hadoop学习总结

## 一、版本

hadoop 2.6

## 二、VMWare虚拟机及配置

1、安装虚拟机的时候，最好有一个iso文件。

2、使用VMWare虚拟机的时候，需要保证网络适配器VMnet1和VMnet8没有禁用。

3、VMWare虚拟机的网络连接方式：

(1)桥接，指的是虚拟机使用宿主机所在的网络中的ip。

二者在同一个网段，可以互相ping通。

与VMnet1和VMnet8无关。

优点：虚拟机和外网的IP互相通讯

缺点：不安全

(2)仅主机，指的是虚拟机和宿主机使用专门的网段。

该网段无法访问外网。

与VMnet1有关。

优点：只有宿主机可以和虚拟机通讯，安全

缺点：无法访问外网IP

(3)NAT，指的是虚拟机和宿主机使用专门的网段，但是共享宿主机的ip地址。

该虚拟机可以访问外网。

与VMnet8有关。

优点：虚拟机可以访问外网IP，但是外网IP无法访问虚拟机，有足够的灵活性

|  |
| --- |
|  |

4、VMware修改NAT的网关ip

|  |
| --- |
|  |

设置linux的固定IP时，网关和DNS服务器都使用NAT的网关IP

|  |
| --- |
|  |

## 三、基础入门

LANP ：linux+apache+ngnix+php

### 1、云计算、大数据、人工智能

|  |
| --- |
|  |

云计算的三层次：Iaas、Paas、Saas

大数据属于Pass、Saas

### 2、硬盘读写数据的过程

硬盘读取数据时，读写磁头沿径向移动，移到要读取的扇区所在磁道的上方，这段

时间称为寻道时间(seek time)。因读写磁头的起始位置与目标位置之间的距离不同，寻

道时间也不同。目前硬盘一般为2到30毫秒，平均约为9毫秒。磁头到达指定磁道后，

然后通过盘片的旋转，使得要读取的扇区转到读写磁头的下方，这段时间称为旋转延

迟时间(rotational latencytime)。

一个7200（转/每分钟）的硬盘，每旋转一周所需时间为60×1000÷7200=8.33毫秒，

则平均旋转延迟时间为8.33÷2=4.17毫秒（平均情况下，需要旋转半圈）。平均寻道时

间和平均选装延迟称为平均存取时间。

所以，最后看一下硬盘的容量计算公式：

硬盘容量=盘面数×柱面数×扇区数×512字节

|  |
| --- |
|  |

数据存储硬件有：磁带、软盘、光盘、机械硬盘、固态硬盘SSD

### 3、计算机存储结构

计算机的基本组成:

• 存储器:实现记忆功能的部件用来存放计算程序及参与运算的各种数据

• 运算器:负责数据的算术运算和逻辑运算即数据的加工处理

• 控制器:负责对程序规定的控制信息进行分析,控制并协调输入,输出操作或内存访问

• 输入设备:实现计算程序和原始数据的输入

• 输出设备:实现计算结果输出

|  |
| --- |
|  |

从速度之殇可以看出：

如果一个大数据计算框架，需要很多的依赖磁盘IO，那么该框架的计算速度一定慢。

如果一个大数据计算框架，需要更多的依赖内存IO，那么该框架的计算速度一定快。

硬盘IO的读写比网络IO的读写速度快，如果计算的时候，需要数据大量的在网络间传播，那么速度一定慢，所以在分布式计算的时候，尽量减少网络间的数据传输。

### 4、计算机网络

|  |
| --- |
|  |

光钎一定比双绞线速度快

## 四、分布式入门

分布式指的是多台机器组网共同完成任务。分布式有分布式存储和分布式计算。分门别类的分层次管理，管理的是数据，分的门、别的类，都是被管理对象的元数据，元数据是数据的数据，数据的标识。

分布式管理的时候一定有元数据，每个节点都会有元数据，所以要有统一的元数据，这统一的元数据，一定是分布式、跨多台节点的。

### 1、数据的分布式存储

1.1 产生的原因，是数据量太大，只能分散存储在多个节点上

1.2 在管理的时候，需要客户端知晓文件到底在哪个节点上，管理不方便

1.3 要求出现一个统一的管理系统，包括文件的路径、名称、大小、修改时间等等。

这个统一的管理系统，包括各个节点上的数据的元数据信息。

1.4 分布式的存储系统，一般包括两类节点：

一类节点是存储元数据信息的，称作主节点。

一类节点是存储真实的数据的，称作从节点。

1.5 当客户端读写数据的时候，需要先和主节点打交道，因为需要知道数据到底在哪个从节点上。当知道数据在哪个从节点上后，就可以直接访问从节点，从而读取数据。

1.6 当海量的数据存储的时候，为了对文件管理上的方便，我们把文件按照固定大小进行分割，每一份称作一个块(block)。

1.7 数据存储在网络上，网络不够稳定。为了保证数据都可以随时访问，就需要把数据复制几份（一般复制3份即可），分别存储在不同的节点上。

|  |
| --- |
|  |

如果元数据所在的节点丢失了，那么真实的数据会不会丢失呢？

答案：不会，真实的数据不好查找了，就像没有了目录的书本

### 2、分布式存储过程

* 在分布式存储系统中，分散在不同节点中的数据可能属于同一个**文件**，为了组织众多的文件，把文件可以放到不同的**文件夹**中，文件夹可以一级一级的包含。我们把这种组织形式称为 **命名空间（namespace）**。命名空间管理着整个服务器集群中的所有文件。
* 集群中不同的 **节点**承担不同的职责。负责命名空间职责的节点称为 **主节点（master node）**，负责存储真实数据职责的节点称为 **从节点（slave node）**。主节点负责管理文件系统的文件结构，从节点负责存储真实的数据，称为**主从式结构（master-slaves）**。用户操作时，应该先和主节点打交道，查询数据在哪些从节点上存储，然后再从从节点读取。在主节点，为了加快用户访问的速度，会把整个命名空间信息都放在内存中，当存储的文件越多时，那么主节点就需要越多的内存空间。在从节点存储数据时，有的原始数据文件可能很大，有的可能很小，大小不一的文件不容易管理，那么可以抽象出一个独立的存储文件单位，称为 **块（block）**。数据存放在集群中，可能因为网络原因或者节点硬件原因造成访问失败，最好采用 **副本（replication）**机制，把数据同时备份到多台节点中，这样数据就安全了，数据丢失或者访问失败的概率就小了。

