看网络配置是否正确。

2．配置/etc/hostname

该文件是配置主机名，每台虚拟机的主机名各不相同。IP 为 129 的设置为 master（主节点），IP 为 130 的设置为 slave1（从节点 1），IP 为 131 的设置为 slave2（从节点 2）。

3．配置/etc/hosts

该文件是配置 IP 和主机名的对应关系，三个虚拟机均采用如下配置（第一行配置不能丢掉，否则 Hadoop 启动的时候会报错）。

```
127.0.0.1    localhost
192.168.56.129    master
192.168.56.130    slave1
192.168.56.131    slave2
```

可以在任意主机执行命令"ping 主机名"，确定是否有正确回复。

4．SSH 免秘钥登录

Hadoop 中的 NameNode 和 DataNode 数据通信采用了 SSH 协议，需要配置 master 节点对各 slave 节点的免密钥登录。

在 slave1 和 slave2 中执行命令"sudo mkdir /~/.ssh"。

在 master 节点执行如下操作。

```
ssh-keygen -t dsa     #采用了 dsa 加密，也可以采用 rsa 加密
cat /~/.ssh/id_dsa.pub >> /root/.ssh/authorized_keys
scp /~/.ssh/authorized_keys    slave1:/root/.ssh/
scp /~/.ssh/authorized_keys    slave2:/root/.ssh/
```

配置完成后，在 master 节点可以通过“ssh slave1”无密钥登录 slave1，如图 3-16 所示。如果还需要密码，说明上面操作有误（容易犯的错误是误把/root/.ssh 认为是一个文件）。可以通过“ssh slave1”判断是否配置正确。

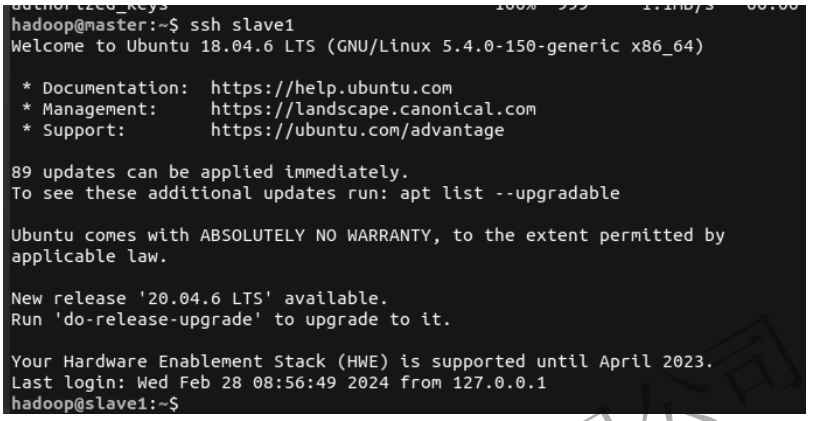

图 3-16　无密钥登录

5．安装 JDK

三台虚拟机安装 Java，并修改环境变量（vi ~/.bashrc），在文件末尾添加如下内容。

```
export JAVA_HOME=/hadoop/jdk1.8.0_162
export PATH=$PATH:$JAVA_HOME/bin:$JAVA_HOME/jre/bin
export CLASSPATH=$CLASSPATH:.:$JAVA_HOME/lib:$JAVA_HOME/jre/lib
```

在 Linux 中环境变量可以配置在~/.bashrc 中，也可以配置在/etc/profile 下，区别是后者对所有用户都生效，而前者只对当前用户生效。

6．安装 Hadoop

将 Hadoop 3.13 解压到/hadoop 目录并将 hadoop-3.1.3 重命名为 hadoop，可使用如下命令。

```
sudo mv hadoop-3.1.3 hadoop
```

进入/hadoop/hadoop/etc/hadoop/目录修改里面的配置文件。

（1）修改 hadoop-env.sh 和 yarn-env.sh。

在 Hadoop 中，以 env.sh 结尾的文件通常是配置所需的环境变量。

修改 Java 和 Hadoop 的环境变量如下。

```
export JAVA_HOME=/hadoop/jdk1.8.0_162
export HADOOP_HOME=/hadoop/hadoop
```

（2）配置 core-site.xml。

core-site.xml 是 Hadoop 的全局配置文件，主要设置一些核心参数信息，如 fs.defaultFS 设置集群的 HDFS 访问路径，hadoop.tmp.dir 指定 NameNode、DataNode 等存放数据的公共目录。

首先，创建如下目录。

```
sudo mkdir -p /hadoop/hadoop-3.1.3/tmp
sudo chmod 777 /hadoop/hadoop-3.1.3/tmp/
```

其次，在<configuration>里面添加如图 3-17 所示内容。

```
<configuration>
        <property>
                <name>hadoop.tmp.dir</name>
                <value>file:/hadoop/hadoop-3.1.3/tmp</value>
                <description>Abase for other temporary directories.</description>
        </property>
        <property>
                <name>fs.defaultFS</name>
                <value>hdfs://localhost:9000</value>
        </property>
</configuration>
```

图 3-17　配置 core-site.xml

（3）配置 hdfs-site.xml。

hdfs-site.xml 是 HDFS 的配置文件，在<configuration>里面添加如图 3-18 所示内容。

```
<configuration>
        <property>
                <name>dfs.replication</name>
                <value>1</value>
        </property>
        <property>
                <name>dfs.namenode.name.dir</name>
                <value>file:/hadoop/hadoop-3.1.3/tmp/dfs/name</value>
        </property>
        <property>
                <name>dfs.datanode.data.dir</name>
                <value>file:/hadoop/hadoop-3.1.3/tmp/dfs/data</value>
        </property>
</configuration>
```

图 3-18　配置 hdfs-site.xml

dfs.replication 用来设置副本存放个数，在实际生产中还会设置 dfs.namenode.name.dir 和 dfs.datanode.data.dir（NameNode 和 DataNode 的存放路径），同时将 core-site.xml 中的 hadoop.tmp.dir 配置去掉，避免冲突。

（4）配置 mapred-site.xml。

mapred-site.xml 是 MapReduce 的配置文件，由于 Hadoop 中不存在该文件，所以要先复制一个。

```
cp /hadoop/hadoop-3.1.3/etc/hadoop/mapred-site.xml.template　/hadoop/hadoop-3.1.3/etc/
hadoop/mapred-site.xml
```

然后将其修改为如图 3-19 所示内容，指定由 YARN 作为 MapReduce 的程序运行框架。如果没有配置这项，那么提交的程序只会在 Local 模式上运行，而不会在分布式模式上运行。

```
<configuration>
<!-- 指定MR运行在Yarn上 -->
<property>
                <name>mapreduce.framework.name</name>
                <value>yarn</value>
</property>

<!--hadoop3配置以下内容-->
<property>
  <name>mapreduce.map.env</name>
  <value>HADOOP_MAPRED_HOME=${HADOOP_HOME}</value>
</property>
<property>
  <name>mapreduce.reduce.env</name>
  <value>HADOOP_MAPRED_HOME=${HADOOP_HOME}</value>
</property>
<property>
        <name>mapreduce.application.classpath</name>
        <value>$HADOOP_HOME/share/hadoop/mapreduce/*:$HADOOP_HOME/share/hadoop/mapred
uce/lib/*</value>
</property>

</configuration>
```

