# 分布式与并行计算项目设计报告书

### 班级

* 大数据1班

### 小组成员（学号）

* 严斌 （2018080910158）
* 王方永（2018080910126）
* 石家龙 （2018080910113）
* 李鑫雨 （2018080910063）
* 张蔓 （2018080910181）

## 前言

当今时代，互联网世界中每天都有数以亿计的数据产出，比如淘宝的商城订单，日志流量等数据。很多人都投身大数据领域，基于海量数据进行数据处理，从数据中挖掘出各种有价值的信息，而Hadoop作为大数据的基石，核心组件包括HDFS、MapReduce、YARN，为海量数据的存储和计算提供了可行方案。大数据涉及到数据采集、数据处理/分析/挖掘、数据存储、数据可视化等，各种大数据框架应运而生，而这些框架大都基于Hadoop，为大数据的处理提供多种多样的技术选型，共同组成了一个大的技术栈也就Hadoop生态系统。

Hadoop是由java语言编写的，在分布式服务器集群上存储海量数据并运行分布式分析应用的开源框架，其核心部件是HDFS与MapReduce。Hadoop的框架最核心的设计就是：HDFS和MapReduce。HDFS为海量的数据提供了存储，则MapReduce为海量的数据提供了计算。

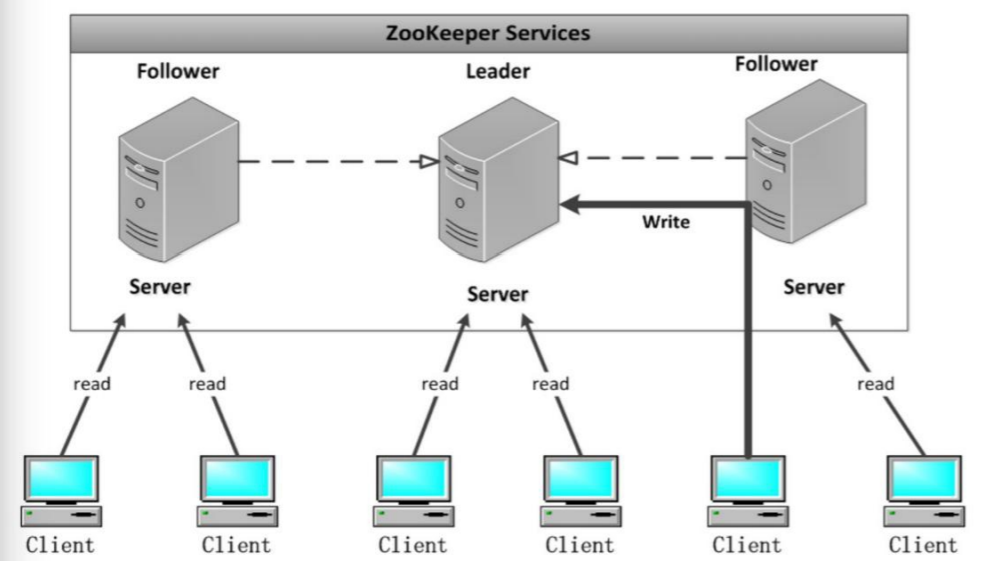
HDFS是一个分布式文件系统：引入存放文件元数据信息的服务器Namenode和实际存放数据的服务器Datanode，对数据进行分布式储存和读取。把HDFS理解为一个分布式的，有冗余备份的，可以动态扩展的用来存储大规模数据的大硬盘。本次项目安装的是Hadoop完全分布式，并实现HDFS的高可用，通过课本上内容配置了Zookeeper，Hadoop，了解了高可用的架构和各组件间的协作关系，在小组组员的共同努力下，一步一步不断尝试，直到最后搭建成功。

MapReduce是一个计算框架：MapReduce的核心思想是把计算任务分配给集群内的服务器里执行。通过对计算任务的拆分（Map计算/Reduce计算）再根据任务调度器（JobTracker）对任务进行分布式计算。MapReduce可以理解成为一个计算引擎，按照MapReduce的规则编写Map计算、Reduce计算的程序，可以完成计算任务。集群搭建成功后我们就是使用了MapReduce自带的单词计数案例，这里的单词计数实际上是调用了它本身写好的countword类，通过HDFS上传文件后使用MapReduce计算后得出结果，后面还会有介绍。

## 项目框架介绍

* **Zookeeper的架构模型**

Zookeeper集群是一个基于主从架构的高可用集群



每个服务器承担如下三种角色中的一种：

**Leader：** 一个Zookeeper集群同一时间只会有一个实际工作的Leader，它会发起并维护与各Follwer及Observer间的心跳。所有的写操作必须要通过Leader完成再由Leader将写操作广播给其它服务器。

**Follower ：**一个Zookeeper集群**可能**同时存在多个Follower，它会响应Leader的心跳。Follower可直接处理并返回客户端的读请求，同时会将写请求转发给Leader处理，并且负责在Leader处理写请求时对请求进行投票。

**Observer ：**角色与Follower类似，但是无投票权。

**Zookeeper的应用场景：**

1. 数据发布/订阅
2. 命名服务
3. 分布式协调/通知
4. 分布式锁
5. 分布式队列

这里主要介绍**分布式协助/通知：**

Zookeeper中特有的Watcher注册于异步通知机制，能够很好地实现分布式环境下不同机器，甚至不同系统之间的协调与通知，从而实现对数据变更的实时处理。通常的做法是不同的客户端都对Zookeeper上的同一个数据节点进行Watcher注册，监听数据节点的变化（包括节点本身和子节点），若数据节点发生变化，那么所有订阅的客户端都能够接收到相应的Watcher通知，并作出相应处理。在绝大多数分布式系统中，系统机器间的通信无外乎 心跳检测、工作进度汇报和系统调

度 。

**➀心跳检测：**不同机器间需要检测到彼此是否在正常运行，可以使用Zookeeper实现机 器间的心跳检测，基于其临时节点特性（临时节点的生存周期是客户端会话，客户端若当即 后，其临时节点自然不再存在），可以让不同机器都在Zookeeper的一个指定节点下创建临时 子节点，不同的机器之间可以根据这个临时子节点来判断对应的客户端机器是否存活。通过 Zookeeper可以大大减少系统耦合。

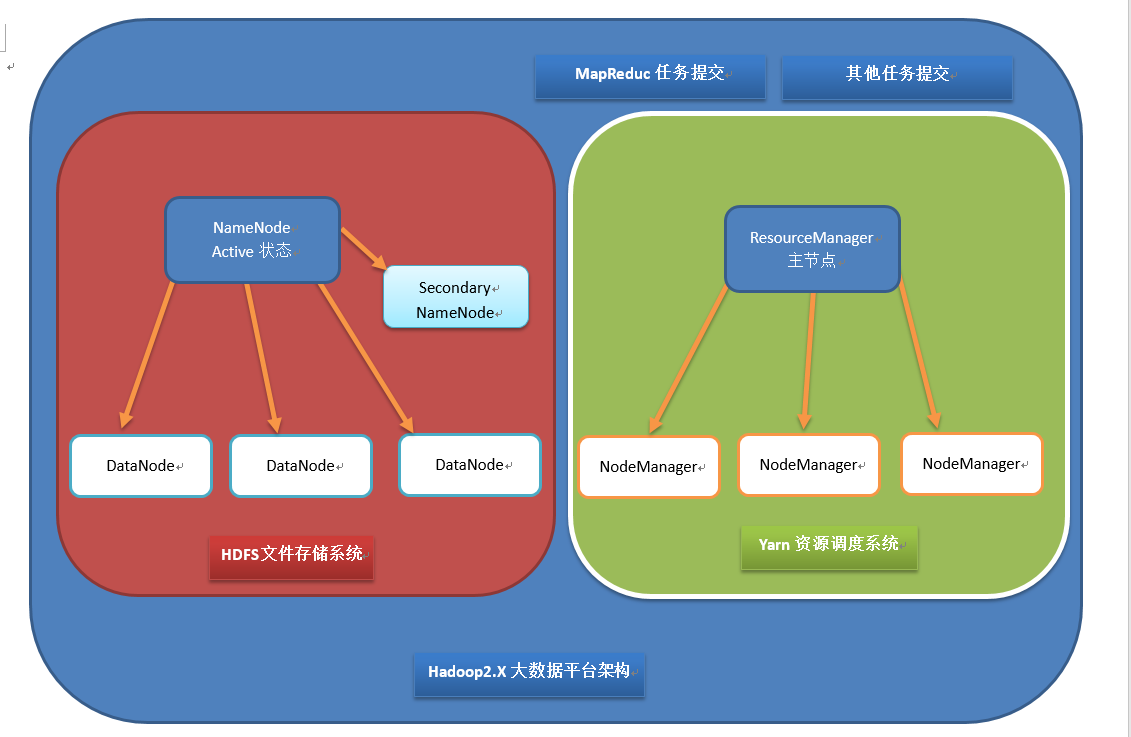
**➁工作进度汇报：**通常任务被分发到不同机器后，需要实时地将自己的任务执行进度汇报给分发系统，可以在Zookeeper上选择一个节点，每个任务客户端都在这个节点下面创建临时子节点，这样不仅可以判断机器是否存活，同时各个机器可以将自己的任务执行进度写到该临时节点中去，以便中心系统能够实时获取任务的执行进度。

**➂系统调度：**Zookeeper能够实现如下系统调度模式：分布式系统由控制台和一些客户端系统两部分构成，控制台的职责就是需要将一些指令信息发送给所有的客户端，以控制他们进行相应的业务逻辑，后台管理人员在控制台上做一些操作，实际上就是修改Zookeeper上某些节点的数据，Zookeeper可以把数据变更以时间通知的形式发送给订阅客户端。