### 3、分布式计算

1.1 一个场景：Mysql服务器、apache/tomcat服务器。当客户端读取1GB的数据时，那么时间会非常慢。为什么慢？数据从mysql传输到tomcat很慢，从tomcat传输到客户端很慢。

当传输的数据量大的时候，耗费的时间主要在IO上。这种情况，称作“移动数据”。

解决之道就是把apache/tomcat上的程序代码移动到数据所在的节点上，这样就避免了IO了。提高传输效率。这种情况，叫做“移动计算”，移动的是程序代码。

总结一下，在处理大数据量的情况下， 建议移动计算，而不是移动数据。

1.2 移动计算情况下，进一步思考。如果计算程序是统计所有文件中单词的出现次数。

那么在各个节点上，需要分别统计本节点的单词次数；最后有一个程序进行汇总。各个节点计算的单词次数传输到汇总节点上即可。

这种情况下，网络IO极大的降低，提高了效率。

但是，采用这种分布式计算的方法，会对原来的程序产生影响。由集中的代码改为分步的代码。分步的代码是指分步骤执行的代码。

|  |
| --- |
|  |

## 五、hadoop概述

大数据产品栈，底层是存储(HDFS、Oracle、MySQL、redis)，中间做计算(MR\hive\storm\es\solr cloud\mahout等)，上层做应用(如图表echarts、D3等)

Hadoop特点：

1. 扩容能力（Scalable）：能可靠地（reliable）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千个字节。
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，这使得处理分成的快速。
4. 可靠性（Reliable）：hadoop能自动地维护数据的多份副本，并且在任务失败后 自动重新部署（redeploy）计算任务。

### 1、HDFS2的架构

* 负责数据的分布式存储
* 主从结构
* 主节点，可以有2个：**namenode**
* 从节点，可以有很多个：**datanode**
* **Namenode负责：**负责接收操作请求，是用户操作的入口。维护文件系统的目录结构，称作为命名空间
* **Datanode负责：**负责存储文件

### 2、Yarn的架构

* 资源的调度和管理平台
* 资源是指带宽、CPU、内存、存储
* 主从结构
* 主节点，可以有2个: **ResourceManager**
* 从节点，有很多个: **NodeManager**
* **ResourceManager负责**：
* 集群资源的分配与调度
* MapReduce、Storm、Spark等应用，必须调用ApplicationMasterProtocol接

口，才能被RM管理

* **NodeManager负责**：单节点资源的管理

### 3、MapReduce的架构

* 依赖磁盘 io 的批处理计算模型
* 主从结构
* 主节点，只有一个: **MRAppMaster**
* **MRAppMaster负责**：
* 接收客户提交的计算任务
* 把计算任务分给NodeManager的Container中执行，即任务调度
* 监控Container中Task的执行情况
* **Task负责**：执行任务

## 六、hadoop伪分布搭建

### 1、单节点的物理结构

|  |
| --- |
|  |

在区分集群中的主节点、从节点的时候，是根据不用的java进程区分的。

### 2、hadoop集群的物理分布

|  |
| --- |
|  |

问题：为什么NodeManage、DataNode要放在同一个节点上？

回答：因为是移动计算而不是移动数据，存储节点DataNode和计算节点NodeManager位于同一个节点上，节省数据传输的带宽。

### 3、安装hadoop

#### 3.1 安装前的准备工作

##### 3.1.1 设置静态ip，略

##### 3.1.2 设置主机名，修改文件/etc/sysconfig/network

|  |
| --- |
|  |

##### 3.1.3 让ip绑定主机名，修改文件/etc/hosts

|  |
| --- |
|  |

##### 3.1.4 关闭防火墙

防火墙是linux系统的一个service服务。

查看防火墙状态：service iptables status，

关闭防火墙：service iptables stop

关闭防火墙自动启动：chkconfig iptables off

##### 3.1.5 ssh免密码登录

ssh ： secure shell

免密码登录并不是代表不使用密码，是通过制作密钥的方式是linux系统的登录。

启动hadoop集群的时候，主节点会启动从节点上的相应进程。因为从节点很多，总是输入密码，会很麻烦。因此需要免密码登录。

免密码登录设置过程：

1. 产生密钥，执行命令ssh-keygen -t rsa ，一路按回车键，产生的密钥位于~/.ssh目录下
2. 把公钥id\_rsa.pub复制给目标节点，执行命令ssh-copy-id -i xxx ，xxx为目标节点的主机名或者ip地址

免密码登录测试如下：

|  |
| --- |
|  |

##### 3.1.6 JDK的安装

1. 卸载本地openJDK
2. 执行命令rpm -qa|grep jdk 查询openJDK

|  |
| --- |
| [root@mini01 dupenghui]# rpm -qa|grep jdk  java-1.7.0-openjdk-1.7.0.9-2.3.4.1.el6\_3.i686  java-1.6.0-openjdk-1.6.0.0-1.50.1.11.5.el6\_3.i686 |

1. 把openJDK卸载，执行以下命令

|  |
| --- |
| rpm -e --nodeps java-1.7.0-openjdk-1.7.0.9-2.3.4.1.el6\_3.i686  rpm -e --nodeps java-1.6.0-openjdk-1.6.0.0-1.50.1.11.5.el6\_3.i68 |

1. 解压jdk-8u91-linux-x64.tar.gz文件，执行命令tar –zxvf jdk-8u91-linux-x64.tar.gz
2. 设置环境vi /etc/profile 增加JDK的环境变量：

export JAVA\_HOME=/opt/jdk1.8.0\_91

export PATH=.:$JAVA\_HOME/bin: $PATH

1. 保存后执行命令：source /etc/profile，使环境变量生效
2. 执行java –version测试安装是否成功

java version "1.8.0\_91"