图 3-19　配置 mapred-site.xml

（5）配置 yarn-site.xml。

yarn-site.xml 用来配置 YARN 的一些信息。yarn.nodemanager.aux-services 配置用户自定义服务，例如 MapReduce 的 shuffle。yarn.resourcemanager.hostname 是配置 ResourceManager 通信地址最基础、最简洁的方式。一旦设置了这个参数，YARN 会自动基于它来推导出其他关键的 RPC 和服务地址，如 yarn.resourcemanager.admin.address、yarn.resourcemanager.address、yarn.resourcemanager.scheduler.adress、yarn.resourcemanager.resource-tracker.address 和 yarn.resourcemanager.webapp.address 等。yarn.resourcemanager.address 用于设置客户端访问的地址，客户端通过该地址向 ResourceManager 提交应用程序、杀死应用程序等。yarn.resourcemanager.scheduler.adress 用于设置 ApplicationMaster 的访问地址，通过该地址向 ResourceManager 申请资源、释放资源等。yarn.resourcemanager.resource-tracker.address 用于设置 NodeManager 的访问地址，通过该地址向 ResourceManager 汇报心跳、领取任务等。yarn.resourcemanager.admin.address 用于设置管理员的访问地址，通过该地址向 ResourceManager 发送管理命令等。yarn.resourcemanager.webapp.address 用于设置对外 ResourceManager Web 访问地址，用户可通过该地址在浏览器中查看集群各类信息。yarn-site.xml 配置信息如图 3-20 所示。

```
<configuration>

<!-- Reducer获取数据的方式 -->
<property>
                <name>yarn.nodemanager.aux-services</name>
                <value>mapreduce_shuffle</value>
</property>

<!-- 指定YARN的ResourceManager的地址 -->
<property>
                <name>yarn.resourcemanager.hostname</name>
                <value>master</value>
</property>

</configuration>
```

图 3-20　yarn-site.xml 信息配置

（6）配置 workers。

通过配置 workers 文件 Master 节点知道集群中有几个 Salve 节点，然后通过主机名和 /etc/hosts 的信息就可以知道各 Salve 节点对应的 IP，并和其通信。该文件只需在 Master 节点配置即可，不需要在 Salve 节点配置。

```
touch /hadoop/hadoop-3.1.3/etc/hadoop/workers
```

将其内容修改为各 Salve 节点的主机名，如图 3-21 所示。

```
hadoop@master:/hadoop/hadoop-3.1.3/etc/hadoop$ cat workers
master
slave1
slave2
```

图 3-21　配置 workers

（7）将修改后的 Hadoop 分发到 Salve 节点。

```
scp -r /hadoop/hadoop-3.1.3 slave1:/hadoop
scp -r /hadoop/hadoop-3.1.3 slave2:/hadoop
```

（8）修改所有节点的环境变量。

在~/.bashrc 文件末尾添加 Hadoop 的环境变量，在后续 HBase、Hive 等的安装过程中仍需要执行类似操作。

```
export HADOOP_HOME=/hadoop/hadoop-2.6.5
export PATH=$PATH:$HADOOP_HOME/bin: $HADOOP_HOME/sbin
```

在配置环境变量的时候如果不小心配置错误，可能会造成无法正常启动到登录界面的情况。可以通过“Ctrl+Alt+F1”，切换到其他登录窗口，如 tty1，在里面输入管理员账号和密码，然后再对错误的环境配置修改即可。

（9）在 Master 节点格式化 NameNode。

```
hdfs namenode -format
```

NameNode 格式化后的显示内容如图 3-22 所示。

```
文件(F)  编辑(E)  查看(V)  搜索(S)  终端(T)  帮助(H)
2024-02-28 16:59:53,087 INFO namenode.FSDirectory: ACLs enabled? false
2024-02-28 16:59:53,087 INFO namenode.FSDirectory: POSIX ACL inheritance enabled? true
2024-02-28 16:59:53,087 INFO namenode.FSDirectory: XAttrs enabled? true
2024-02-28 16:59:53,087 INFO namenode.NameNode: Caching file names occurring more than 10 times
2024-02-28 16:59:53,094 INFO snapshot.SnapshotManager: Loaded config captureOpenFiles: false, skipCaptureAccessTimeOnlyChange: f
alse, snapshotDiffAllowSnapRootDescendant: true, maxSnapshotLimit: 65536
2024-02-28 16:59:53,095 INFO snapshot.SnapshotManager: SkipList is disabled
2024-02-28 16:59:53,102 INFO util.GSet: Computing capacity for map cachedBlocks
2024-02-28 16:59:53,103 INFO util.GSet: VM type       = 64-bit
2024-02-28 16:59:53,103 INFO util.GSet: 0.25% max memory 868 MB = 2.2 MB
2024-02-28 16:59:53,103 INFO util.GSet: capacity      = 2^18 = 262144 entries
2024-02-28 16:59:53,134 INFO metrics.TopMetrics: NNTop conf: dfs.namenode.top.window.num.buckets = 10
2024-02-28 16:59:53,135 INFO metrics.TopMetrics: NNTop conf: dfs.namenode.top.num.users = 10
2024-02-28 16:59:53,135 INFO metrics.TopMetrics: NNTop conf: dfs.namenode.top.windows.minutes = 1,5,25
2024-02-28 16:59:53,145 INFO namenode.FSNamesystem: Retry cache on namenode is enabled
2024-02-28 16:59:53,146 INFO namenode.FSNamesystem: Retry cache will use 0.03 of total heap and retry cache entry expiry time is
 600000 millis
2024-02-28 16:59:53,151 INFO util.GSet: Computing capacity for map NameNodeRetryCache
2024-02-28 16:59:53,151 INFO util.GSet: VM type       = 64-bit
2024-02-28 16:59:53,151 INFO util.GSet: 0.029999999329447746% max memory 868 MB = 266.6 KB
2024-02-28 16:59:53,151 INFO util.GSet: capacity      = 2^15 = 32768 entries
2024-02-28 16:59:53,414 INFO namenode.FSImage: Allocated new BlockPoolId: BP-1011906404-192.168.56.129-1709110793406
2024-02-28 16:59:53,460 INFO common.Storage: Storage directory /hadoop/hdfs/tmp/dfs/name has been successfully formatted.
2024-02-28 16:59:53,520 INFO namenode.FSImageFormatProtobuf: Saving image file /hadoop/hdfs/tmp/dfs/name/current/fsimage.ckpt_00
0000000000000000 using no compression
2024-02-28 16:59:53,642 INFO namenode.FSImageFormatProtobuf: Image file /hadoop/hdfs/tmp/dfs/name/current/fsimage.ckpt_000000000
0000000000 of size 393 bytes saved in 0 seconds .
2024-02-28 16:59:53,660 INFO namenode.NNStorageRetentionManager: Going to retain 1 images with txid >= 0
2024-02-28 16:59:53,684 INFO namenode.FSImage: FSImageSaver clean checkpoint: txid = 0 when meet shutdown.
2024-02-28 16:59:53,685 INFO namenode.NameNode: SHUTDOWN_MSG:
/************************************************************
SHUTDOWN_MSG: Shutting down NameNode at master/192.168.56.129
************************************************************/
hadoop@master:/hadoop/hadoop$
```