* **Hadoop的架构模型**

**1、2.x的版本架构模型介绍**

**第一种：NameNode与ResourceManager单节点架构模型。**



文件系统核心模块：

NameNode：集群当中的主节点，主要用于管理集群当中的各种数据。

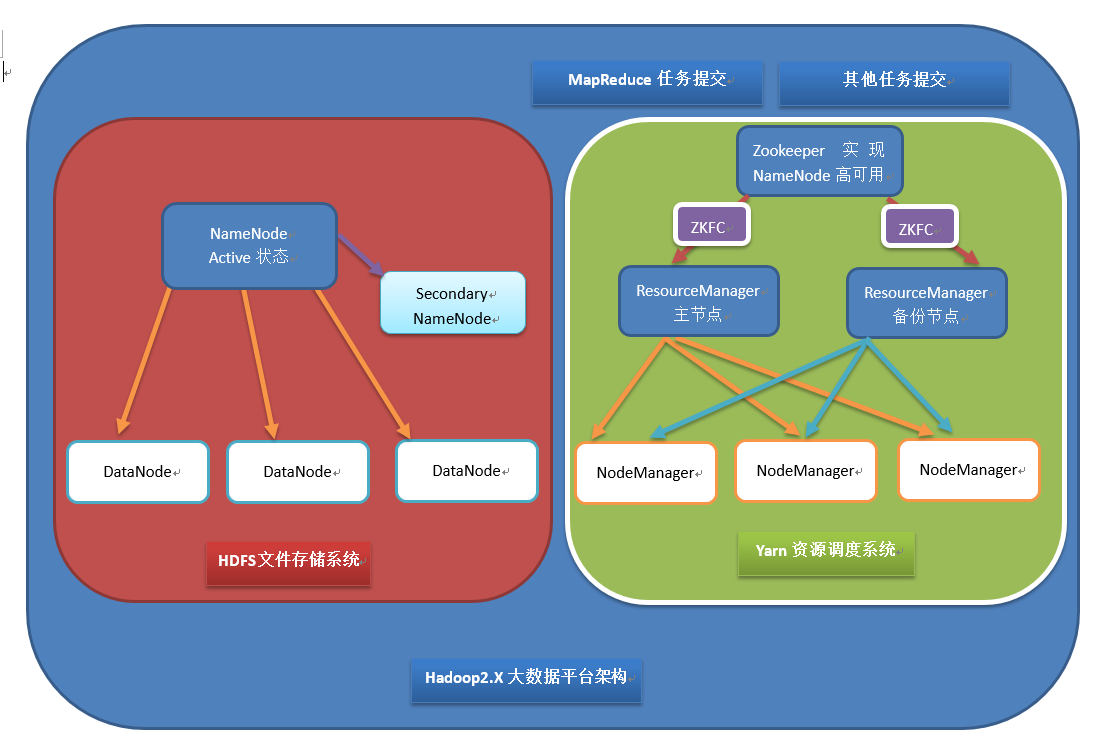
secondaryNameNode：主要能用于hadoop当中元数据信息的辅助管理。

DataNode：集群当中的从节点，主要用于存储集群当中的各种数据。

数据计算核心模块：

ResourceManager：接收用户的计算请求任务，并负责集群的资源分配。

NodeManager：负责执行主节点APPmaster分配的任务。**第二种：NameNode单节点与ResourceManager高可用架构模型。**



文件系统核心模块：

NameNode：集群当中的主节点，主要用于管理集群当中的各种数据。

secondaryNameNode：主要能用于hadoop当中元数据信息的辅助管理。

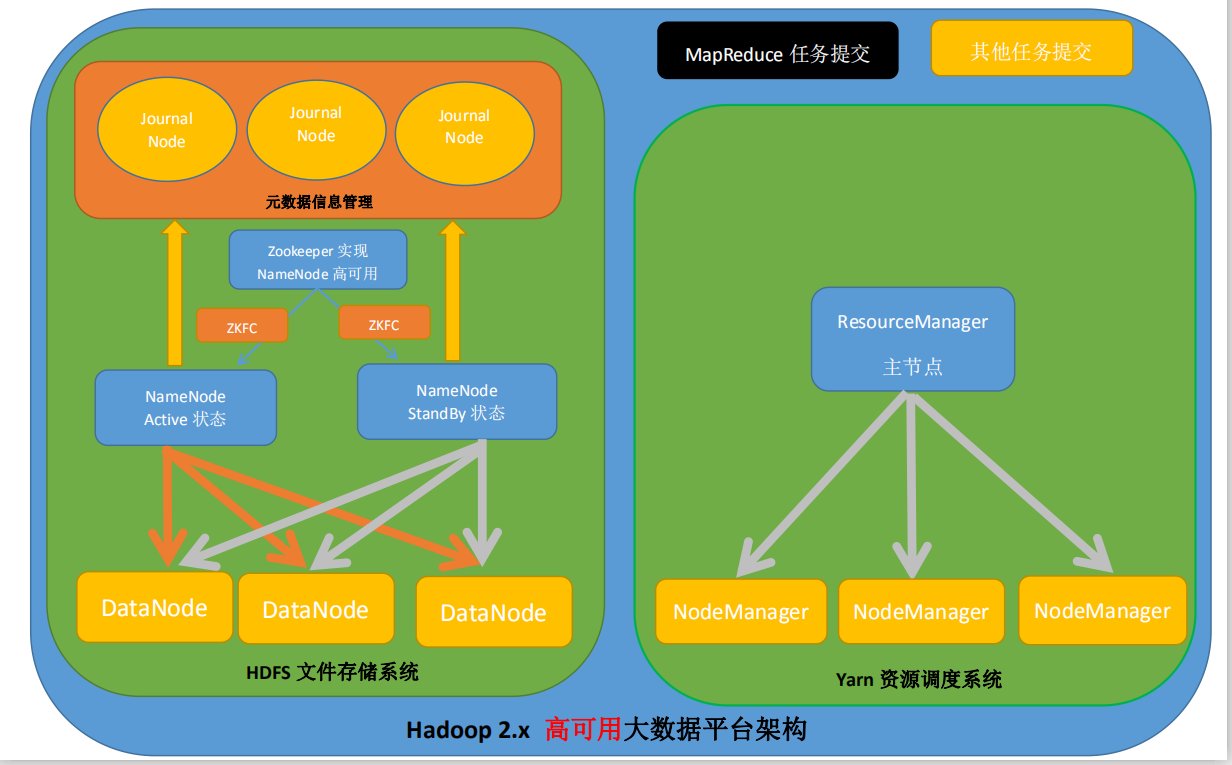
DataNode：集群当中的从节点，主要用于存储集群当中的各种数据。

数据计算核心模块：

ResourceManager：接收用户的计算请求任务，并负责集群的资源分配，以及计算任务的划分，通过zookeeper实现ResourceManager的高可用。

NodeManager：负责执行主节点ResourceManager分配的任务。

**第三种：NameNode高可用与ResourceManager单节点架构模型。**



文件系统核心模块：

NameNode：集群当中的主节点，主要用于管理集群当中的各种数据，其中nameNode可以有两个，形成高可用状态。

DataNode：集群当中的从节点，主要用于存储集群当中的各种数据。

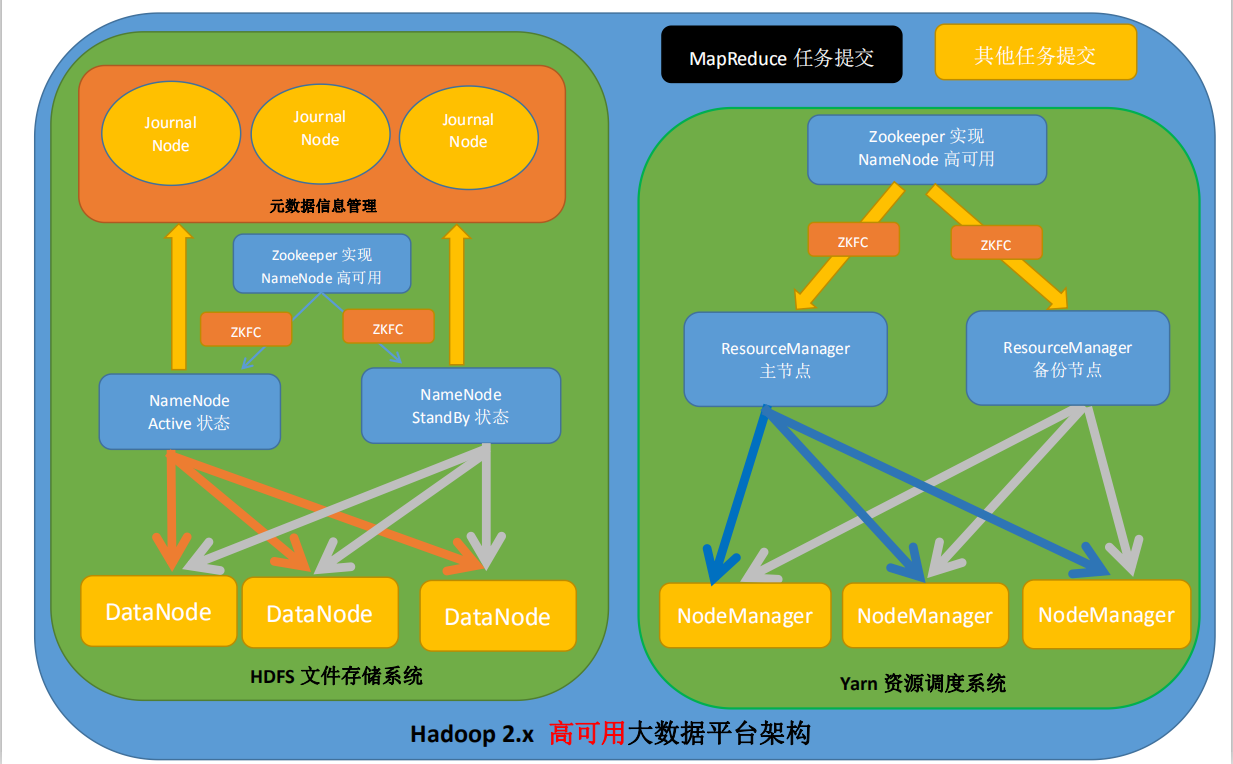
JournalNode：文件系统元数据信息管理数据。

计算核心模块：

ResourceManager：接收用户的计算请求任务，并负责集群的资源分配，以及计算任务的划分。

NodeManager：负责执行主节点ResourceManager分配的任务。

**第四种：NameNode与ResourceManager高可用架构模型**



文件系统核心模块：

NameNode：集群当中的主节点，主要用于管理集群当中的各种数据，一般都是使用两个，主要有两种状态：active和standby，实现HA高可用。

JournalNode：元数据信息管理进程，一般都是奇数个。JournalNode节点共享edits日志文件。

FailoverController（ZKFC）：节点用于监控和控制NameNode的状态切换，当一个节点挂掉后，会把standbyNameNode状态切换为active。