Java(TM) SE Runtime Environment (build 1.8.0\_91-b14)

Java HotSpot(TM) 64-Bit Server VM (build 25.91-b14, mixed mode)

#### 3.2 安装及配置hadoop

##### 3.2.1 解压hadoop-2.6.0.tar.gz文件

执行命令tar -zxvf hadoop-2.6.0.tar.gz

##### 3.2.2 设置环境vi /etc/profile 增加hadoop的环境变量：

export HADOOP\_HOME=/opt/hadoop-2.6.0

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

##### 3.2.3 保存后执行命令：source /etc/profile，使环境变量生效

##### 3.2.4 修改hadoop下的配置文件

配置文件在${HADOOP\_HOME}/etc/hadoop目录下

###### 3.2.4.1 hadoop-env.sh

执行命令 vi hadoop-env.sh，在最后一行增加jdk和hadoop日志的环境变量：

# jdk home

export JAVA\_HOME=/opt/jdk1.8.0\_91

# hadoop log

export HADOOP\_LOG\_DIR=/opt/hadoop\_repo/logs

###### 3.2.4.2 slaves

从节点，配置的是主机名

|  |
| --- |
|  |

###### 3.2.4.3 yarn-site.xml

crxy118 是主机名

|  |
| --- |
| <configuration>  <!-- Site specific YARN configuration properties -->  <!--  yarn.nodemanager.aux-servces的默认值为mapreduce.shuffle,在启动NodeManager时会失败，报  "java.lang.IllegalArgumentException: The ServiceName: mapreduce.shuffle set in  yarn.nodemanager.aux-services is invalid.The valid service name should only contain a-zA-Z0-9\_  and can not start with numbers"异常。  解决办法就是：把yarn.nodemanager.aux-servces的值改为mapreduce\_shuffle  -->  <property>  <name>yarn.nodemanager.aux-servces</name>  <value>mapreduce\_shuffle</value>  </property>  <!--  日志聚集是YARN提供的日志中央化管理功能，  它能将运行完成的Container/任务日志上传到HDFS上，  从而减轻NodeManager负载，且提供一个中央化存储和分析机制。  yarn.log-aggregation-enable是否启用日志聚集功能，默认为false  -->  <property>  <name>yarn.log-aggregation-enable</name>  <value>true</value>  </property>  <!-- resourcemanager的hostname -->  <property>  <name>yarn.resourcemanager.hostname</name>  <value>crxy118</value>  </property>  <!--  resourcemanager对客户端暴露的地址，  客户端通过该地址向resourcemanager提交应用程序等  -->  <property>  <name>yarn.resourcemanager.address</name>  <value>crxy118:8032</value>  </property>  <!--  resourcemanager对ApplicationMaster暴露的地址，  ApplicationMaster通过地址向resourcemanager申请资源，释放资源等  -->  <property>  <name>yarn.resourcemanager.scheduler.address</name>  <value>crxy118:8030</value>  </property>  <!--  resourcemanager对nodemanager暴露地址，  nodemanager通过该地址向resourcemanager汇报心跳，领取任务等  -->  <property>  <name>yarn.resourcemanager.resource-tracker.address</name>  <value>crxy118:8031</value>  </property>  <!--  管理员可以通过该地址向resourcemanager发送管理命令等  -->  <property>  <name>yarn.resourcemanager.admin.address</name>  <value>crxy118:8033</value>  </property>  <!--  resourcemanager对外暴露的web http地址，  用户可通过该地址在浏览器中查看集群信息  -->  <property>  <name>yarn.resourcemanager.webapp.address</name>  <value>crxy118:8088</value>  </property>  </configuration> |

Hadoop yarn 参数配置说明：

**RM** ：ResourceManager

**AM** ：ApplicationMaster

**NM** ：NodeManager

|  |  |  |
| --- | --- | --- |
| **参数** | **默认值** | **描述** |
| **yarn.resourcemanager.hostname** |  | RM的hostname |
| **yarn.resourcemanager.address** | ${yarn.resourcemanager.hostname}:8032 | RM对客户端暴露的地址，客户端通过该地址向RM提交应用程序等 |
| **yarn.resourcemanager.scheduler.address** | ${yarn.resourcemanager.hostname}:8030 | RM对AM暴露的地址，AM通过地址想RM申请资源，释放资源等 |
| **yarn.resourcemanager.webapp.address** | ${yarn.resourcemanager.hostname}:8088 | RM对外暴露的web  http地址，用户可通过该地址在浏览器中查看集群信息 |
| **yarn.resourcemanager.webapp.https.address** | ${yarn.resourcemanager.hostname}:8090 | web https 地址 |
| **yarn.resourcemanager.resource-tracker.address** | ${yarn.resourcemanager.hostname}:8031 | RM对NM暴露地址，NM通过该地址向RM汇报心跳，领取任务等 |
| **yarn.resourcemanager.resource-tracker.client.thread-count** | 50 | 处理来自NM的RPC请求的handler数 |
| **yarn.resourcemanager.admin.address** | ${yarn.resourcemanager.hostname}:8033 | 管理员可以通过该地址向RM发送管理命令等 |
| **yarn.resourcemanager.scheduler.class** | org.apache.hadoop.yarn.server.resourcemanager  .scheduler.capacity.CapacityScheduler | 资源调度器主类 |
| **yarn.resourcemanager.scheduler.client.thread-count** | 50 | 处理来自AM的RPC请求的handler数 |
| **yarn.scheduler.minimum-allocation-mb** | 1024 | 可申请的最少内存资源，以MB为单位 |
| **yarn.scheduler.maximum-allocation-mb** | 8192 | 可申请的最大内存资源，以MB为单位 |
| **yarn.scheduler.minimum-allocation-vcores** | 1 | 可申请的最小虚拟CPU个数 |
| **yarn.scheduler.maximum-allocation-vcores** | 32 | 可申请的最 大虚拟CPU个数 |
| **yarn.nodemanager.local-dirs** | ${hadoop.tmp.dir}/nm-local-dir | 中间结果存放位置，可配置多目录 |
| **yarn.log-aggregation-enable** | false | 是否启用日志聚合 |
| **yarn.nodemanager.remote-app-log-dir** | /tmp/logs | 日志聚合目录 |
| **yarn.nodemanager.resource.memory-mb** | 8192 | NM总的可用物理内存，以MB为单位。一旦设置，不可动态修改 |
| **yarn.nodemanager.resource.cpu-vcores** | 8 | 可分配的CPU个数 |
| **yarn.nodemanager.aux-services** |  | NodeManager上运行的附属服务。需配置成mapreduce\_shuffle，才可运行MapReduce程序 |