图 3-22　NameNode 格式化后的显示内容

（10）在 Maser 节点启动 start-all.sh。

start-all.sh 也可由 star-dfs.sh 和 start-yarn.sh 代替。执行命令后，提示输入 yes/no 时，输入 yes。在 Master 节点和 Salve 节点分别输入 jps 命令，如果出现如图 3-23～图 3-25 所示进程，表明 Hadoop 集群已正常搭建。

```
hadoop@master:/hadoop/hadoop$ jps
4563 DataNode
4420 NameNode
5079 ResourceManager
5225 NodeManager
4794 SecondaryNameNode
5658 Jps
```

图 3-23　Master jps 进程

```
hadoop@slave1:~$ jps
6851 Jps
6469 DataNode
6664 NodeManager
```

图 3-24　Slave1 jps 进程

```
hadoop@slave2:~$ jps
6369 Jps
6194 NodeManager
6038 DataNode
```

图 3-25　Slave2 jps 进程

3.3　HDFS

分布式文件系统是指文件系统管理的物理存储资源不一定直接连接在本地节点上，而是通过计算机网络与节点相连。分布式文件系统中一个文件的数据虽然保存在不同的物理节点，但是对用户而言和普通的文件系统没有区别。

HDFS 作为一个可扩展的分布式文件系统，隐藏了底层负载均衡、冗余复制等复杂细节，为上层程序提供一个统一的文件系统 API 接口，具有易于管理、高容错性、高可扩展性、高可靠性和高可用性等特性。

3.3.1　HDFS 体系结构

HDFS 主要由 3 个组件构成，分别是 Client、NameNode 和 DataNode。HDFS 是以主从模式运行的，其中 NameNode 运行在 Master 节点，DataNode 运行在 Slave 节点。HDFS 架构图如图 3-26 所示。

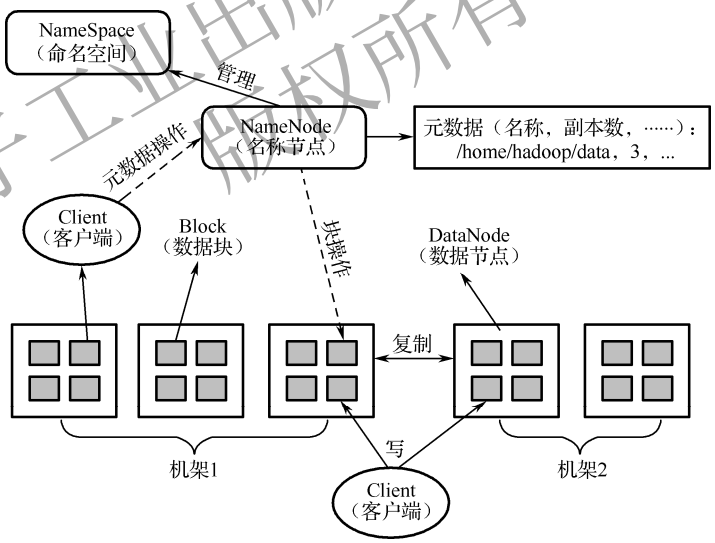

图 3-26　HDFS 架构图

（1）Client（客户端）。

用户可以通过类似 Linux shell 命令行方式或者 API 访问 HDFS，客户端是用户操作数据的入口。客户端通过与 NameNode 交互，获取文件位置信息；通过与 DataNode 交互，实现读取和写入数据。

（2）NameSpace（命名空间）。

HDFS 采用传统的层次型文件组织结构，其结构和 Linux 的文件组织结构很像，都有一个根节点“/”，所有文件都从该节点延伸而来，整个文件结构类似于树。

NameSpace 负责维护 HDFS 文件系统树及树内所有的文件和目录，并将这些信息以两种形式永久保存在本地磁盘上，即命名空间镜像（fsimage）和编辑日志（edits）。NameSpace 记录着每个数据块（包括备份数据块）位于哪个 DataNode 上，而且这些信息都保存在内存中，因此每次集群启动时会通过 DataNode 汇报重新生成。

NameSpace 由 NameNode 负责管理，所有对命名空间的改动（包括创建、删除、重命名和改变属性等，但是不包括打开、读取、写入数据）都会被 HDFS 记录下来并保存在 edits 中。

（3）NameNode（名称节点）。

Hadoop 1.x 版本只有一个 NameNode，Hadoop 2.x 版本可以设置多个 NameNode，但整个 Hadoop 集群只有一个 NameNode 处于 Active 状态。NameNode 用来管理整个文件系统的命名空间，管理数据块映射信息，处理来自 Client 的文件访问请求（打开、关闭、重命名）和设置文件副本。

由于 NameNode 把文件系统的元数据信息保存在内存中，因此文件系统所能容纳的文件数目是由 NameNode 所在机器的内存大小决定的，这也决定了 Hadoop 集群的 DataNode 节点不能无限扩展。一般来说，一个文件、文件夹和数据块需要占据大约 150 字节的空间。假设集群有 100 万个文件，每一个文件占据一个数据块，就至少需要 300MB 内存；如果 NameNode 节点内存为 64GB，那么集群大概能存储两亿个文件。

（4）DataNode（数据节点）。

DataNode 负责存储实际的数据，执行用户对数据的读写请求，响应 NameNode 对数据块的创建、删除指令，汇报数据块存储信息给 NameNode，以心跳包的形式向 NameSpace 发送存储的数据块列表。

DataNode 将 HDFS 数据以数据块的形式存储在本地文件系统的一个单独的文件中。实际上，DataNode 不会在同一个目录下创建所有的文件，它采用试探的方法来确定每个目录的最佳文件数目，并且在适当的时候创建子目录。

（5）Secondary NameNode（第二名称节点）。

Secondary NameNode 帮助 NameNode 收集文件系统运行的状态信息，周期性（可以配置）保存 NameNode 的元数据（包括 fsimage 和 edits）。Secondary NameNode 是在文件系统中设置一个检查点来帮助 NameNode 更好地工作，但在 Hadoop 2.x 中采用了 HA 来取代它。

（6）Block（数据块）。

为了提高系统的读写效率，HDFS 采用了数据块作为数据读写单位。一个文件可以分成若干块，每个块可以保存在不同的 DataNode 上，这样 HDFS 就可以存储比本地磁盘容量大得多的文件。数据块对用户是透明的，用户看到的仍是一个完整文件。

与一般文件系统中大小为几 KB 的数据块不同，HDFS 数据块的默认大小是 128MB（在 Hadoop 1.0 中是 64MB），根据实际需要，还可以设置成 256MB 甚至更大。HDFS 数据块之所以设置这么大，目的是减少寻址开销。由于 HDFS 存储大文件较多，如果数据块设置太小，就需要读取大量数据块，而数据块又是分布式存储的，会造成硬盘寻址时间比传输时间长。为了减少寻址时间提高系统吞吐量，数据块不能设置得太小。数据表明，如果寻址时间在 10 毫秒左右，传输速率是 100MB/s，为了使寻址时间为传输时间的 1%，则数据块需要 100MB 左右。数据块大的另一个好处就是一个大文件会由较少的数据块组成，减少 NameNode 的内存消耗，增加集群存储文件的数目。