DataNode：从节点，用于数据的存储。定时和NameNode进行通信，接受NameNode指令，DataNode之间还会相互通信，执行数据块复制任务。

数据计算核心模块：

ResourceManager：Yarn平台的主节点，主要用于接收各种任务，通过两个，状态分别为active和standby，构建成高可用。

NodeManager：Yarn平台的从节点，主要用于处理ResourceManager分配的任务。

**2、Hadoop环境网络特性**

Hadoop集群中的各节点通过网络连接起来，而且MapReduce中的以下过程会在网络中传输数据。网络对Hadoop的控制层非常重要，比如HDFS的信令和运维操作，以及MapReduce架构都受到网络影响。

**（1） 写数据。**当向HDFS写入初始数据或者大块数据时，会发生数据写入过程。写入的数据块需要备份到其他节点，需要在网络中传输这些数据。

**（2） 作业执行。**

**（3） 读数据。**当应用程序如网站、索引或者SQL数据库从HDFS读取数据时，会发生数据读取的过程。

**五种网络特性**

**（1） 网络可用性和弹性。**要部署一个高冗余性和可扩展的网络，支持Hadoop集群的增长。在Datanode之间部署多条链路的技术要比那些有单点失效或两点失效的技术要好。交换机和路由器已经在业界被证明能够为服务器提供网络可用性。

**（2） Burst流量突发处理和队列深度。**HDFS的有些操作和MapReduce Job会产生突发流量，如向HDFS加载文件或者把结果文件写入HDFS都需要通过网络。网络如果处理不了突发流量，就会丢弃数据包，所以适当的缓存可以缓解突发流量的影响。确保选择使用缓存和队列的交换机和路由器，来有效处理流量突发。

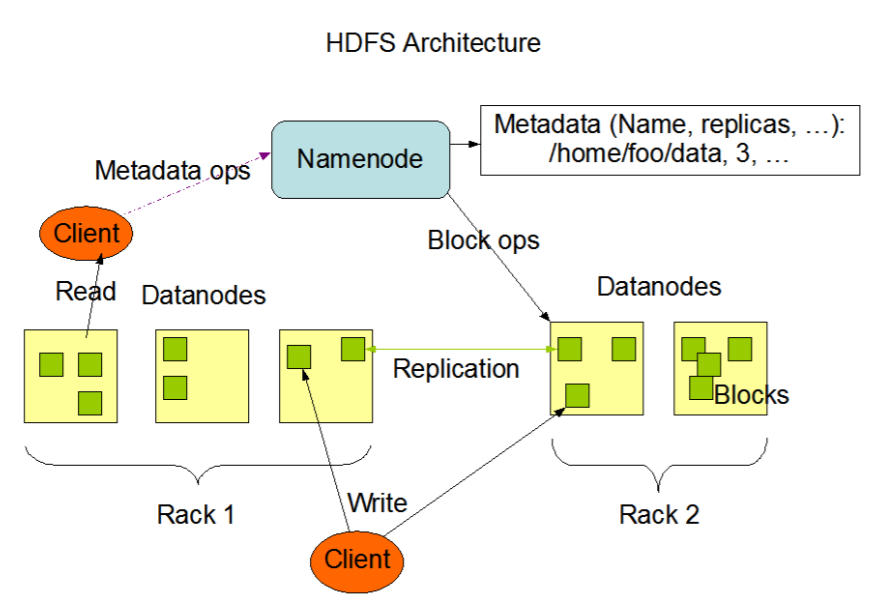
**（3） 网络过载比。**一个好的网络设计需要考虑到网络中关键节点的拥塞情况。

**（4） Datanode网络接入。**要基于集群工作负荷来推荐带宽配置。

**（5） 网络延迟。**交换机和路由器延迟的变化对集群性能的影响有限。相比网络延迟，应用层延迟对任务的影响比例更大。但是网络的延迟会对应用系统造成潜在的影响，例如造成不必要的应用切换等。

## 项目技术介绍

**一、HDFS**



HDFS即Hadoop分布式文件系统（Hadoop Distributed File System），HDFS是一个高度容错性的系统，适合部署在廉价的机器上，HDFS能提供高吞吐量的数据访问，非常适合大规模数据集上的应用。HDFS包含了一个单独的名称节点NameNode和多个数据节点DataNode。

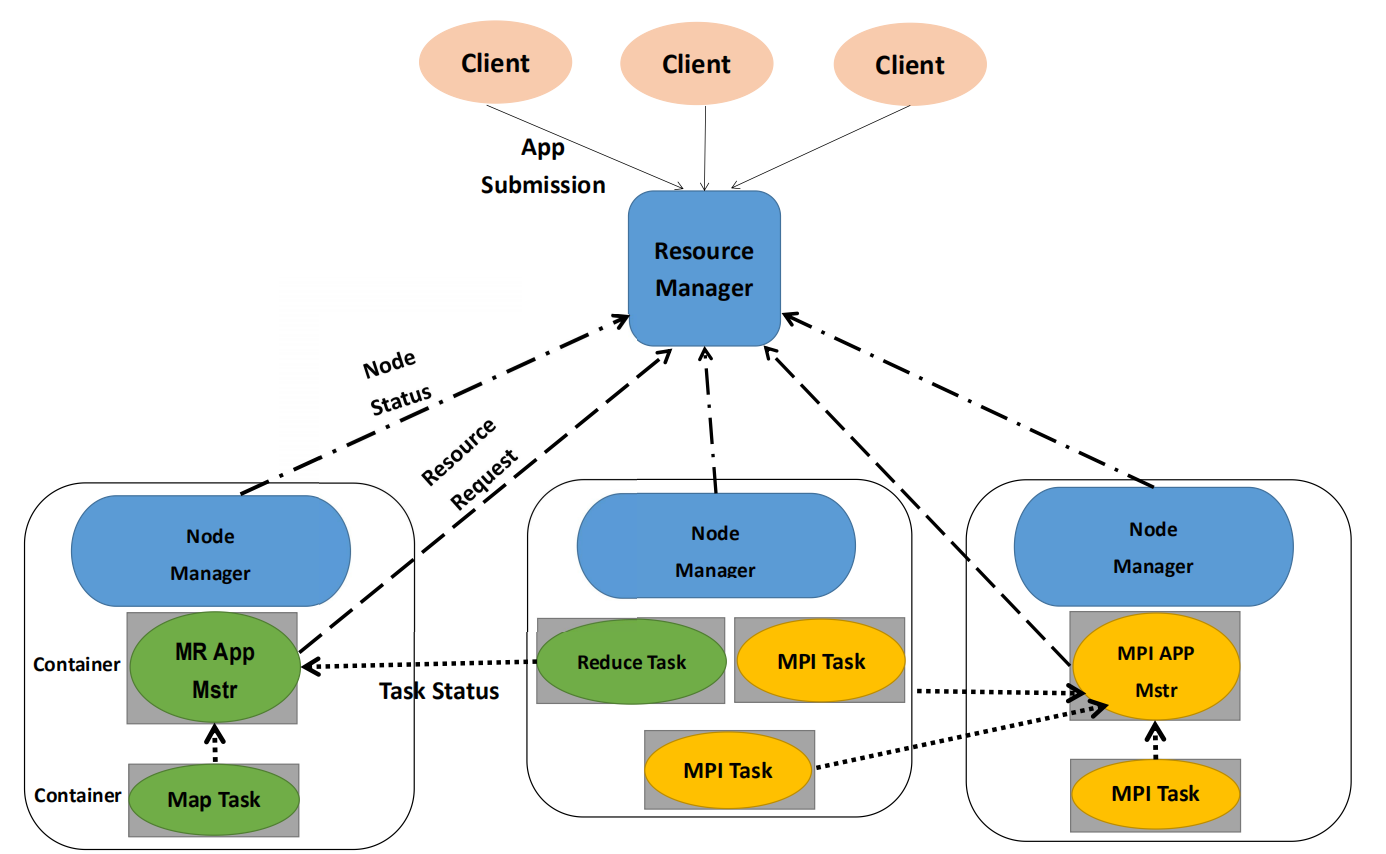
NameNode存储了HDFS的元数据，是整个文件系统目录，基于内存存储，存储的是一些文件的详细信息。NameNode在内存中保存着整个文件系统的名字空间和文件数据块映射的映像，数据块和DataNode列表的映射关系。其中文件名和数据块的关系保存在磁盘上，但是NameNode上不保存数据块和DataNode列表的关系，该列表是通过DataNode上报建立起来的。

DataNode负责实际的数据存储，并将数据定期汇报给NameNode。DataNode按大小分成若干个Block，存储到不同的节点上。Block大小和副本数通过Client端上传文件时设置，文件上传成功后副本数可以变更，BlockSize不可变更。默认情况下每个Block都有3个副本。

SecondaryNameNode（辅助节点），用于同步元数据信息，辅助NameNode对fsimage和edits的合并。

**二、YARN**

YARN的核心思想是将功能分开，分别由ResourceManager和ApplicationMaster进程来实现。其中ResourceManager负责整个系统的资源管理和调度，而ApplicationMaster则负责应用程序的相关事务，如任务调度、容错和任务监控等。



从架构图来看，它主要由ResourceManager和ApplicationMaster、NodeManager、ApplicationMaster和Container等组件组成。

**1、Resource Manager**

(1) 对提交的Application进行资源分配。

(2)和NodeManager进行通信，得到每个NodeManage上的资源情况r

(3)启动和监控Application Master

**2、Node Manager**

管理yarn集群中的每个节点，提供从监督对一个容器的终生管理到监视资源和跟踪节点健康的服务，这些容器代表着可供一个特定应用程序的使用针对每个节点的资源。

(1) 管理本节点的资源情况

(2)处理来自Resource Manager的命令

(3)处理来自Applicaiton Manager 的命令

**3、Application Master**

每个应用有一个，负责应用程序的管理。和Resource Manager通信，获取资源。且通过和Node Manager监控container的运行情况（执行情况和资源情况），任务的监控和容错。