###### 3.2.4.4 mapred-site.xml

crxy118 是主机名

|  |
| --- |
| <configuration>  <!-- mapreduce.framework.name配置使用yarn来运行mapreduce程序 -->  <property>  <name>mapreduce.framework.name</name>  <value>yarn</value>  </property>  <!-- jobhistory server -->  <property>  <name>mapreduce.jobhistory.address</name>  <value>crxy118:10020</value>  </property>  <!-- jobhistory web http address -->  <property>  <name>mapreduce.jobhistory.webapp.address</name>  <value>crxy118:19888</value>  </property>  <property>  <name>yarn.app.mapreduce.am.staging-dir</name>  <value>/history</value>  </property>  <property>  <name>mapreduce.jobhistory.done-dir</name>  <value>${yarn.app.mapreduce.am.staging-dir}/history/done</value>  </property>  <property>  <name>mapreduce.jobhistory.intermediate-done-dir</name>  <value>${yarn.app.mapreduce.am.staging-dir}/history/done\_intermediate</value>  </property>  </configuration> |

###### 3.2.4.5 hdfs-site.xml

|  |
| --- |
| <configuration>  <!-- namenode所使用的元数据保存，一般建议在nfs上保留一份 -->  <property>  <name>dfs.namenode.name.dir</name>  <value>file:///opt/hadoop\_repo/name</value>  </property>  <!-- 真正的datanode数据保存路径，可以写多块硬盘，逗号分隔.  把这些位置分散在每个节点上的所有磁盘上可以实现磁盘 I/O 平衡，因此会显著改进磁盘 I/O 性能。 -->  <property>  <name>dfs.datanode.data.dir</name>  <value>file:///opt/hadoop\_repo/data</value>  </property>  <!-- secondary namenode 节点存储 checkpoint 文件目录 -->  <property>  <name>dfs.namenode.checkpoint.dir</name>  <value>file:///opt/hadoop\_repo/namesecondary</value>  </property>  <!-- hdfs数据块的复制份数，默认3，理论上份数越多跑数速度越快，但是需要的存储空间也更多。 -->  <property>  <name>dfs.replication</name>  <value>1</value>  </property>  <!-- 访问namenode的hdfs使用50070端口，访问datanode的webhdfs使用50075端口。  访问文件、文件夹信息使用namenode的IP和50070端口，访问文件内容或者进行打开、上传、修改、  下载等操作使用datanode的IP和50075端口。要想不区分端口，直接使用namenode的IP和端口进行所有的webhdfs操作，  就需要在所有的datanode上都设置hefs-site.xml中的dfs.webhdfs.enabled为true。 -->  <property>  <name>dfs.webhdfs.enabled</name>  <value>true</value>  </property>  <!-- 在HDFS中是否启用权限检查 -->  <property>  <name>dfs.permissions</name>  <value>false</value>  </property>  <!-- 指定datanode的最大数量的线程用于传输数据。默认 4096 -->  <property>  <name>dfs.datanode.max.transfer.threads</name>  <value>4096</value>  </property>  </configuration> |

###### 3.2.4.6 core-site.xml

crxy118 是主机名

|  |
| --- |
| <configuration>  <property>  <name>fs.defaultFS</name>  <value>hdfs://crxy118:9000</value>  </property>  <property>  <name>hadoop.tmp.dir</name>  <value>/opt/hadoop\_repo/tmp</value>  </property>  <!—回收站 -->  <!--  <property>  <name>fs.trash.interval</name>  <value>1440</value>  <description>  Number of minutes between trash checkpoints. If zero, the trash feature is disabled.  </description>  </property>  -->  </configuration> |

###### 3.2.4.7 yarn-env.sh

执行命令 vi yarn-env.sh，在最后一行增加jdk和yarn日志的环境变量：

# jdk home

export JAVA\_HOME=/opt/jdk1.8.0\_91

# yarn log

export YARN\_LOG\_DIR=/opt/hadoop\_repo/logs

#### 3.3 启动hadoop

##### 3.3.1 格式化文件系统

1. cd切换到HADOOP\_HOME目录
2. 执行格式化命令：bin/hdfs namenode –format

**注意：格式化命令只能执行一次，严禁执行多次，多次格式化会导致NameNode和DataNode的ClusterID不一致，HDSF集群启动失败。**

###### 3.3.1.1 比较NameNode和DataNode的ClusterID/VERSION

|  |
| --- |
| **NameNode** |
|  |
| **DataNode** |
|  |

##### 3.3.2 启动hdfs

执行命令sbin/start-dfs.sh

然后通过浏览器http://ip:50070访问hdfs

|  |
| --- |
|  |

##### 3.3.3 启动yarn

执行命令sbin/start-yarn.sh

然后通过浏览器http://ip:8088 访问yarn

|  |
| --- |
|  |

##### 3.3.4 启动historyserver

Jobhistoryserver 实现web查看作业的历史运行情况

启动执行命令sbin/mr-jobhistory-daemon.sh start historyserver

然后浏览器访问<http://ip:19888>

|  |
| --- |
|  |

## HDFS

### 1、HDFS SHELL 操作

#### 1.1、hadoop fs(hdfs dfs)文件操作

##### 1)ls 显示目录下的所有文件或者文件夹

使用方法： hdfs dfs -ls [uri形式目录]

示例: hdfs dfs –ls / 显示根目录下的所有文件和目录

显示目录下的所有文件（包含子级）可以加 -R 选项

示例: hdfs dfs -ls -R /

##### 2) cat 查看文件内容

使用方法：hdfs dfs -cat URI [URI …]