需要注意的是数据块只是 HDFS 中文件分割的单位，并不是数据存储单位。例如，一个文件为 200MB，它会被分成两个数据块，一个数据块为 128MB，另一个数据块为 72MB 而不是 128MB。

3.3.2　HDFS 存储原理

1．HDFS 写数据的流程

客户端要向 HDFS 写数据，首先要跟 NameNode 通信以确认可以写文件并获得接收文件数据块的 DataNode，然后客户端按顺序将文件数据块逐个传递给相应的 DataNode，并由接收到数据块的 DataNode 负责向其他 DataNode 复制数据块的副本。详细步骤如下。

（1）客户端向 NameNode 发送上传文件请求。NameNode 对要上传的目录和文件进行检查，判断是否可以上传，并向客户端返回检查结果。

（2）客户端得到上传文件的允许后读取客户端配置，如果没有指定配置则会读取默认配置（如副本数和块大小分别默认为 3 个和 128MB，副本是由客户端决定的），向 NameNode 请求上传一个数据块。

（3）NameNode 会根据客户端的配置来查询 DataNode 信息，如果使用默认配置，那么最终结果会返回同一个机架的两个 DataNode 和另一个机架的 DataNode。这被称为机架感知策略。

HDFS 采用机架感知策略来提升数据的可靠性、可用性和网络带宽的利用率。大型 HDFS 实例一般运行在跨越多个机架的计算机组成的集群上，不同机架上的两台机器之间的通信需要经过交换机。在大多数情况下，同一个机架内的两台机器间的带宽会比不同机架的两台机器间的带宽大。通过一个机架感知的过程，NameNode 可以确定每个 DataNode 所属的机架 ID。一个简单但没有优化的策略就是将副本存放在不同的机架上，这样可以有效防止当整个机架失效时数据的丢失，并且允许读数据的时候充分利用多个机架的带宽。这种策略设置可以将副本均匀分布在集群中，有利于组件失效情况下的负载均衡。但是，因为这种策略的一个写操作需要传输数据块到多个机架，所以增加了写的代价。在大多数情况下，副本系数是 3。HDFS 的存放策略是将一个副本存放在本地机架的节点上，一个副本放在同一机架的另一个节点上，最后一个副本放在不同机架的节点上。这种策略减少了机架间的数据传输，所以提高了写操作的效率。机架的错误远远比节点的错误少，所以这个策略不会影响数据的可靠性和可用性。与此同时，因为数据块只放在两个（不是三个）不同的机架上，所以此策略减少了读取数据时需要的网络传输总带宽。在这种策略下，副本并不是均匀分布在不同的机架上的：三分之一的副本在一个节点上，三分之二的副本在一个机架上，其他副本均匀分布在剩下的机架中。这一策略在不损害数据可靠性和读取性能的情况下改进了写的性能。

（4）客户端在开始传输数据块之前会把数据缓存在本地，当缓存大小超过了一个数据块大小时，客户端就会从 NameNode 获取要上传的 DataNode 列表，之后会在客户端和第一个 DataNode 间建立连接，开始流式传输数据，这个 DataNode 会一小部分（4K）地接收数据然后写入本地仓库，同时会把这些数据传输到第二个 DataNode，第二个 DataNode 也同样会一小部分地接收数据并写入本地仓库，同时传输给第三个 DataNode，依此类推。这样逐级调用和返回之后，待这个数据块传输完成客户端后告知 NameNode 数据块传输完成，这时候 NameNode 才会更新元数据信息、记录操作日志。

（5）第一个数据块传输完成后会使用同样的方式传输下面的数据块直到整个文件上传完成。

① 请求和应答时使用 RPC 的方式，客户端通过 Client Protocol 与 NameNode 通信，NameNode 和 DataNode 之间使用 DataNode Protocol 交互。在设计上，NameNode 不是主动发起 RPC，而是响应来自客户端或 DataNode 的 RPC 请求。客户端和 DataNode 之间使用 Socket 进行数据传输，客户端和 NameNode 之间的交互采用 NIO 封装的 RPC。

② HDFS 有自己的序列化协议。

③ 在数据块传输成功后但客户端没有告诉 NameNode 之前，如果 NameNode 宕机，那么这个数据块就会丢失。

④ 在流式复制时，逐级传输和响应采用队列响应来等待传输结果，队列响应完成后返回给客户端。

⑤ 在流式复制时，如果有一台或两台（不是全部）没有复制成功，不影响最后结果，只不过 DataNode 会定期向 NameNode 汇报自身信息。如果发现异常，NameNode 会指挥 DataNode 删除残余数据和完善副本。如果副本数量少于某个最小值，就会进入安全模式。

安全模式：NameNode 启动后会进入一个称为安全模式的特殊状态，处于安全模式的 NameNode 是不会进行数据块的复制的。NameNode 从所有的 DataNode 接收心跳信号和块状态报告。块状态报告包括了某个 DataNode 所有的数据块列表，每个数据块都有一个指定的最小副本数。当 NameNode 检测确认某个数据块的副本数目达到这个最小值时，该数据块就会被认为是安全的；在一定百分比（这个参数可配置）的数据块被 NameNode 检测确认是安全的之后（加上一个额外的 30 秒等待时间），NameNode 将退出安全模式状态。接下来 NameNode 会确定还有哪些数据块的副本没有达到指定数目，并将这些数据块复制到其他 DataNode 上。

2. HDFS 读数据的流程

客户端将要读取的文件路径发送给 NameNode，NameNode 获取文件的元信息（主要是 Block 的存放位置信息）后返回给客户端。客户端根据返回的信息找到相应的 DataNode 逐个获取文件的 Block 并在客户端本地进行数据追加合并从而获得整个文件。详细步骤如下。

（1）客户端向 NameNode 发起 RPC 调用，请求读取文件数据。

（2）NameNode 检查文件是否存在，如果存在则获取文件的元信息（Block ID 及对应的 DataNode 列表）。

（3）客户端收到元信息后选取一个网络距离最近的 DataNode，依次请求读取每个数据块。客户端首先要校检文件是否损坏，如果损坏，客户端会选取另外的 DataNode 请求。

（4）DataNode 与客户端建立 Socket 连接，传输对应的数据块。客户端收到数据缓存到本地，之后写入文件。

（5）依次传输剩下的数据块，直到整个文件合并完成。

从某个 DataNode 获取的数据块有可能是损坏的，损坏可能是由 DataNode 的存储设备错误、网络错误或者软件 Bug 造成的。HDFS 客户端软件实现了对 HDFS 文件内容的校验和检查。客户端在创建一个新的 HDFS 文件时，会计算这个文件每个数据块的校验和，并将校验和作为一个单独的隐藏文件保存在同一个 HDFS 名字空间下。当客户端获取文件内容后，它会检验从 DataNode 获取的数据的校验和与文件中的校验和是否匹配。如果不匹配，客户端可