(1) 负责数据的切分

(2)为应用程序申请资源并分配给内部的任务

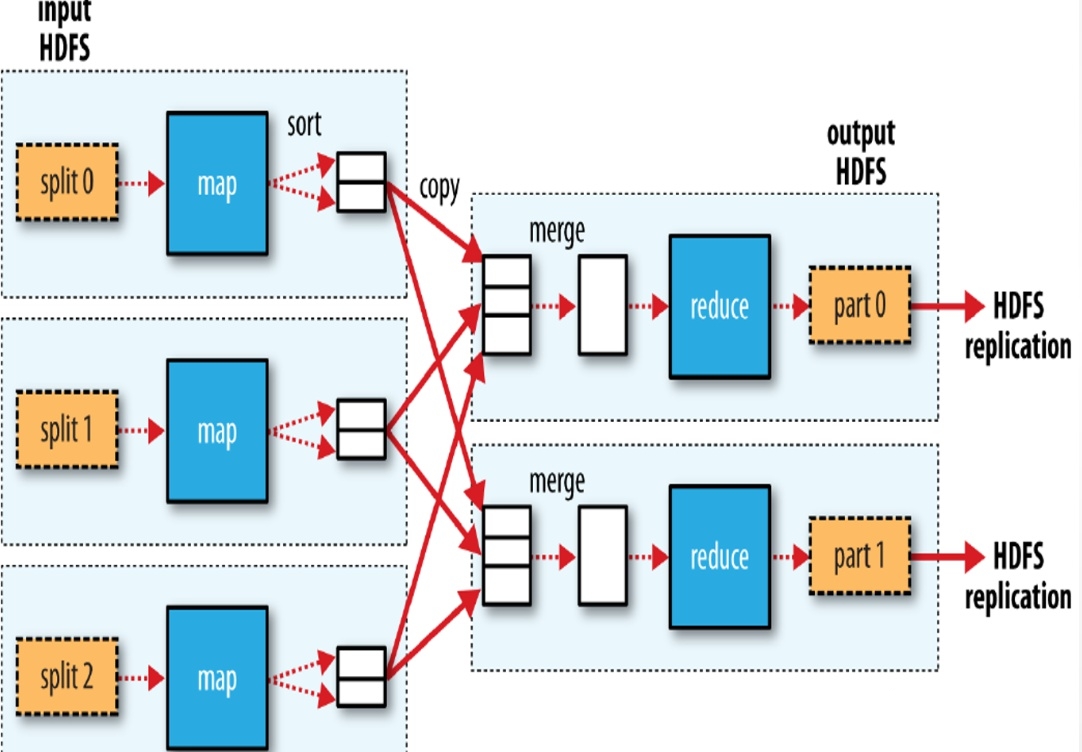
(3)任务的监控与容错

**4、Container**

Yarn上的资源抽象，封装节点上的多维度资源，当AM向RM申请资源时，RM为AM返回的资源便是用Container表示。Yarn会为每个任务分配一个Container。且该任务只能使用该Container中的资源。

**三、MapReduce**

MapReduce是一种分布式计算框架，以一种可靠的，具有容错能力的方式并行地处理上TB级别的海量数据集。主要用于搜索领域，解决海量数据的计算问题。



一个MapReduce作业通常会把输入的数据集切分为若干独立的数据块，由map任务以完全并行的方式处理它们。框架会对map的输出先进行排序，然后把结果输入给reduce任务。通常作业的输入和输出都会被存储在文件系统中。整个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。MapReduce框架和分布式文件系统是运行在一组相同的节点上的，计算节点和存储节点通常在一起。这种配置允许框架在那些已经存好数据的节点上高效地调度任务，这可以使整个集群的网络带宽被非常高效地利用。

一个MapReduce任务包含一般会这几个部分：Split、Map、Shuffle、Reduce。

1、split阶段：

此阶段，每个输入文件被分片输入到map。如一个文件有200M，默认会被分成2片，因为每片的默认最大值和每块的默认值128M相同。

2、map阶段：

此阶段，执行map任务。map数由分片决定，若要增加map数，可增大mapred.map.tasks，若减少map数，可增大mapred.min.split.size。

3、shuffle阶段：

此阶段，将map的输出经过“整理”后给到reduce，也称为“混洗”。分为map端操作和reduce端操作。

4、reduce阶段：

执行reduce任务。reduce数量由分区数决定，结果文件的数量也由此决定，且记录默认按key升序排列。

## 项目实施过程

**在Ubuntu系统下搭建Hadoop集群，并实现HDFS高可用：**

* 网络环境搭建
* 安装JDK
* 安装Zookeeper
* 安装Hadoop
* 启动集群
* 查看集群
* 案例运行测试

**所需系统和工具准备：**

1. 三台已安装好Ubuntu20.04系统的笔记本电脑
2. java -1.8.0\_141
3. hadoop-2.7.5
4. zookeeper-3.5.7

* **网络环境搭建（三台）：**

1.创建自定义用户

1.1创建用户：

sudo useradd -m hadoop -s /bin/bash

1.2设置用户密码：

sudo passpd hadoop

1.3设置用户权限：

编辑sudoers文件：

sudo vim /etc/sudoers

写入以下内容并保存退出：

hadoop ALL=(ALL) NOPASSWD: ALL

2.修改主机名并配置主机名映射

2.1打开 hostname文件写入HOSTNAME=hadoop1保存退出：

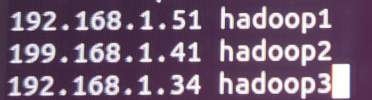
vim /etc/hostname

节点二、三写入hadoop2、hadoop3

2.2打开hosts文件：

vim /etc/hosts

写入：



主机名、主机名映射修改好后重启生效，节点二、三操作同上。

3.修改IP地址

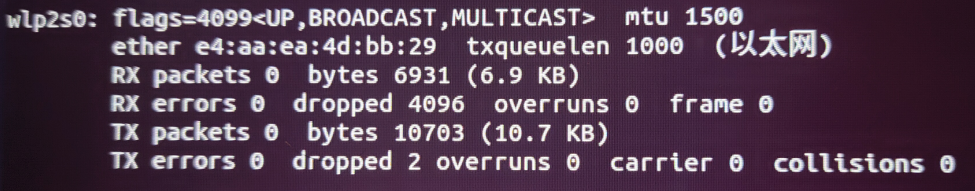
3.1使用有线或无线连接局域网（二选一）

有线连接：交换机、采用T568B标准接好的水晶头网线

无线连接：路由器

3.2查看IP地址：

Ifconfig



修改IP地址：

sudo ifconfig wlp2s0 192.168.1.51



节点二、三分别修改为：

192.168.1.41

192.168.1.34

4.配置SSH免密登录

4.1安装ssh服务：

sudo apt-get update #更新apt

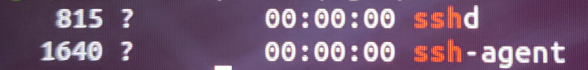
sudo apt-get install openssh-server #安装服务，一路回车

sudo /etc/init.d/ssh restart  #启动服务

sudo ufw disable #关闭防火墙

执行以下命令看是否成功运行，如果成功的话，会出现俩个进程

ps –e | grep ssh



4.2然后在hadoop1节点生成SSH公钥，让 hadoop1节点可以无密码 SSH 本机，在hadoop1节点上执行

ssh-keygen -t rsa # 一直按回车就可以

cat ./id\_rsa.pub >> ./authorized\_keys

完成后可执行 ssh hadoop1验证一下（需要输入 yes，成功后执行exit返回原来的终端）

ssh hadoop1

4.3接着在hadoop1节点将上公匙传输到 hadoop2节点，过程中需要输入 hadoop2 节点的密码，传输100%以后就是传过去了

scp ~/.ssh/id\_rsa.pub hadoop@hadoop2:/home/hadoop/

在 hadoop2节点上，把公钥加入授权

cat ~/id\_rsa.pub >> ~/.ssh/authorized\_keys

4.4对 hadoop3重复上面两步，这样hadoop1节点就可以无密码登陆hadoop2、hadoop3节点了。

ssh hadoop2 #可以用来检验是否能成功登陆，exit退出

* **安装JDK （三台）**

**1.安装jdk-1.8.0\_275**

创建jvm文件夹:

mkdir /usr/lib/jvm

解压到/usr/lib/jvm目录下

sudo tar zxvf jdk-1.8.0\_275.tar.gz -C /usr/lib/jvm/

进入jvm目录:

cd /usr/lib/jvm/

重命名为:

mv jdk-1.8.0\_275 java

**2.配置jdk环境变量**

编辑bashrc文件：

vim ~/.bashrc

写入以下内容:

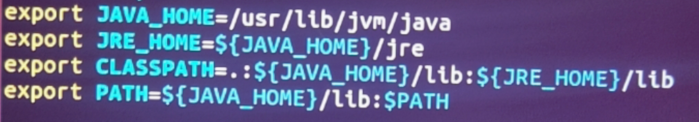
#Java Environment

export JAVA\_HOME=/usr/lib/jvm/java

export JRE\_HOME=${JAVA\_HOME}/jre

export CLASSPATH=.:${JAVA\_HOME}/lib:${JRE\_HOME}/lib

export PATH=${JAVA\_HOME}/bin:$PATH



修改后保存退出执行以下命令：

source ~/.bashrc

检测是否安装成功，查看java版本：

java –version



* **安装zookeeper**

**1.安装zookeeper-3.5.7**

解压到/usr/local/目录下

sudo tar zxvf apache-zookeeper-3.5.7.tar.gz -C /usr/local/

进入local目录:

cd /usr/local/

重命名为zookeeper-3.5.7:

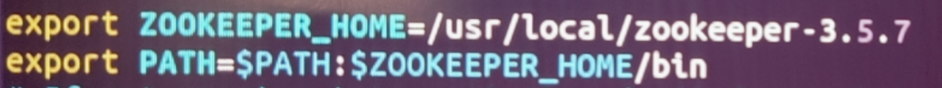
mv apache-zookeeper-3.5.7 zookeeper-3.5.7