示例： hdfs dfs -cat /in/test2.txt

##### 3) mkdir 创建目录

使用方法：hdfs dfs -mkdir [uri形式目录]

示例: hdfs dfs –mkdir /test

创建多级目录 加上 –p

示例: hdfs dfs –mkdir -p /a/b/c

##### 4) rm 删除目录或者文件

使用方法:hadoop fs -rm [文件路径]   删除文件夹加上 -r

示例: hadoop fs -rm /test1.txt

删除文件夹加上 –r ，表示逐级删除全部文件

示例:hadoop fs -rm -r /test

##### 5) put 复制文件

将文件复制到hdfs系统中，也可以是从标准输入中读取文件，此时的dst是一个文件

使用方法: hadoop fs -put <localsrc> ... <dst>

示例：Hadoop fs -put /usr/wisedu/temp/test1.txt /

从标准输入中读取文件：hadoop fs -put -/in/myword

##### 6)cp复制系统内文件

使用方法：hadoopfs -cp URI [URI …] <dest>

将文件从源路径复制到目标路径。这个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：hadoop fs -cp /in/myword/word

##### 7)copyFromLocal 复制本地文件到hdfs

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](http://hadoop.apache.org/docs/r1.0.4/cn/hdfs_shell.html" \l "putlink" \t "_blank)**命令相似

##### 8)get 复制文件到本地系统

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/word /usr/wisedu/temp/word.txt

##### 9)copyToLocal 复制文件到本地系统

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](http://hadoop.apache.org/docs/r1.0.4/cn/hdfs_shell.html" \l "getlink" \t "_blank)命令类似。

示例：hadoop fs - copyToLocal/word /usr/wisedu/temp/word.txt

##### 10)mv

将文件从源路径移动到目标路径。这个命令允许有多个源路径，此时目标路径必须是一个目录。不允许在不同的文件系统间移动文件。

使用方法：hadoop fs -mv URI [URI …] <dest>

示例：hadoop fs -mv /in/test2.txt /test2.txt

##### 11)du 显示文件大小

显示目录中所有文件的大小。

使用方法：hadoop fs -du URI [URI …]

示例: hadoop fs -du /

显示当前目录或者文件夹的大小可加选项 -s

示例: hadoop fs -du -s /

##### 12)touchz  创建空文件

  使用方法：hadoop fs -touchz URI [URI …]

创建一个0字节的空文件

示例:hadoop fs -touchz /empty.txt

##### 13)chmod 改变文件权限

使用方法：hadoop fs -chmod[-R] <MODE[,MODE]... | OCTALMODE> URI [URI …]

与Linux平台下chmod命令相似，改变文件的权限。使用-R将使改变在目录结构下递归进行。命令的使用者必须是文件的所有者或者超级用户。

示例：先创建一个普通用户test:sudo useradd -m test

      再用wisedu用户在hdfs系统目录/a下创建hello.txt文件，此时test具有读取/a/hello.txt文件的权限.

      在切换回wisedu用户修改文件的权限，让/a目录下的文件对于其他用户都不可读，命令: hadoop fs -chmod -R o-r /a  如下图所示，再切换回test用户查看/a/hello.txt文件时提示没有权限.

##### 14)chown 改变文件所有者

使用方法：hadoop fs -chown [-R] [OWNER][:[GROUP]] URI [URI]

改变文件的拥有者。使用-R将使改变在目录结构下递归进行。命令的使用者必须是超级用户。

示例:hadoop fs -chown -R test /a

##### 15)chgrp 改变文件所在组

使用方法：hadoop fs -chgrp [-R] GROUP URI [URI …]

改变文件所属的组。使用-R将使改变在目录结构下递归进行。命令的使用者必须是文件的所有者或者超级用户。

示例：hadoop fs -chgrp -R test /a

#### 1.2、hdfs dfsadmin 管理命令

##### 1) -report

查看文件系统的基本信息和统计信息。

示例：hdfs dfsadmin -report

##### 2) -safemode

enter | leave | get | wait：安全模式命令。安全模式是NameNode的一种状态，在这种状态下，NameNode不接受对名字空间的更改（只读）；不复制或删除块。NameNode在启动时自动进入安全模式，当配置块的最小百分数满足最小副本数的条件时，会自动离开安全模式。enter是进入，leave是离开。

示例:hdfs dfsadmin -safemode get

hdfsdfsadmin -safemode enter

##### 3) -refreshNodes

重新读取hosts和exclude文件，使新的节点或需要退出集群的节点能够被NameNode重新识别。这个命令在新增节点或注销节点时用到。

示例：hdfs dfsadmin -refreshNodes

##### 4) -finalizeUpgrade

终结HDFS的升级操作。DataNode删除前一个版本的工作目录，之后NameNode也这样做。

##### 5) -upgradeProgress

status| details | force：请求当前系统的升级状态 | 升级状态的细节| 强制升级操作

##### 6) -metasave filename

 保存NameNode的主要数据结构到hadoop.log.dir属性指定的目录下的<filename>文件中。

##### 7) -setQuota<quota><dirname>……<dirname>

 为每个目录<dirname>设定配额<quota>。目录配额是一个长整形整数，强制设定目录树下的名字个数。

##### 8) -clrQuota<dirname>……<dirname>

为每个目录<dirname>清除配额设定。

##### 9) -help

显示帮助信息

### 2、HDFS的体系结构

|  |
| --- |
|  |

#### 2.1、NameNode

1）namenode是整个文件系统的管理节点。它维护着整个文件系统的文件目录树，文件/目录的元信息和每个文件对应的数据块列表。接收用户的操作请求。

2）数据非常的宝贵，不能丢失。

3）这些核心的元数据信息保存在hdfs-site.xml的dfs.namenode.name.dir配置项所指定的路径下。

fsimage\_xxxxx文件是存储静态的元数据信息的。

edits\_xxxxx文件存储的是正在处理的文件。

1. 在集群运行的时候，NameNode中的元数据信息都是保存在内存中的。所以NameNode节点的内存一定要大。
2. 如果有1GB的数据量，是分为1个文件存储好，还是1000个文件存储好？