以选择从其他 DataNode 获取该数据块的副本。

3．HDFS 删除数据的流程

HDFS 删除数据的流程相对简单，详细步骤如下。

（1）客户端向 NameNode 发起 RPC 调用，请求删除文件。NameNode 检查其合法性。

（2）NameNode 查询文件的相关元信息，向存储文件数据块的 DataNode 发出删除请求。

（3）DataNode 删除相关数据块，返回结果给 NameNode。

（4）NameNode 返回结果给客户端。

当用户或应用程序删除某个文件时，这个文件并没有立刻从 HDFS 中删除。实际上，HDFS 会将这个文件重命名并转移到/trash 目录。只要文件还在/trash 目录中，该文件就可以被迅速地恢复。/trash 目录仅仅保存被删除文件的最后副本。文件在/trash 中保存的时间是可配置的，目前的默认策略是删除/trash 中保留时间超过 6 小时的文件；当超过这个时间，NameNode 就会将该文件从名字空间中删除，删除文件会使得与该文件相关的数据块被释放。

当一个文件的副本系数被减小后，NameNode 会将过剩的副本删除。下次心跳检测时会将该信息传递给 DataNode。DataNode 随机移除相应的数据块，会增加集群中的空闲空间。同样，调用的 setReplication API 结束和集群中空闲空间增加会有一定的延迟。

3.3.3　HDFS 实战

HDFS 的访问方式有两种，一种是 HDFS shell 方式，一种是 Java API 方式。HDFS shell 命令应使用 hadoop fs 或 hdfs dfs（其中官网建议使用 hdfs dfs 方式访问）。所有的 HDFS shell 命令都使用 URI 路径作为参数。URI 格式是 scheme://path：对于 HDFS 文件系统，scheme 是 hdfs；对于本地文件系统，scheme 是 file。其中 scheme 是可选的，如果未加指定，默认是 HDFS 文件系统。一个 HDFS 文件或目录，如/parent/path，可以表示成 hdfs://parent/path，或者更简单的/parent/path。

HDFS shell 的用法和 Linux shell 的用法很相似，可以通过 help 命令查看详细帮助，如查看 ls 命令的帮助 hdfs dfs -help ls。下面将常用命令做一介绍。

1．打印文件列表 ls

```
hdfs dfs -ls hdfs:/
```

明确说明访问 HDFS 系统的路径，只访问根目录下的目录和文件，不会递归访问里面的子目录。

```
hdfs dfs -ls /
```

默认情况下访问 HDFS 系统的根目录。

```
hdfs dfs -ls /package/test/
```

访问指定 HDFS 系统下某个目录。

```
hdfs dfs -ls -R /
```

递归访问 HDFS 系统下所有目录和文件，包含子目录。

2．上传文件 put

```
hdfs fs -put file:/root/test.txt /
```

上传本地 test.txt 文件到 HDFS 根目录，HDFS 根目录需无同名文件，否则报“File exists”。

```
hdfs dfs -put /root/test.txt /
```

该命令同上。

```
hdfs dfs -put /root/test.txt /test2.txt
```

上传并重命名为 test2.txt。

```
hdfs dfs -put test1.txt test2.txt hdfs:/
```

一次上传多个文件到 HDFS 路径，采用的是相对路径。

```
hdfs dfs -put /root/folder /
```

上传文件夹。

```
hdfs dfs -put /root/ folder /new folder
```

上传并重命名文件夹。

```
hdfs dfs -put -f /root/test.txt /
```

上传本地 test.txt 文件到 HDFS 根目录，如果有同名文件会覆盖掉。

3．下载文件 get

```
hdfs dfs -get hdfs:/test.txt file:/root/
```

复制文件到本地目录。

```
hdfs dfs -get /test.txt /root/test.txt
```

复制文件到本地目录并重命名。

4．复制文件 cp

```
hdfs dfs -cp file:/test.txt hdfs:/test2.txt
```

从本地复制到 HDFS。

```
hdfs dfs -cp hdfs:/test.txt hdfs:/test2.txt
```

从 HDFS 复制到 HDFS。

```
hdfs dfs -cp /test.txt /test2.txt
```

从 HDFS 复制到 HDFS。

5．删除文件 rm

```
hdfs dfs -rm /test2.txt
```

删除 HDFS 中的 test2.txt 文件。

```
hdfs dfs -rm -R /root/
```

删除 HDFS 中 root 目录下的所有文件。

6．创建文件 touchz

```
hdfs dfs - touchz /newfile.txt
```

在 HDFS 中创建新文件 newfile.txt。

7．创建文件夹 mkdir

```
hdfs dfs -mkdir /newdir
```

在 HDFS 中创建文件夹 newdir。

```
hdfs dfs -mkdir -p / newdir/newdir1/newdir2
```

在 HDFS 中创建多级文件夹。

8．移动文件 mv

```
hdfs dfs -mv /test1.txt /root
```

将 HDFS 中的 test1.txt 移动到 root 目录下。

```
hdfs dfs -mv /test1.txt /test4.txt
```

达到类似重命名的效果。

9．写入文件 appendToFile

```
hadoop fs -appendToFile file:/test.txt hdfs:/newfile.txt
```

将本地文件内容追加到 HDFS 文件。

10．修改文件权限 chmod

权限模式同 Linux shell 命令中的模式。

```
hdfs dfs -chmod -R 775 /tmp
```

将 tmp 目录下所有文件的权限更改为 775。

3.4　MapReduce

MapReduce 提供了一个高度抽象化的编程模型，它的思想就是“分而治之”。用户只需要使用 Map 和 Reduce 函数就可以完成分布式编程。它极大地方便了编程人员在不会分布式并行编程的情况下，将程序运行在分布式系统上。

（1）Map 函数负责“分”，即把复杂的任务分解为若干个简单、耦合度低且可以并行计算的子任务来处理。分解后，每个子任务的数据和计算规模都不太大，本着“数据不动，计算动”的原则，这些子任务通常会被分配到数据所在的节点上运行。

（2）Reduce 函数负责对 Map 阶段的结果进行汇总，并将最终结果输送给 HDFS。

3.4.1　MapReduce 逻辑结构

MapReduce 是一种并行计算编程模型，可以使用多种语言开发，如 Java、Ruby、Python、C++等。MapReduce 通过抽象模型和计算框架把需要做什么与具体怎么做分开了，为程序员

提供了一个抽象和高层的编程接口和框架。程序员仅需要关心其应用层的具体计算问题，仅需编写少量的处理应用本身计算问题的程序代码即可，而与具体如何完成这个并行计算任务相关的诸多系统层细节都被隐藏起来，交给计算框架去处理：从分布代码的执行到数千个节点集群的自动调度使用。

（1）map()。map()函数以 key/value 键值对作为输入，以 key/value 作为输出。MapReduce 框架会自动将输出数据按照 key 的字典升序排序，通常会启动多个 Map 任务执行，Map 任务执行结束后将结果写入本地硬盘，而非 HDFS 文件系统。