**2.配置zookeeper环境变量**

编辑bashrc文件：

vim ~/.bashrc

写入以下内容：



**3.修改zookeeper配置文件**

进入conf文件：

cd $ZOOKEEPER\_HOME/conf

将zoo\_sample.cfg拷贝到zoo.cfg:

cp zoo\_sample.cfg zoo.cfg

编辑zoo.cfg文件：

vim zoo.cfg

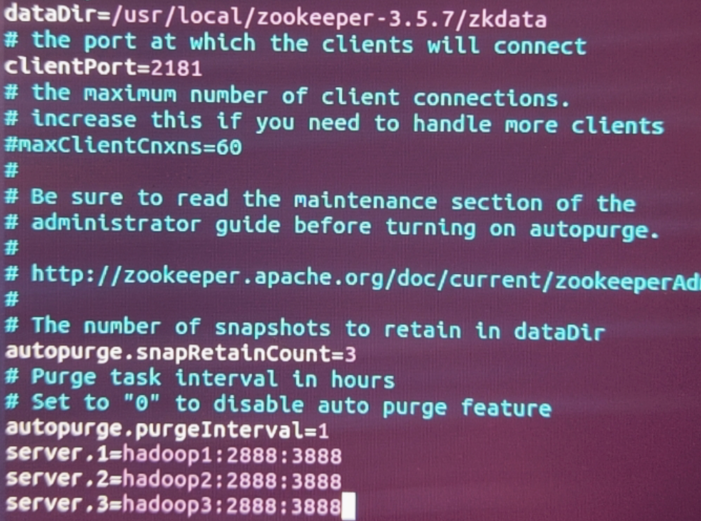
写入以下内容：

dataDir=/usr/local/zookeeper-3.5.7/zkdatas # 保留多少个快照autopurge.snapRetainCount=3 # 日志多少小时清理一次 autopurge.purgeInterval=1 # 集群中服务器地址

server.1=hadoop1:2888:3888

server.2=hadoop2:2888:3888

server.3=hadoop3:2888:3888



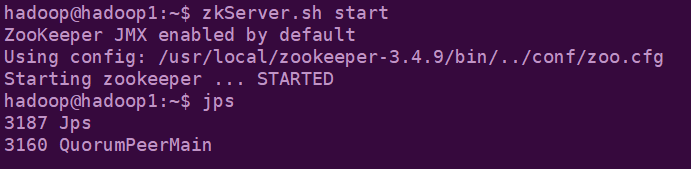
在/usr/local/zookeeper-3.5.7/下创建zkdir文件夹

mkdir –r zkdir

在zkdir文件下创建文件myid并写入数字1保存退出（节点二、三修改为2、3）：

vim myid

启动zookeeper：zkServer.sh start



**到此hadoop1上的zookeeper安装并配置完毕,下面通过scp命令将zookeeper发送到hadoop2，hadoop3两个节点上，由于部分文件的权限问题，直接使用scp命令发送文件会有失败的情况，这里直接将zookeeper文件进行压缩打包后通过scp发送到其他节点最后解压到所需路径即可。后面的hadoop文件配置完毕后发送到其他节点也是同理**

**4.打包zookeeper发送到hadoop2、hadoop3**

cd /usr/local #进入zookeeper目录

sudo tar –zcf /home/hadoop/zookeeper.master.tar.gz zookeeper-3.5.7

cd #回到有压缩包的路径下

scp zookeeper.master.tar.gz hadoop@hadoop2:/home/hadoop/

scp zookeeper.master.tar.gz hadoop@hadoop3:/home/hadoop/

在hadoop2上ls查看压缩包，解压到/usr/local/目录下



sudo tar –zxf zookeeper.master.tar.gz –C /usr/local/

在hadoop3上重复解压步骤，并分别修改myid文件内容为2、3

启动三个zookeeper：

hadoop@hadoop1:~$ zkServer.sh start

hadoop@hadoop2:~$ zkServer.sh start

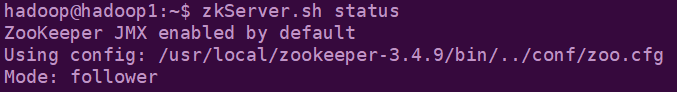
hadoop@hadoop3:~$ zkServer.sh start

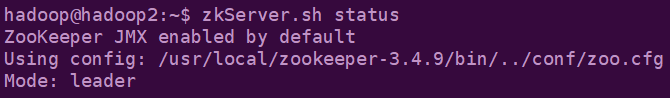
查看zookeeper状态：

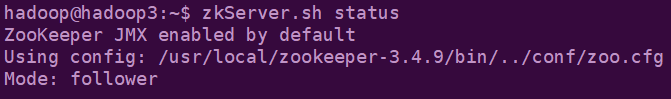
hadoop@hadoop1:~$ zkServer.sh status

hadoop@hadoop2:~$ zkServer.sh status

hadoop@hadoop3:~$ zkServer.sh status







* **安装hadoop**

**1.安装Hadoop-2.7.5**

解压hadoop-2.7.5.tar.gz到/usr/local

sudo tar -zxf hadoop-2.7.5.tar.gz -C /usr/local

进入/usr/local/

cd /usr/local/

将文件夹名改为hadoop并修改权限

sudo mv hadoop-2.7.5 hadoop

sudo chown –R hadoop hadoop

**2.配置hadoop环境变量**

将下面代码添加到.bashrc文件：

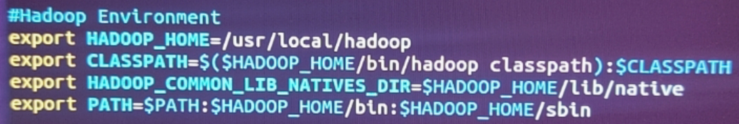
#Hadoop Environment

export HADOOP\_HOME=/usr/local/hadoop

export CLASSPATH=$($HADOOP\_HOME/bin/hadoop classpath):$CLASSPATH

export HADOOP\_COMMON\_LIB\_NATIVE\_DIR=$HADOOP\_HOME/lib/native

export PATH=$PATH:$HADOOP\_HOME/bin:$HADOOP\_HOME/sbin



保存退出后执行.bashrc文件：

source ~/.bashrc

**3.修改hadoop配置文件**

**创建所需目录：**

mkdir -p /usr/local/ hadoop/datatmp

mkdir -p / usr/local/ hadoop / dfs/journalnode\_data

mkdir -p / usr/local/hadoop / dfs/edits

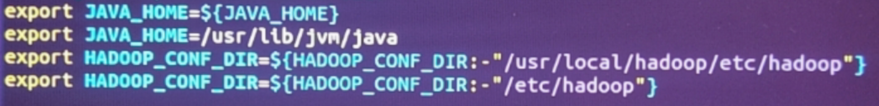
mkdir -p usr/local/ hadoop / dfs/datanode\_data

mkdir -p /usr/local/ hadoop/ dfs/namenode\_data

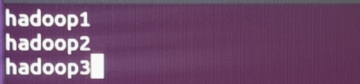
需要修改 usr/local/ hadoop/etc/hadoop 中的6个配置文件。

hadoop-env.sh 、slaves、core-site.xml、hdfs-site.xml、mapred-site.xml、yarn-site.xml

将以下内容写入hadoop-env.sh文件



**slaves**



**core-site.xml**改为如下配置：

<configuration>

<property>

<!--指定hadoop集群在zookeeper上注册的节点名-->

<name>fs.defaultFS</name>

<value>hdfs://hacluster</value>

</property>

<property>

<!--用来指定hadoop运行时产生文件的存放目录-->

<name>hadoop.tmp.dir</name>

<value>file:///usr/local/hadoop/datatmp</value>

</property>

<property>

<!--设置缓存大小，默认4kb-->

<name>io.file.buffer.size</name>

<value>4096</value>

</property>

<property>

<!--指定zookeeper的存放地址 -->

<name>ha.zookeeper.quorum</name>

<value>hadoop1:2181,hadoop2:2181,hadoop3:2181</value>

</property>

</configuration>

**hdfs-site.xml**

<configuration>

<property>

<!--数据块默认大小128M-->

<name>dfs.block.size</name>

<value>134217728</value>

</property>

<property>

<!--副本数量，不配置的话默认为3-->

<name>dfs.replication</name>

<value>3</value>

</property>

<property>

<!--namenode节点数据（元数据）的存放位置-->

<name>dfs.name.dir</name>

<value>file:///usr/local/hadoop/dfs/namenode\_data</value>

</property>

<property>

<!--datanode节点数据（元数据）的存放位置-->

<name>dfs.data.dir</name>

<value>file:///usr/local/hadoop/dfs/datanode\_data</value>

</property>

<property>

<name>dfs.webhdfs.enabled</name>

<value>true</value>

</property>

<property>

<name>dfs.datanode.max.transfer.threads</name>

<value>4096</value>

</property>

<property>

<!--指定hadoop集群在zookeeper上注册的节点名-->

<name>dfs.nameservices</name>

<value>hacluster</value>

</property>

<property>

<!-- hacluster集群下有两个namenode，分别为nn1,nn2 -->

<name>dfs.ha.namenodes.hacluster</name>

<value>nn1,nn2</value>

</property>

<!-- nn1的rpc、servicepc和http通信 -->

<property>

<name>dfs.namenode.rpc-address.hacluster.nn1</name>

<value>hadoop1:9000</value>

</property>

<property>

<name>dfs.namenode.servicepc-address.hacluster.nn1</name>

<value>hadoop1:53310</value>

</property>

<property>

<name>dfs.namenode.http-address.hacluster.nn1</name>

<value>hadoop1:50070</value>

</property>

<!-- nn2的rpc、servicepc和http通信 -->

<property>

<name>dfs.namenode.rpc-address.hacluster.nn2</name>

<value>hadoop2:9000</value>

</property>

<property>

<name>dfs.namenode.servicepc-address.hacluster.nn2</name>

<value>hadoop2:53310</value>

</property>

<property>

<name>dfs.namenode.http-address.hacluster.nn2</name>

<value>hadoop2:50070</value>

</property>

<property>

<!-- 指定namenode的元数据在JournalNode上存放的位置 -->

<name>dfs.namenode.shared.edits.dir</name>

<value>qjournal://hadoop1:8485;hadoop2:8485;hadoop3:8485/hacluster</value>

</property>

<property>

<!-- 指定JournalNode在本地磁盘存放数据的位置 -->

<name>dfs.journalnode.edits.dir</name>

<value>/usr/local/hadoop/dfs/journalnode\_data</value>

</property>

<property>

<!-- namenode操作日志的存放位置 -->

<name>dfs.namenode.edits.dir</name>

<value>/usr/local/hadoop/dfs/edits</value>

</property>

<property>

<!-- 开启namenode故障转移自动切换 -->

<name>dfs.ha.automatic-failover.enabled</name>

<value>true</value>

</property>

<property>

<!-- 配置失败自动切换实现方式 -->

<name>dfs.client.failover.proxy.provider.hacluster</name>

<value>org.apache.hadoop.hdfs.server.namenode.ha.ConfiguredFailoverProxyProvider</value>