答：1个文件好。因为1000个文件，每个文件大小不到1MB，会产生1000个block元数据信息了；而1个文件的话，会产生1024MB/128MB=8个block元数据信息，占有NameNode的内存就会少。

|  |
| --- |
| [root@crxy118 current]# pwd  /opt/hadoop\_repo/name/current  [root@crxy118 current]# more VERSION  #Fri Oct 13 21:44:31 CST 2017  namespaceID=61677345  clusterID=CID-3c3e19f7-332e-4a95-8ce9-4263b81eda09  cTime=0  storageType=NAME\_NODE  blockpoolID=BP-1570153413-192.168.21.118-1505139104742  layoutVersion=-60 |

#### 2.2、DataNode

1. 提供真实文件数据的存储服务
2. 这些真实的数据信息保存在hdfs-site.xml的dfs.datanode.data.dir配置项所指定的路径下。

|  |
| --- |
| [root@crxy118 current]# pwd  /opt/hadoop\_repo/data/current  [root@crxy118 current]# more VERSION  #Fri Oct 13 21:44:35 CST 2017  storageID=DS-d116dd05-04f3-4956-99ae-6209df404667  clusterID=CID-3c3e19f7-332e-4a95-8ce9-4263b81eda09  cTime=0  datanodeUuid=ffde4ed1-b018-4d6b-88af-e733f6fb36a1  storageType=DATA\_NODE  layoutVersion=-56 |

观察NameNode中的VERSION和DataNode中的VERSION的clusterID的值，发现是相同的。如果2个clusterID不相同，那么就不是同一个hdfs集群。

当多次执行hdfs namenode -format的时候，会改变NameNode的clusterID的值，但是不会改变DataNode的clusterID的值。这时，两个值就不相同，就不属于同一个hdfs集群。虽然能够看到2个进程都存在，但是不属于同一个hdfs集群。

1. block块管理

最基本的存储单位。对于文件内容而言，一个文件的长度大小是size，那么从文件的０偏移开始，按照固定的大小，顺序对文件进行划分并编号，划分好的每一个块称一个Block。HDFS默认Block大小由参数dfs.blocksize控制，默认值是128MB（在hadoop1中，默认大小是64MB）,可以小于该尺寸，但是不能超过该尺寸。

如果一个文件是150MB，那么就分为2个块，大小分别是128MB和22MB。

如果有2个大小都是10MB的文件，那么会存储为2个块，实际大小都是10MB。

1. 副本replica管理

多复本，由在hdfs-site.xml的参数dfs.replication控制，默认是3个，实际工作中可以改为2个。

Client端读取多副本文件的过程



### HDFS读文件过程（非常重要）

1.首先调用FileSystem对象的open方法，其实是一个DistributedFileSystem的实例

2.DistributedFileSystem通过rpc获得文件的第一个block的locations，同一block按照副本数会返回多个locations，这些locations按照hadoop拓扑结构排序，距离客户端近的排在前面.

3.前两步会返回一个FSDataInputStream对象，该对象会被封装成DFSInputStream对象，DFSInputStream可以方便的管理datanode和namenode数据流。客户端调用read方法，DFSInputStream最会找出离客户端最近的datanode并连接。

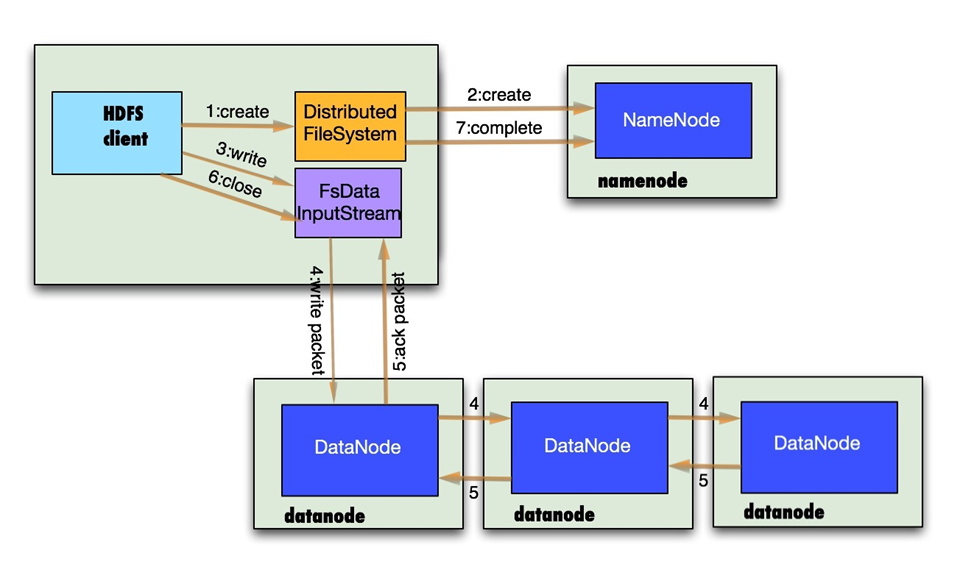
4.数据从datanode源源不断的流向客户端。

5.如果第一块的数据读完了，就会关闭指向第一块的datanode连接，接着读取下一块。这些操作对客户端来说是透明的，客户端的角度看来只是读一个持续不断的流。

6.如果第一批block都读完了，DFSInputStream就会去namenode拿下一批blocks的location，然后继续读，如果所有的块都读完，这时就会关闭掉所有的流。

如果在读数据的时候，DFSInputStream和datanode的通讯发生异常，就会尝试正在读的block的排第二近的datanode,并且会记录哪个datanode发生错误，剩余的blocks读的时候就会直接跳过该datanode。DFSInputStream也会检查block数据校验和，如果发现一个坏的block,就会先报告到namenode节点，然后DFSInputStream在其他的datanode上读该block的镜像

该设计的思想就是客户端直接连接datanode来读取数据，namenode负责为每一个block提供最优的datanode，namenode仅仅处理block location的请求，这些信息都加载在namenode的内存中。这样设计，可以让hdfs通过datanode集群可以承受大量客户端的并发访问。



### HDFS写文件过程（非常重要）

1.客户端通过调用DistributedFileSystem的create方法创建新文件

2.DistributedFileSystem通过RPC调用namenode去创建一个没有blocks关联的新文件，创建前，namenode会做各种校验，比如文件是否存在，客户端有无权限去创建等。如果校验通过，namenode就会记录下新文件，否则就会抛出IO异常.