（2）reduce()。reduce()函数以 map()函数的输出结果作为输入，合并相同 key 的 value 值后，生成另外一系列 key/value 作为输出并将其写入 HDFS 中。MapReduce 框架会默认启动一个 Reduce，也可以通过配置设置 0 个或多个 Reduce 任务。

MapReduce 提供一个统一的计算框架，可完成计算任务的划分和调度，数据的分布存储和划分，处理数据与计算任务的同步，结果数据的收集整理，系统通信，负载平衡，计算性能优化处理，处理系统节点出错检测和失效恢复。

（1）任务调度：提交的一个计算作业将被划分为很多个计算任务，任务调度功能主要负责为这些划分后的计算任务分配和调度计算节点（Map 节点或 Reduce 节点）；同时负责监控这些节点的执行状态，并负责 Map 节点执行的同步控制；也负责进行一些计算性能优化处理，如对最慢的计算任务采用多备份执行、选最快完成者作为结果。

（2）数据/代码互定位：为了减少数据通信，一个基本原则是本地化数据处理，即一个计算节点尽可能处理其本地磁盘上所分布存储的数据，这实现了代码向数据的迁移；当无法进行这种本地化数据处理时，再寻找其他可用节点并将数据经网络传送给该节点（数据向代码迁移），但将尽可能从数据所在的本地机架上寻找可用节点以减少通信延迟。

（3）出错处理：在以低端商用服务器构成的大规模 MapReduce 计算集群中，节点硬件（主机、磁盘、内存等）出错和软件有 Bug 是常态，因此，MapReduce 需要能检测并隔离出错节点，并调度分配新的节点接管出错节点的计算任务。

（4）分布式数据存储与文件管理：海量数据处理需要一个良好的分布数据存储和文件管理系统支撑，该文件系统能够把海量数据分布存储在各个节点的本地磁盘上，但保持整个数据在逻辑上成为一个完整的数据文件；为了提供数据存储容错机制，该文件系统还要具有数据块的多备份存储管理能力。

（5）Combiner 和 Partitioner：为了减少数据通信开销，中间结果数据进入 Reduce 节点前需要进行合并（Combine）处理，把具有同样主键的数据合并到一起避免重复传送；一个 Reduce 节点所处理的数据可能会来自多个 Map 节点，因此，Map 节点输出的中间结果需使用一定的策略进行适当的划分（Partitioner）处理，以保证相关数据发送到同一个 Reduce 节点。

3.4.2 MapReduce 实战

MapReduce 按照执行的先后顺序，大致分为输入分片（input split）、Map、Partitioner、Sort、Combiner 和 Reduce 阶段，其中 Partitioner、Sort 和 Combiner 通常又被称为 Shuffle。MapReduce 工作流程如图 3-27 所示。

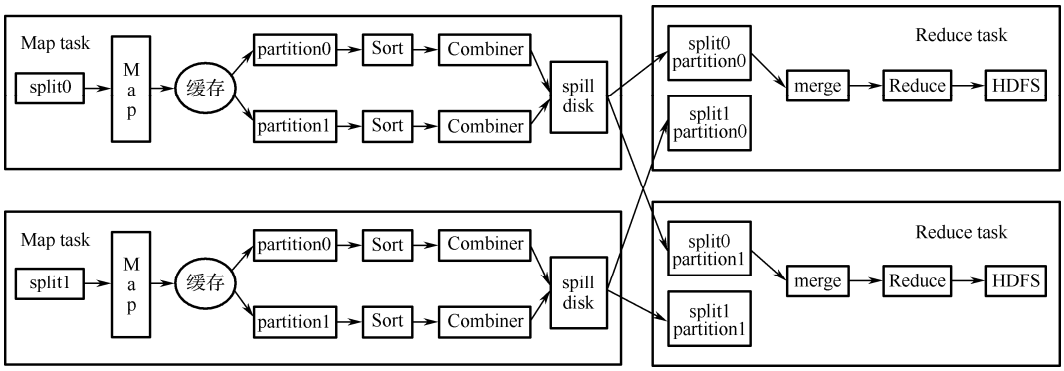

图 3-27　MapReduce 工作流程

（1）输入分片（input split）：在进行 Map 计算之前，MapReduce 会根据输入文件计算输入分片，每个输入分片会启动一个 Map 任务，也就说 Map 任务个数是由输入分片决定的。Map 个数的计算公式为 splitsize=max[minimumsize,min(maximumsize,blocksize)]。

blocksize 为文件块大小，minimumsize 和 maximumsize 可以在配置文件中设置，没有设置的话 splitsize 的大小默认等于 blocksize，即 128MB。假设有三个输入文件，大小分别是 64MB、129MB 和 256MB，那么 MapReduce 会把 64MB 文件划分为一个输入分片，129MB 文件则是两个输入分片（128MB 和 1MB），256MB 文件也是两个输入分片（都是 128MB），一共有五个输入分片，分别启动五个 Map 任务，但是每个 Map 任务运算的数据量不一样。

（2）Map 阶段：通过自定义 map()函数读入数据分片，输出的 key/value 键值对放到环形内存缓冲区，这个缓冲区专门用来存放 Map 的输出结果，其默认大小是 100MB，并且在配置文件里为这个缓冲区设定一个阀值（默认是 0.80，缓冲区大小和阀值都是可配置的）。当缓冲区的内存达到阀值时，内容会以一个临时文件的方式被存放到磁盘中，这个过程叫 Spill。另外的 20%内存可以继续写入要写入磁盘的数据，写入磁盘和写入内存操作是互不干扰的。如果缓存区满了，Map 就会阻塞写入内存的操作，让写入磁盘操作完成后再继续执行写入内存操作。

（3）Partitioner 阶段：对 Map 产生的中间结果进行分片，以便将同一分组的数据交给同一个 Reduce 处理。一个 Partitioner 对应一个 Reduce 作业，也就是说有几个 Reduce 就会有几个 Partitioner。Partitioner 用来决定 Map 产生的 key 由哪个 Reduce 来处理，默认处理采用对 key 进行 Hash 运算后再对 Reduce 数取模。这种方式只是为了平均 Reduce 的处理能力，但实际上可能会出现“数据倾斜”的情况，如某个 key 有大量相同数据，如有 20 万条数据，而其他所有 key 加起来才有不到 200 条数据，要是由两个 Reduce 处理的话，会造成两个任务处理量严重不均衡，为了避免这种情况就需要用户自定义 Partitioner。

（4）Sort 阶段：根据 key 按照字典升序排序。

（5）Combiner 阶段：Combiner 阶段可以由用户选择是否执行，这是一个本地化的 Reduce 操作。它是在 Map 生成中间文件前做一个合并重复 key 值的操作，这样可以减少磁盘存储和网络传输量。不是所有操作都要使用 Combiner，使用原则是 Combiner 的输出不能影响 Reduce 的最终输入。例如，如果计算只是求总数，最大值或最小值可以使用 Combiner，但是求平均值就不能使用 Combiner，否则 Reduce 会求出比真实值大的结果。

（6）Reduce 阶段：需要用户自定义实现，可以没有，如只是对原始数据做一些格式转换。在 Reduce 执行之前会将各个 Map 中 Partitioner 相同的数据合并排序，将执行的结果保存在 HDFS 上。