</property>

<property>

<!-- 配置隔离机制 -->

<name>dfs.ha.fencing.methods</name>

<value>sshfence</value>

</property>

<property>

<!-- 使用隔离机制需要SSH免密登录 -->

<name>dfs.ha.fencing.ssh.private-key-files</name>

<value>/home/hadoop/.ssh/id\_rsa</value>

</property>

<property>

<!--hdfs文件操作权限,false为不验证-->

<name>dfs.permissions</name>

<value>false</value>

</property>

</configuration>

**mapred-site.xml**

可能默认文件名为 mapred-site.xml.template ，此时需要重命名

mv mapred-site.xml.template mapred-site.xml

写入以下内容：

<configuration>

<property>

<!--指定mapreduce运行在yarn上-->

<name>mapreduce.framework.name</name>

<value>yarn</value>

</property>

<property>

<!--配置任务历史服务器地址-->

<name>mapreduce.jobhistory.address</name>

<value>hadoop1:10020</value>

</property>

<property>

<!--配置任务历史服务器web-UI地址-->

<name>mapreduce.jobhistory.webapp.address</name>

<value>hadoop1:19888</value>

</property>

<property>

<!--开启uber模式-->

<name>mapreduce.job.ubertask.enable</name>

<value>true</value>

</property>

</configuration>

**yarn-site.xml**改为如下配置：

<configuration>

<property>

<!-- 开启Yarn高可用 -->

<name>yarn.resourcemanager.ha.enabled</name>

<value>true</value>

</property>

<property>

<!-- 指定Yarn集群在zookeeper上注册的节点名 -->

<name>yarn.resourcemanager.cluster-id</name>

<value>hayarn</value>

</property>

<property>

<!-- 指定两个ResourceManager的名称 -->

<name>yarn.resourcemanager.ha.rm-ids</name>

<value>rm1,rm2</value>

</property>

<property>

<!-- 指定rm1的主机 -->

<name>yarn.resourcemanager.hostname.rm1</name>

<value>hadoop2</value>

</property>

<property>

<!-- 指定rm2的主机 -->

<name>yarn.resourcemanager.hostname.rm2</name>

<value>hadoop3</value>

</property>

<property>

<!-- 配置zookeeper的地址 -->

<name>yarn.resourcemanager.zk-address</name>

<value>hadoop1:2181,hadoop2:2181,hadoop3:2181</value>

</property>

<property>

<!-- 开启Yarn恢复机制 -->

<name>yarn.resourcemanager.recovery.enabled</name>

<value>true</value>

</property>

<property>

<!-- 配置执行ResourceManager恢复机制实现类 -->

<name>yarn.resourcemanager.store.class</name>

<value>org.apache.hadoop.yarn.server.resourcemanager.recovery.ZKRMStateStore</value>

</property>

<property>

<!--指定主resourcemanager的地址-->

<name>yarn.resourcemanager.hostname</name>

<value>hadoop3</value>

</property>

<property>

<!--NodeManager获取数据的方式-->

<name>yarn.nodemanager.aux-services</name>

<value>mapreduce\_shuffle</value>

</property>

<property>

<!--开启日志聚集功能-->

<name>yarn.log-aggregation-enable</name>

<value>true</value>

</property>

<property>

<!--配置日志保留7天-->

<name>yarn.log-aggregation.retain-seconds</name>

<value>604800</value>

</property>

</configuration>

**4.打包hadoop文件发送到hadoop2、hadoop3**

cd /usr/local #进入hadoop目录

sudo tar –zcf /home/hadoop/hadoop.master.tar.gz hadoop

cd #回到有压缩包的路径下

scp hadoop.master.tar.gz hadoop@hadoop2:/home/hadoop/

scp hadoop.master.tar.gz hadoop@hadoop3:/home/hadoop/

在hadoop2上ls查看压缩包，解压到/usr/local/目录下



sudo tar –zxf hadoop.master.tar.gz –C /usr/local/

在hadoop3上重复解压步骤

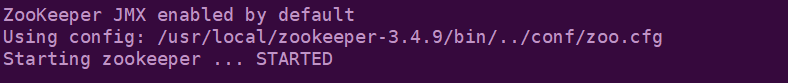
## 启动集群（初始化工作）

**1.启动三个zookeeper：**

hadoop@hadoop1:~$ zkServer.sh start

hadoop@hadoop2:~$ zkServer.sh start

hadoop@hadoop3:~$ zkServer.sh start



**2.启动3个JournalNode**

hadoop@hadoop1:~$ hadoop-daemon.sh start journalnode

hadoop@hadoop2:~$ hadoop-daemon.sh start journalnode

hadoop@hadoop3:~$ hadoop-daemon.sh start journalnode



**3.格式化NameNode(仅hadoop1)**

hadoop@hadoop1:~$ hdfs namenode –format

**4.复制hadoop1上的NameNode的元数据到hadoop2**

**在**/usr/local/hadoop/dfs/namenode\_data/路径下发送current/到hadoop2当前路径

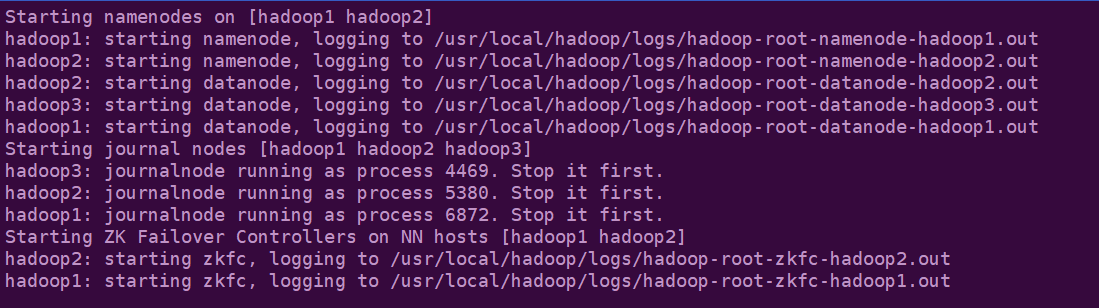
hadoop@hadoop1: /usr/local/hadoop/dfs/namenode\_data/scp -r current/ hadoop@hadoop2:$PWD

**5.** **在NameNode节点hadoop1格式化zkfc**

hadoop@hadoop1:~$hdfs zkfc –formatZK

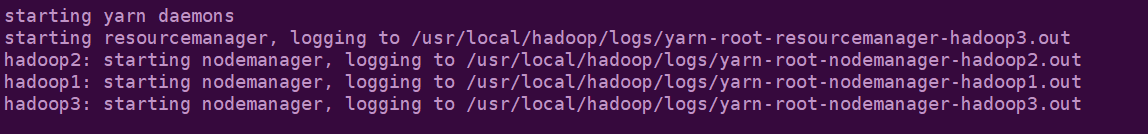
**6. 在hadoop1上启动HDFS相关服务**

hadoop@hadoop1:~$ start-dfs.sh



**7. 在hadoop3上启动YARN相关服务**

hadoop@hadoop3:~$ start-yarn.sh



**8. 最后单独启动hadoop1的历史任务服务器和hadoop2的ResourceManager**

hadoop@hadoop1:~$ mr-jobhistory-daemon.sh start historyserver



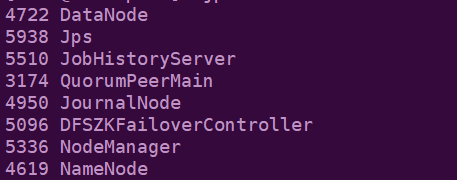
hadoop@hadoop2:~$ yarn-daemon.sh start resourcemanager



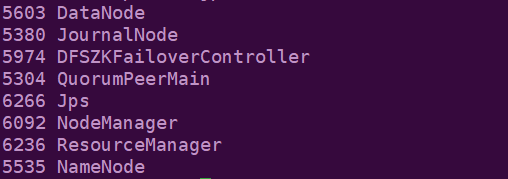
* **查看集群**

**1、jps查看:**

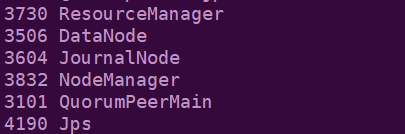
hadoop@hadoop1:~$ jps



hadoop@hadoop2:~$ jps



hadoop@hadoop3:~$ jps



查看NameNode的状态

hadoop@hadoop1:~$ hdfs haadmin -getServiceState nn1

standby

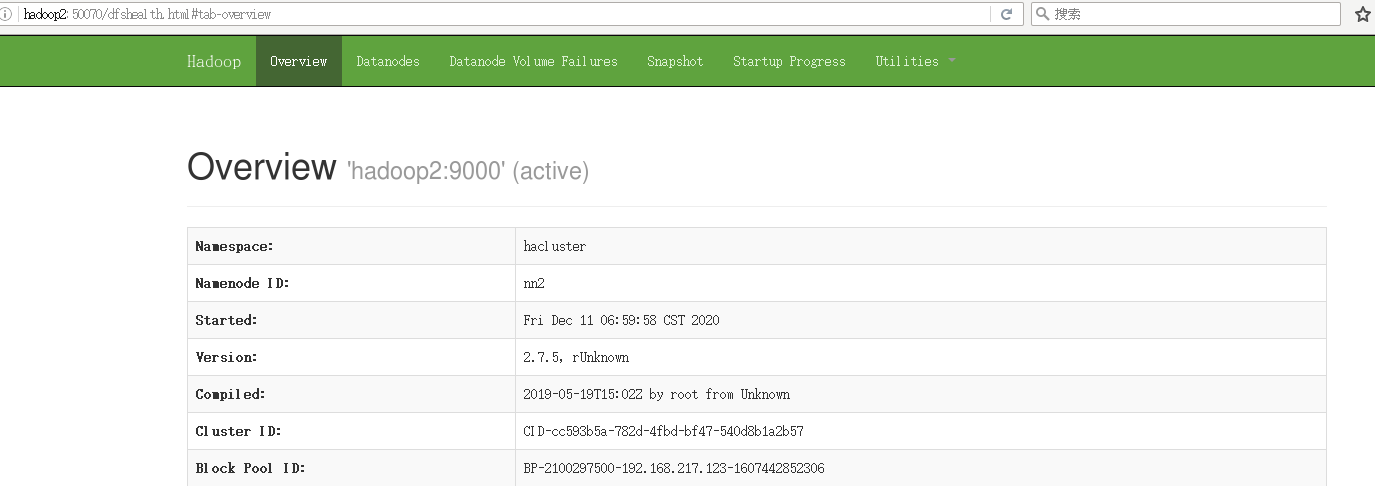
hadoop@hadoop1:~$ hdfs haadmin -getServiceState nn2