3.前两步结束后会返回FSDataOutputStream的对象，象读文件的时候相似，FSDataOutputStream被封装成DFSOutputStream.DFSOutputStream可以协调namenode和datanode。客户端开始写数据到DFSOutputStream,DFSOutputStream会把数据切成一个个小packet，然后排成队列data quene。

4.DataStreamer会去处理接受data queue，他先问询namenode这个新的block最适合存储的在哪几个datanode里，比如副本数是3，那么就找到3个最适合的datanode，把他们排成一个pipeline.DataStreamer把packet按队列输出到管道的第一个datanode中，第一个datanode又把packet输出到第二个datanode中，以此类推。

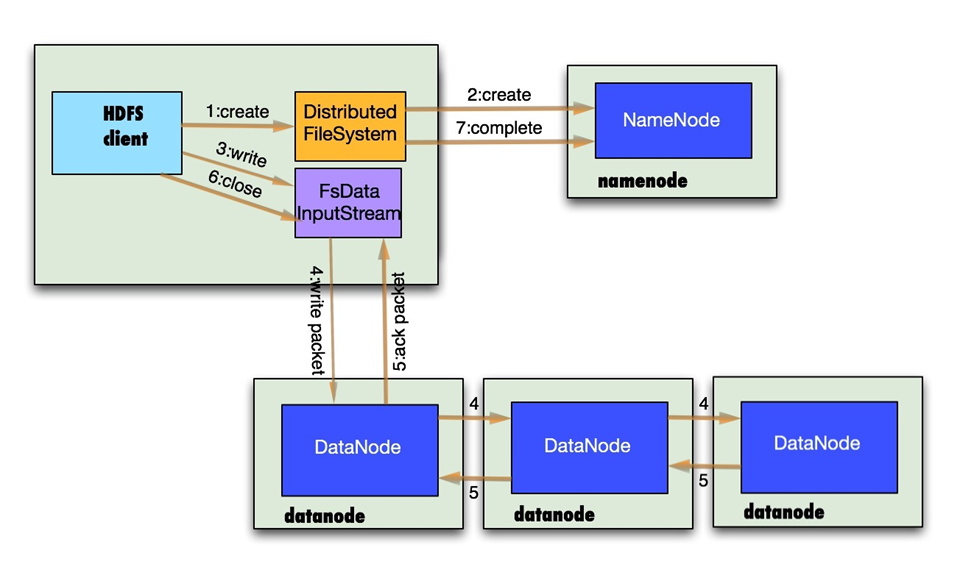
5.DFSOutputStream还有一个对列叫ack queue，也是有packet组成，等待datanode的收到响应，当pipeline中的所有datanode都表示已经收到的时候，这时akc queue才会把对应的packet包移除掉。

如果在写的过程中某个datanode发生错误，会采取以下几步：1) pipeline被关闭掉；2)为了防止丢包ack queue里的packet会同步到data queue里；3)把产生错误的datanode上当前在写但未完成的block删掉；4）block剩下的部分被写到剩下的两个正常的datanode中；5）namenode找到另外的datanode去创建这个块的复制。当然，这些操作对客户端来说是无感知的。

6.客户端完成写数据后调用close方法关闭写入流

7.DataStreamer把剩余得包都刷到pipeline里然后等待ack信息，收到最后一个ack后，通知datanode把文件标示为已完成。

另外要注意得一点，客户端执行write操作后，写完得block才是可见的，正在写的block对客户端是不可见的，只有调用sync方法，客户端才确保该文件被写操作已经全部完成，当客户端调用close方法时会默认调用sync方法。是否需要手动调用取决你根据程序需要在数据健壮性和吞吐率之间的权衡。



### 5、NameNode安全模式

1） 安全模式是NameNode的一种保护和恢复机制。大家记得当NameNode运行的时候，元数据都保存在内存中，block与datanode的映射关系也是在NameNode启动的建立的。这个加载数据的过程需要一段时间，所以NameNode在启动的时候会有安全模式，禁止所有的写操作。这个安全模式默认是30秒的时间。当30秒过后，会自动退出安全模式，就可以进行正常的读写操作了。

2） 当重启NameNode进程的时候，系统会进入安全模式。

在安全模式下，无法执行写操作，可以执行读操作。

### 7、HDFS的高可靠

高可靠，就是很可靠。

为了数据的可靠，我们使用了副本replica机制，可以保证数据不丢失。

对于NameNode中的数据，怎么保证可靠性？就是这里讲的。

NameNode中的数据同时有两份，就是有2个NameNode同时运行，保存数据。

在Hadoop下，会有2个NameNode同时运行，他们的状态不同。客户端只能访问其中那个状态是active的NameNode。状态是standby的NameNode是无法对外提供访问的。

当客户端访问active状态的NameNode的时候，会进行写操作。active的NameNode会把这些写操作实时同步给standby状态的NameNode。这样，就能保证两个NameNode的数据实时同步。

当active状态的NameNode宕机时，需要把standby状态的NN的状态修改为active。修改成功后，就可以对外提供访问了。那么修改状态的方式有两种：一种是手工完成的，一种是自动完成的。