下面通过一个单词计数案例来理解各个过程。

（1）读取文件，文件划片后默认会将文件按行分割形成<key,value>，其中 key 的值是行偏移量，如图 3-28 所示。

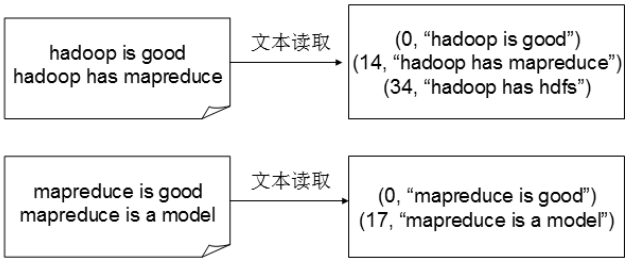

图 3-28　读取文件

（2）执行自定义 Map，如将（1）中的 value 分割成一个个单词，每个单词做新 key/value 键值对的 key，1 作为 value，如图 3-29 所示。

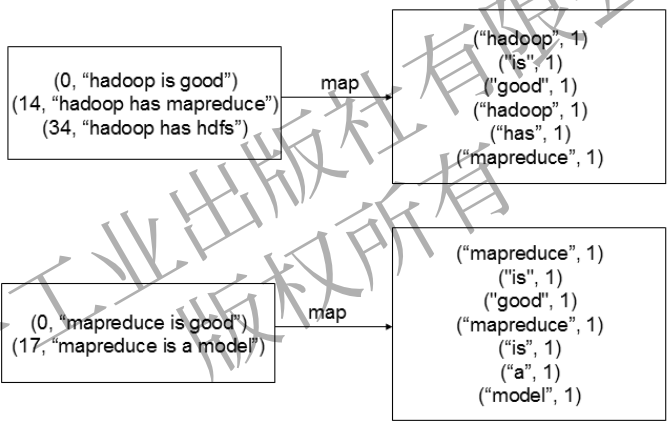

图 3-29　执行自定义 Map

（3）得到 Map 方法输出的<key,value>对后，Map 会执行 shuffle 过程，即先排序后合并，在此默认 Reduce 的执行数量为 1，得到 Map 的最终输出结果，如图 3-30 所示。

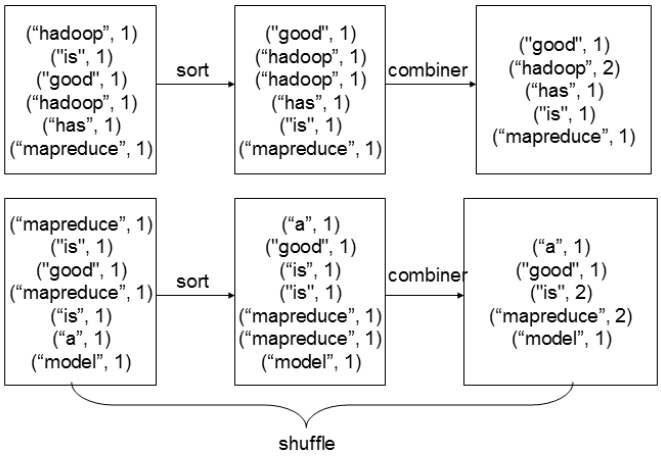

图 3-30　执行 shuffle 过程

（4）Reduce 先对从 Map 接收的数据进行合并排序，再交由用户自定义的 Reduce 方法进行处理，得到新的<key,value>对，并将其输出到 HDFS 上，如图 3-31 所示。

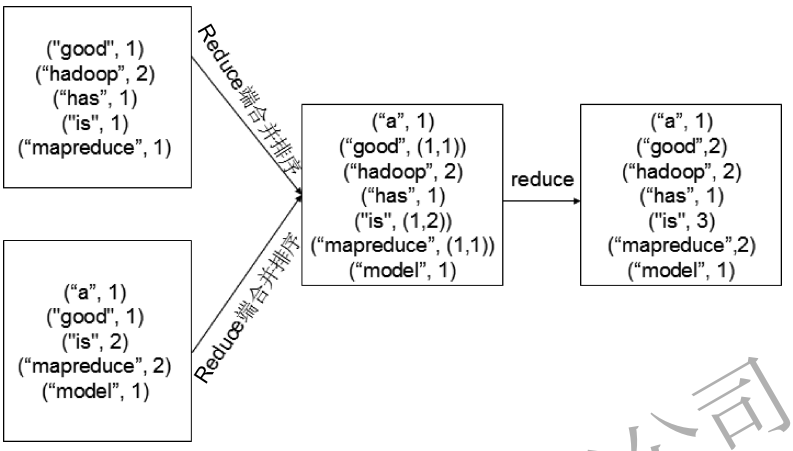

图 3-31　Reduce 操作

Hadoop 1.x MapReduce 框架由 JobTracker 和 TaskTracker 共同组成。其中 JobTracker 是 Master 节点上的主控进程，负责调度一个作业的所有任务并监控它们的执行。TaskTracker 是 Slave 节点上的从进程，负责执行由 JobTracker 指派的任务。具体工作流程如下。

（1）用户提交作业后，JobTracker 首先检查作业的输出目录是否存在，如果存在返回异常，如果不存在作业将被提交。

（2）计算作业的输入划分并实例化作业为其分配 ID，将程序 Jar 包、作业配置文件、分片信息等上传到 HDFS。这些文件都存放在 JobTracker 专门为该作业创建的文件夹中，文件夹名为该作业的 Job ID。

（3）作业被 JobTracker 放入作业队列，作业调度器按照相关算法对其进行调度（常用调度算法有先进先出、计算能力调度和公平调度三种算法）。

（4）作业被调度器调度执行时，会根据输入文件进行输入划分，为每个划分创建一个 Map 任务，并将 Map 任务分配给 TaskTracker 执行。根据“数据本地化”原则，Map 任务会被分配给含有该 Map 处理的文件块的 TaskTracker 上，同时将程序 Jar 文件从共享文件系统复制到该 TaskTracker 上，最后新建一个 TaskTracker 实例来运行任务。

（5）TaskTracker 每隔一段时间会给 JobTracker 发送一个心跳，告诉 JobTracker 它依然在运行。同时，心跳包还携带了很多信息，如当前 Map 任务完成的进度等信息。待所有 Task 执行完毕后，整个作业执行成功。

3.5　ZooKeeper

ZooKeeper 集群是一组 ZooKeeper 服务器的集合，它们共同协作，提供高可用性的 ZooKeeper 服务。ZooKeeper 是一个分布式的协调服务，用于管理分布式环境中的配置信息、命名服务、分布式锁等。在 ZooKeeper 集群中，多个 ZooKeeper 服务器协同工作，共同维护一个 ZooKeeper 服务，以提高其可用性和容错性。

3.5.1　ZooKeeper 集群

1．ZooKeeper 的工作机制

将 ZooKeeper 从设计模式角度来理解，它是一个基于观察者模式设计的分布式服务管理框架，负责存储和管理大家都关心的数据，然后接受观察者的注册。一旦这些数据的状态发生变化，ZooKeeper 就将负责通知已经在 ZooKeeper 上注册的那些观察者做出相应的反应。这也就是说，ZooKeeper = 文件系统 + 通知机制。