active

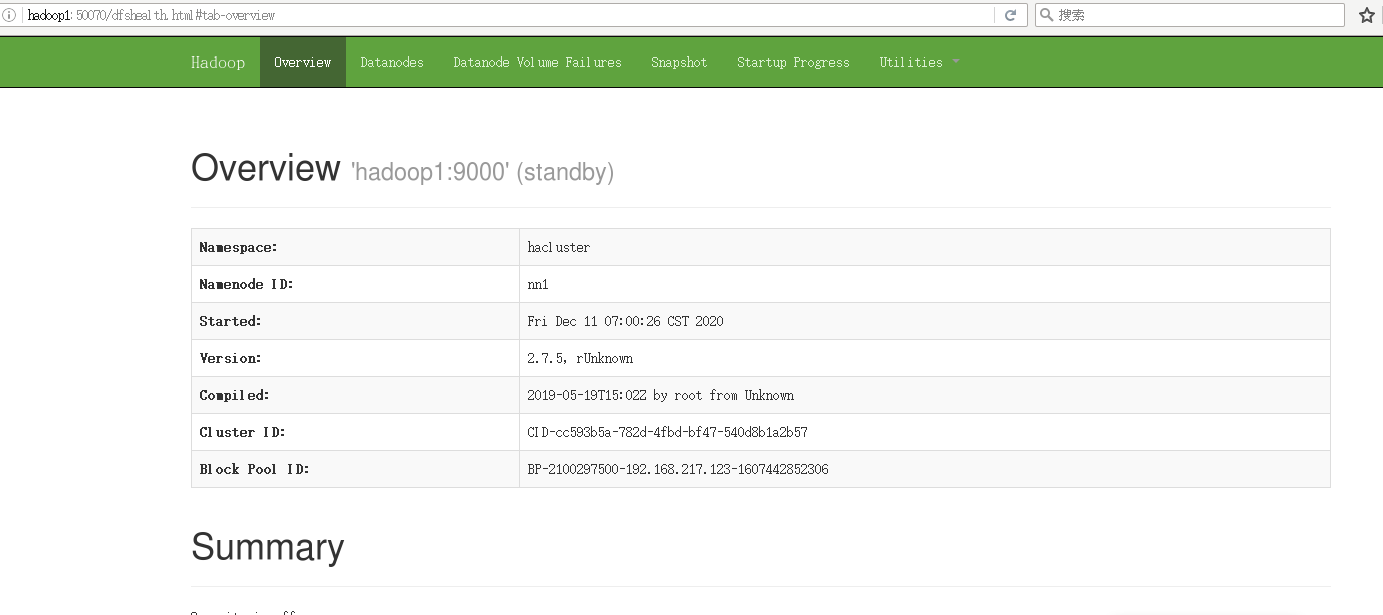
**2、Web查看:**

HDFS 和 YARN 的端口号分别为 50070 和 8088，界面应该如下：

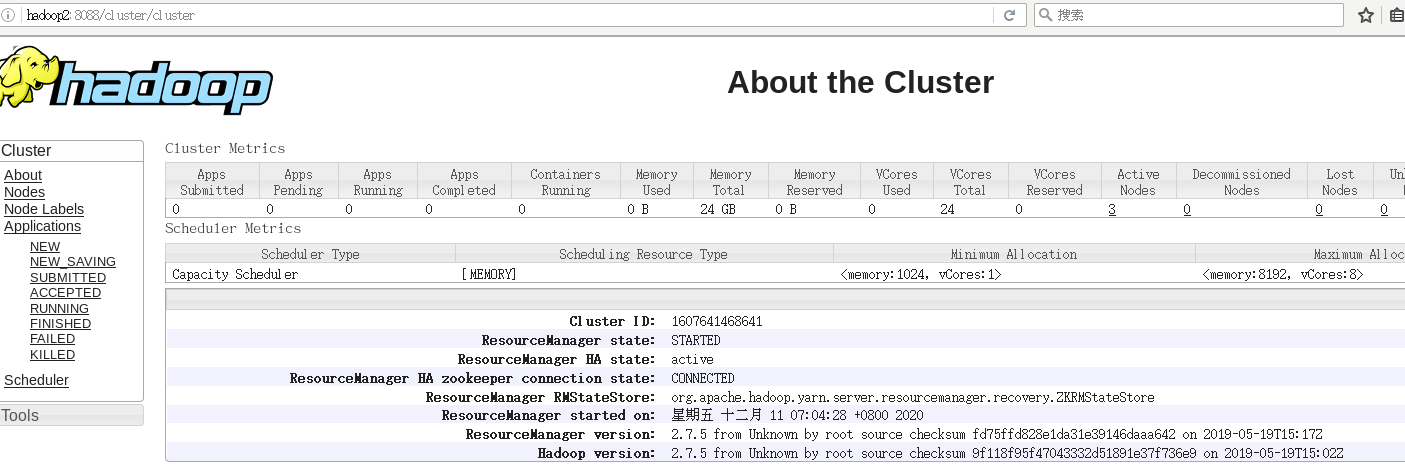
此时 hadoop2 上的 NameNode 处于可用(active)状态：



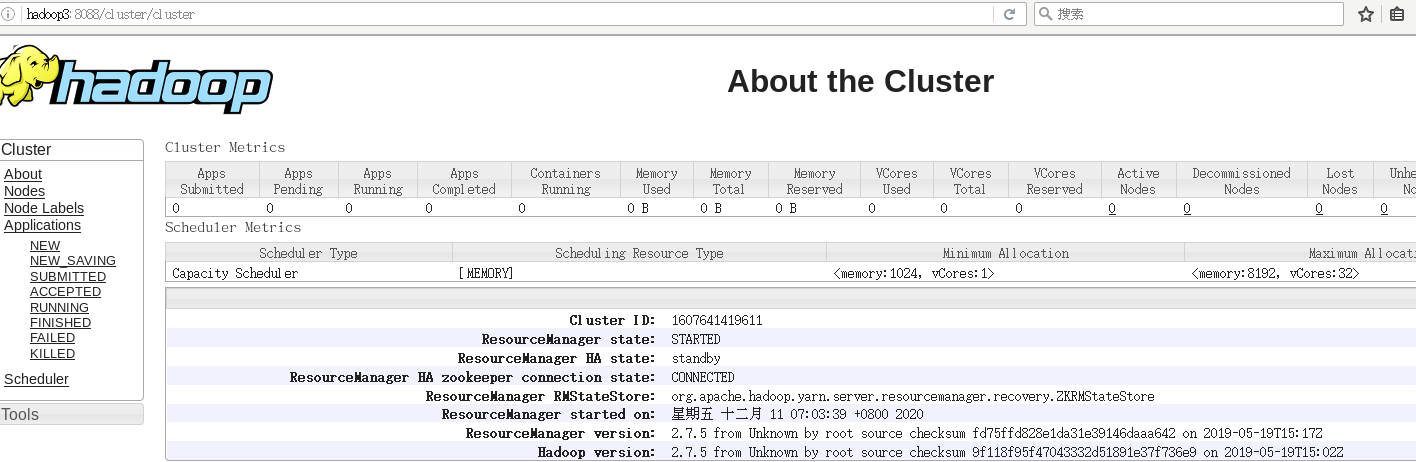
Hadoop1上的Namenode处于备用(standby)状态：



Hadoop2 上的 ResourceManager 处于可用(active)状态：



Hadoop3上的 ResourceManager 处于备用(standby)状态：

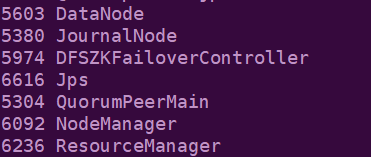


**3、HA高可用测试：**

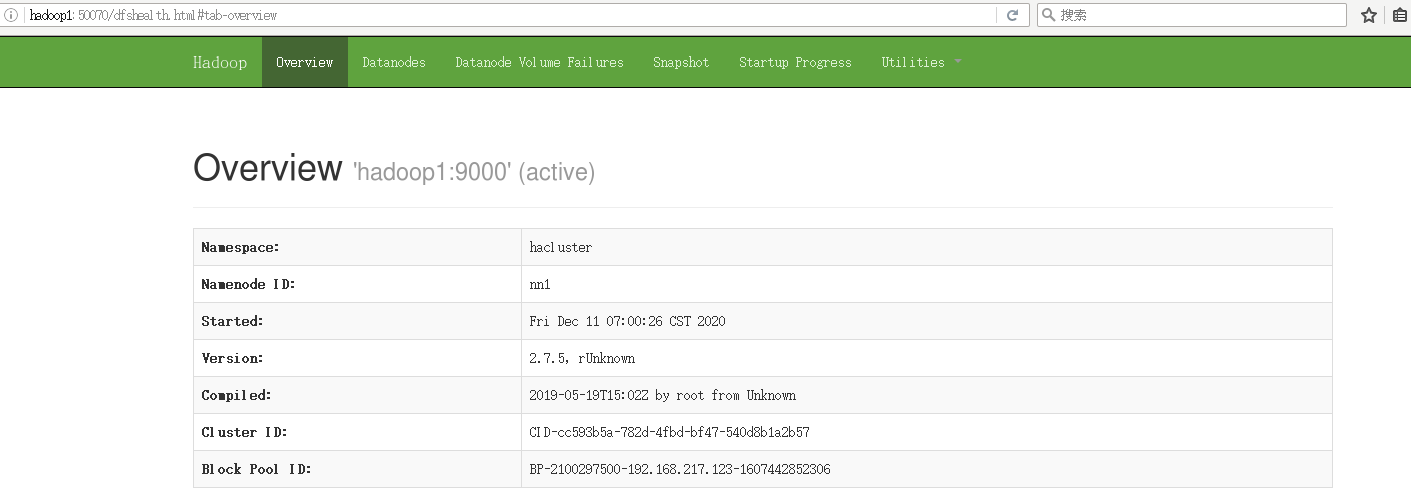
目前hadoop2上的namenode是可用状态，hadoop1是备用状态，现在使用命令kill -9 5535终止hadoop2的namenode进程，看看hadoop1是否有变化.