### 5、HDFS的federation联盟

NameNode节点的内存不足，会造成整个集群的不可用。所以可以使用2个集群，共享DataNode。

联盟指的是有2个HDFS集群，共享DataNode集群。

## 八、MapReduce

### 1、MR简介

mr是一个分布式的计算模型。

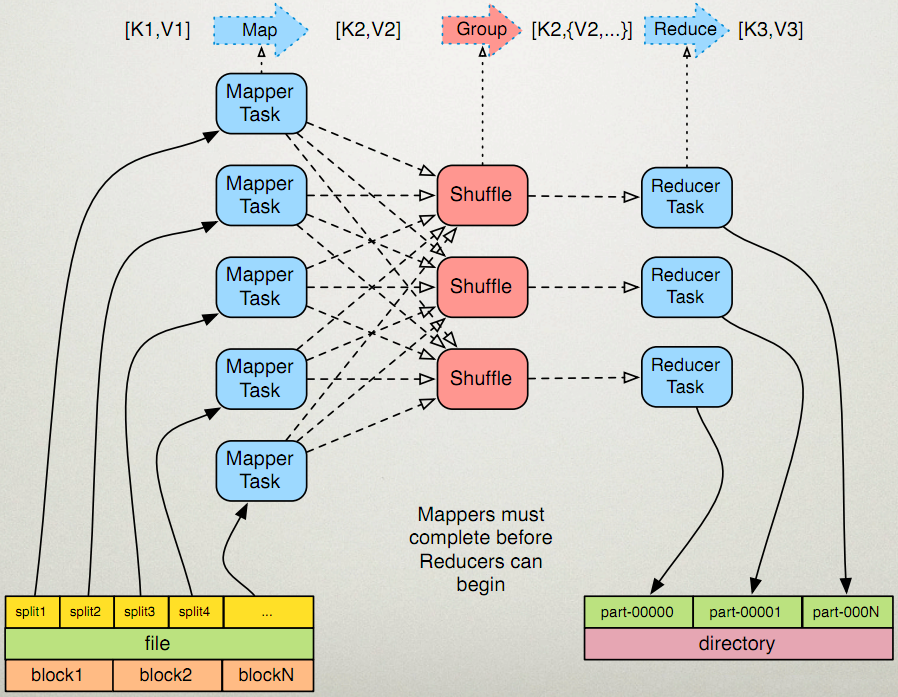
分为两步，第一步是map阶段，第二步是reduce阶段。

既然是计算，一定有输入数据和输出数据。HDFS的数据都是存放在DataNode中的。DataNode和NodeManager都是同一个节点。map阶段负责读取的是每个DataNode输入数据，map阶段处理数据的时候，是针对每个DataNode中的block的每一行甚至每个字符单独做处理。就是说，map阶段是单独处理数据的，不能对处理后的结果进行汇总。reduce阶段负责把map阶段的输出进行再次处理。“再次处理”一般指的是汇总操作。

MapReduce框架都有默认实现，用户只需要覆盖map()和reduce()两个函数，即可实现分布式计算，这两个函数的形参和返回值都是<key、value>，使用的时候一定要注意构造<k,v>。

MapReduce在多于10PB数据时趋向于变慢。

### MR原理



### 3、MR计算过程

map、reduce两个阶段用两个类表示，分别是Mapper类、Reducer类。

public class Mapper{

public void map(key, value, context){

}

}

public class Reducer{

public void reduce(key, values, context){

}

}

业务场景：

对文件

Hello you

Hello me

进行单词汇总

#### 3.1 Map阶段执行过程

##### 3.1.1 Map读取输入数据

使用FileInputFormat类从HDFS中读取数据。

把读取的数据拆分成行，每次map处理一行的数据。文件有2行，那么map函数会被调用2次。因此如果文件有n行，那么map函数会被调用n次。

map处理的输入数据就是kv结构的。这里的k是偏移量，v是一行文本内容。

##### 3.1.2 Map处理输入数据

第一次调用map函数的时候，key=0， value=hello you

第二次调用map函数的时候，key=10，value=hello me

map(key, value, context){

String[] splited = value.split(“\\s”); //拆分字符串

for (String word : splited){

key\_out = word

value\_out = 1

context.write(key\_out, value\_out); //在一次map函数中，该方法会被调用2次

}

}

当map函数2次被调用结束后，会输出4个kv对，

分别是<hello, 1>,<you, 1>,<hello, 1>,<me, 1>。

map进程的数量由输入数据的block的数量决定。

map函数的调用次数由文件中行的数量决定。

##### 3.1.3 对Map输出的<k,v>对进行分区

默认情况下，只有1个分区，即所有的kv对都到一个分区中去。

疑问：分区是如何分区的？？？

##### 3.1.4 对每个分区的<k,v>对进行排序

按照k的大小进行排序，按照字典排序。

上面的输出排序后的结果是<hello, 1>,<hello, 1>,<me, 1>,<you, 1>。

疑问：什么是字典排序？？？字典排序的原理是什么？？？

##### 3.1.5 对每个分区排序后的<k,v>对进行分组

上面排序后的数据分组后是<hello, {1, 1}>,<me,{1}>,<you, {1}>。分成了3组。

分组不会影响kv对的数量，分组后有4个kv对。

##### 3.1.6（可选）对每个分区分组后的数据执行combiner

对每个分区分组后的数据进行压缩，使每个分区针对每一个词只会发送一个值到reducer，大大减少了shuffle过程所需要的带宽并加速了作业的执行，类似局部汇总。combiner后的数据是：<hello,2>,<me,1>,<you,1>

#### 3.2 Reduce阶段执行过程

##### 3.2.1 shuffle

map程序和reduce程序不一定在同一个节点上。

reduce要通过网络读取map阶段运行的结果。每个分区对应一个reduce进程。相同分区的数据进入同一个reduce进程。有多少个分区就有多少个reduce进程。

数据从map节点传输到reduce节点，因为不同map节点中相同分区的数据会被传输到同一个reduce中，所以数据在网络中传输的时候，显得乱糟糟的，称作shuffle过程。【shuffle的本义指的是扑克牌中的洗牌过程】

shuffle实际上是map输出的数据通过网络传输到reduce节点的过程。

##### 3.2.2 reduce 函数处理数据

reduce接收的是分组后（combiner后）的数据。

public class Reducer{

public void reduce(key, values, context){

int total\_times = 0

for (times : values){

total\_times += times

}

key\_out = key

value\_out = total\_times

context.write(key\_out, value\_out)

}

}

一个分组调用一次reduce函数。reduce函数会被调用3次。

最后写出3个kv对，分别是<hello, 2>, <me, 1>, <you, 1>。

##### 3.2.3 把reduce输出写到HDFS

使用FileOutputFormat类把输出写入到HDFS中