2．ZooKeeper 的特点

（1）ZooKeeper 是一个领导者（Leader）、多个跟随者（Follower）组成的集群。

（2）Zookeepe 集群中只要有半数以上节点存活，ZooKeeper 集群就能正常服务。所以 ZooKeeper 适合安装奇数台服务器。

（3）全局数据一致，即每个 Server 保存一份相同的数据副本，Client 无论连接到哪个 Server，数据都是一致的。

（4）更新请求顺序执行，即来自同一个 Client 的更新请求按其发送顺序依次执行，即先进先出。

（5）数据更新原子性，即一次数据更新要么成功、要么失败。

（6）实时性，即在一定时间范围内，Client 能读到最新数据。

3．ZooKeeper 的数据结构

ZooKeeper 的数据结构与 Linux 文件系统很类似，整体上可以看作一棵树，每个节点为一个 ZNode。每个 ZNode 默认能够存储 1MB 的数据，每个 ZNode 都可以通过其路径被唯一标识。

4．ZooKeeper 的应用场景

ZooKeeper 提供的服务包括统一命名服务、统一配置管理、统一集群管理、服务器动态上下线、软负载均衡等。

（1）统一命名服务。

在分布式环境下，经常需要对应用/服务进行统一命名，便于识别。例如，IP 不容易记住，而域名容易记住。

（2）统一配置管理。

① 在分布式环境下，配置文件同步非常常见。一般要求一个集群中，所有节点的配置信息是一致的，如 Kafka 集群。对配置文件修改后，希望能够快速同步到各个节点上。

② 配置管理可交由 ZooKeeper 实现。可将配置信息写入 ZooKeeper 上的一个 Znode，各个客户端服务器监听这个 Znode。一旦 Znode 中的数据被修改，ZooKeeper 将通知各个客户端服务器。

（3）统一集群管理。

① 在分布式环境下，实时掌握每个节点的状态是必要的，可根据节点实时状态做出一些调整。

② ZooKeeper 可以实现实时监控节点状态变化。可将节点信息写入 ZooKeeper 上的一个 ZNode，监听这个 ZNode 可获取它的实时状态变化。

（4）服务器动态上下线。

客户端能实时洞察到服务器上下线的变化。

（5）软负载均衡。

在 ZooKeeper 中记录每台服务器的访问数，让访问数最少的服务器去处理最新的客户端请求。

3.5.2　部署 ZooKeeper

在 ZooKeeper 集群环境下只要一半以上的机器正常启动了，那么 ZooKeeper 服务就是可用的。因此，在集群上部署 ZooKeeper 最好使用奇数台机器，这样如果有 5 台机器，只要 3 台正常工作则服务将正常。在目前的实际生产环境中，一个 Hadoop 集群最多有三台节点做备用 Master，即并不是所有节点都安装 ZooKeeper；如果以实验为目的，可以将所有节点都安装 ZooKeeper 并作为 Master 使用。

本书使用的 ZooKeeper 版本是 ZooKeeper-3-4-5.tar.gz，可以在 Apache 的官网下载，下载地址为 http://apache.fayea.com/zookeeper/。

1．解压

将下载好的 ZooKeeper 文件上传到 Hadoop 集群中的 Master 节点，使用命令“tar-zxvf apache-zookeeper-3.6.3-bin.tar.gz-C /hadoop/ ”将其解压。

2．配置 zoo.cfg

（1）创建文件夹。

```
sudo mkdir /hadoop/apache-zookeeper-3.6.3-bin/data   /hadoop/apache-zookeeper-3.6.3-bin/log
```

（2）进入 ZooKeeper 的 conf 目录修改 zoo.cfg。

```
cp zoo_sample.cfg   zoo.cfg
```

修改 zoo.cfg 的内容如下。

```
dataDir=/hadoop/apache-zookeeper-3.6.3-bin/data
dataLogDir=/hadoop/apache-zookeeper-3.6.3-bin/log
server.0=192.168.56.129:2888:3888
server.1=192.168.56.130:2888:3888
server.2=192.168.56.131:2888:3888
```

除了 dataDir 的内容是修改外，其他配置信息均为新增。

（3）在/hadoop/apache-zookeeper-3.6.3-bin/data 文件夹下创建 myid 文件，将其值修改为 0。需要注意的是，server.后面的数值必须和后面 IP 中的 myid 值保持一致，即 IP 为 192.168.56.130 的节点中 myid 的值必须为 1，IP 为 192.168.56.131 的节点中 myid 的值必须为 2。

（4）分发到 slave1 和 slave2 节点。

```
scp -r /hadoop/apache-zookeeper-3.6.3-bin/ slave1:/hadoop/
scp -r /hadoop/apache-zookeeper-3.6.3-bin/ slave2:/hadoop/
```

3．修改三个节点的环境变量

在 PATH 值末尾添加/hadoop/apache-zookeeper-3.6.3-bin/bin。

4．测试

在三个节点上分别运行 zkServer.sh start，启动后可以通过 zkServer.sh status 查看 ZooKeeper 的运行状态，其中只能有一个节点充当 leader，其余所有节点均为 follower，如图 3-32～图 3-34 所示。ZooKeeper 的进程名为 QuorumPeerMain。

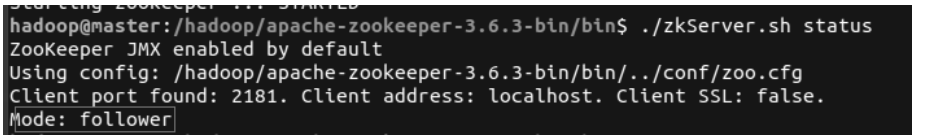

图 3-32　master 节点 ZooKeeper 运行状况

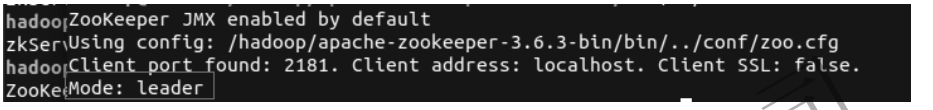

图 3-33　slave1 节点 ZooKeeper 运行状况

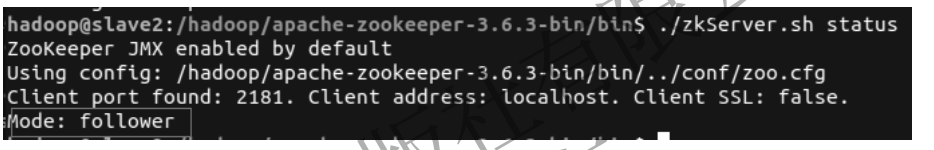

图 3-34　slave2 节点 ZooKeeper 运行状况

习　题

1．Hadoop 中的核心配置文件有哪些？

2．Hadoop 3.1.3 中的 Block 大小为多少？

3．格式化 HDFS 的命令是什么？

4．简述 shuffle 的工作原理。

5．ZooKeeper 实现数据发布/订阅的机制是什么？