hadoop@hadoop2:~$ kill -9 5535

hadoop@hadoop2:~$ jps #查看没有namenode



刷新hadoop1的页面结果：



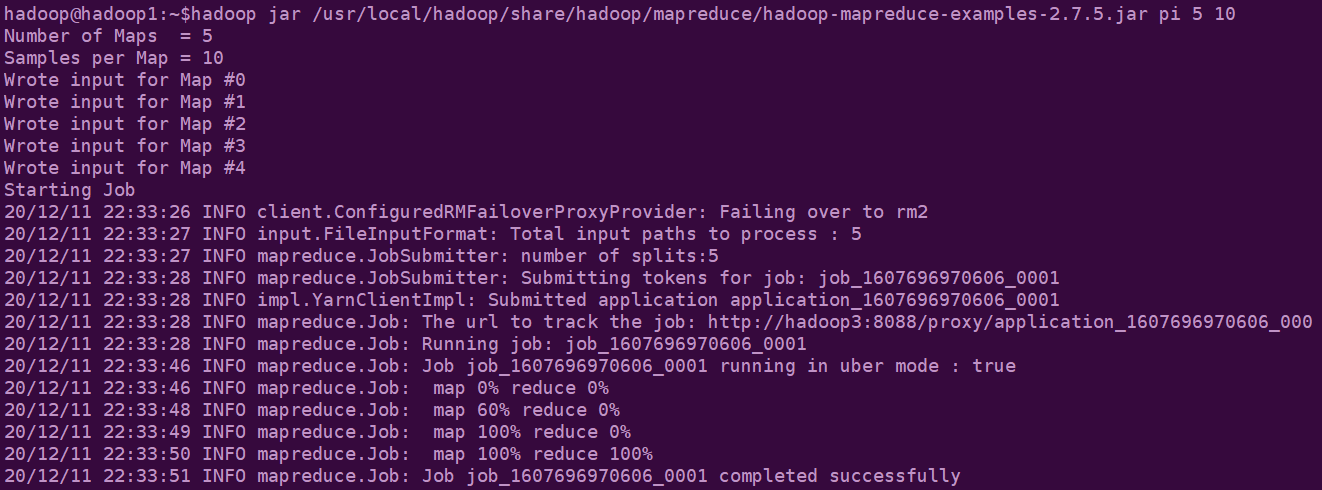
此时hadoop1的状态变成了active状态，hadoop1顶替了hadoop2的工作，HDFS的高可用就搭建成功了。

* **用自带案例运行测试**

**1、使用如下命令测试**：

hadoop jar /usr/local/hadoop/share/hadoop/mapreduce/hadoop-mapreduce-examples-2.7.5.jar pi 5 10

此命令是hadoop自带的用来求圆周率，pi是类名，5表示Map次数，10表示随机生成点的次数。

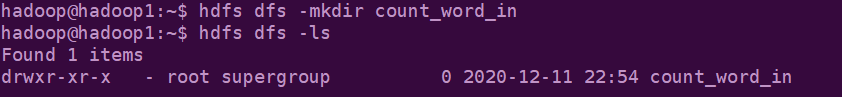


运行结果：



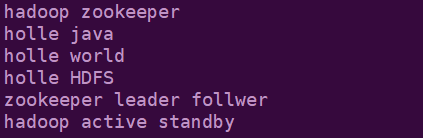
**2、单词计数测试：**

1、hadoop自带的wordcount例子，这个是统计单词个数的，首先要在hdfs系统中创建文件夹word\_count\_in，要查看hdfs系统可以通过hadoop fs -ls来查看hdfs系统的文件以及目录情况，如下图所示：

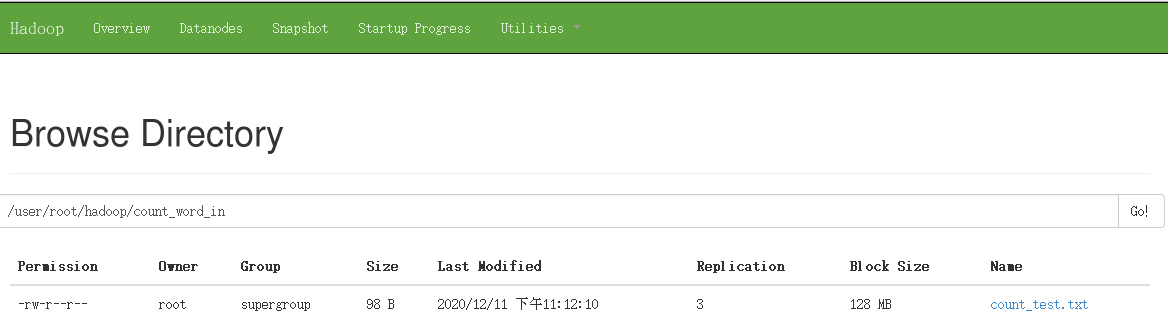


2、在本地创建文件count\_test.txt并写入单词后保存上传到HDFS:

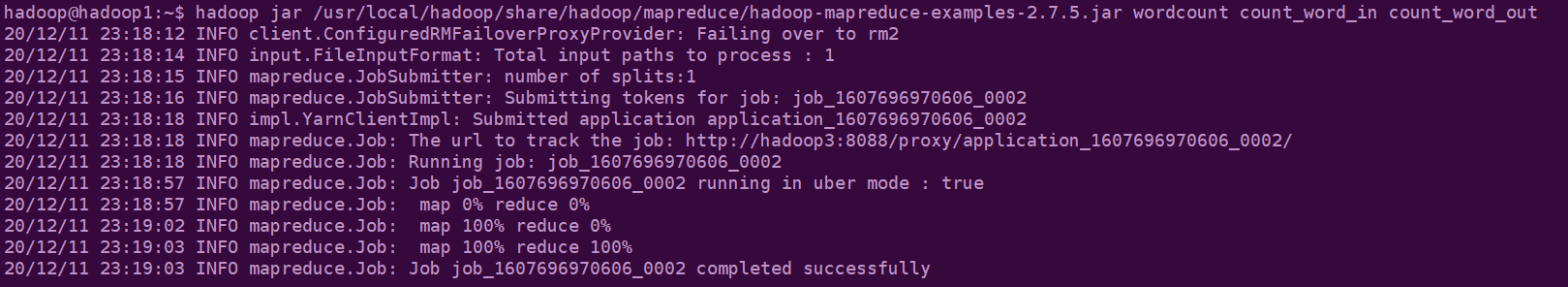
hadoop@hadoop1:~$vim count\_test.txt



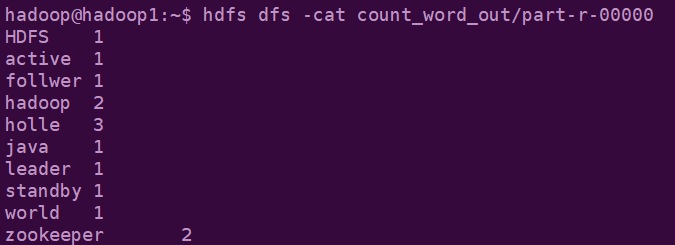
hadoop@hadoop1:~$ hdfs dfs -put count\_test.txt count\_word\_in



3、使用如下命令运行coun\_word\_in，count\_word\_out是输出文件名



打开count\_word\_out文件夹里面的part-r-00000文件就是单词统计结果。如下图所示：



统计结果和刚才输入的一致，单词计数测试成功！

## 项目结果分析和总结

本次项目是以成功安装Ubuntu系统为基础，搭建网络环境为保障，从最先安装和配置jdk开始，安装hadoop单机模式，到伪分布式，再到完全分布式，最后到实现HDFS高可用。整个过程难度是逐层递增的。

网络搭建时出现过的问题就是网线断裂，导致连接不上，后面搭建高可用时用的都是无线连接，做小项目有线连接和无线连接其实区别不大。在配置SSH免密登录时，印象最深的还是我们可以远程操控其他主机，并使用scp传送文件，这是本次项目用的最多，也是最实用的地方。

由于之前做过伪分布式的安装，在修改hadoop的环境变量和配置文件变得顺畅很多。尽管是这样，还是避免不了错误的发生，因为我们是在一个节点上把hadoop文件配置好后逐个发送到其他节点，所以减少了很多工作量，正是因为这样，一个节点出错全部出错，在配置四个文件时，一定不能大意，需要注意的是scp命令对于权限问题的要求，权限不够发送不了，考虑到工作量和安全问题，建议还是将hadoop整个文件打包后发送到其他节点，在其他节点解压到所需路径即可。

完全分布式搭建很顺利的搭建成功了。我们又把目标放到了HDFS高可用，HA（High Availability）高可用的搭建就离不开Zookeeper了，Zookeeper集群是为主备切换控制器提供主备选举支持。主备切换控制器 ZKFailoverController：ZKFailoverController 作为独立的进程运行，对 NameNode 的主备切换进行总体控制。ZKFailoverController 能及时检测到 NameNode 的健康状况，在主 NameNode 故障时借助 Zookeeper 实现自动的主备选举和切换。首先是先在一个节点上安装zookeeper并修改配置文件和环境变量，运行成功后再打包发送到其他节点。我们小组是从搭建完全分布式用的五个节点，一个NameNode节点，四个DataNode节点。因为是第一次搭建高可用，按照书本上的教程是三个节点，为确保万无一失所以我们把节点也改为三个，这是本次项目不足的地方。Zookeeper安装完后，hadoop的配置文件也要重新修改，需要做的是一定要删除之前完全分布式格式化时产生的日志文件，否则高可用格式化时肯定会出错。所有都准备就绪，接下来就是启动高可用集群。启动高可用集群必须先启动zookeeper的QuorumPeerMain进程，另外格式化后把主NameNode的元数据复制到备NameNode节点上，这是和完全分布式启动时区别最大的地方。

毫不夸张地说，当前网络信息化时代已经是大数据的时代，在大量的数据信息中，人们能够通过正确利用这些巨量数据而方便自己的生活，提高生活质量。随着大数据的快速发展，未来五年内，数据量和数据类型都会快速增长，那时，PB级的数据量已经不能被称为大数据了。随着数据的猛增，企业对数据分析和存储能力的需求必然大幅上升。Hadoop可以在商业服务器集群上分布数据和执行并行处理。Hadoop除了具备低成本分布式硬件和数据冗余安全模式之外，还拥有并行处理海量数据的能力。可通过添加节点轻松扩展Hadoop系统的可扩展性等等。我相信Hadoop系统将继续保持作为大数据处理领域的主流技术和平台的地位。