大数据4V特征：Volume（海量的数据规模，体积数据量大），Variety（多样的数据类型，种类多数据关联性大），Velocity（快速的数据流转，变化速度快），Value（价值密度低）

Hadoop技术大多来源于Google大数据技术，Google大数据技术可以在普通计算机上使用，Google只发表了论文并没有将大数据技术实现代码开源，Hadoop组织根据论文将其进行代码实现并进行了开源，Hadoop大数据技术HDFS对应Google大数据技术GFS技术解决存储容量问题，Hadoop大数据技术HBase对应Google大数据技术BigTable技术解决读写速度问题，Hadoop大数据技术MapReduce技术对应Google大数据技术MapReduce技术解决计算效率问题

Hadoop官方网址<http://hadoop.apache.org/>

分布式文件系统HDFS（Hadoop Distributed File System）特点：扩展性，容错性，海量数据存储

上传到分布式文件系统HDFS的文件，会被分成指定大小的数据块（默认128M）并以多副本形式存放在多台机器上

资源调度系统YARN（Yet Another Resource Negotiate）：负责整个集群资源的管理和调度，通用的资源管理系统，为上层应用提供统一的资源管理和调度，特点：扩展性，容错性，多框架资源统一调度

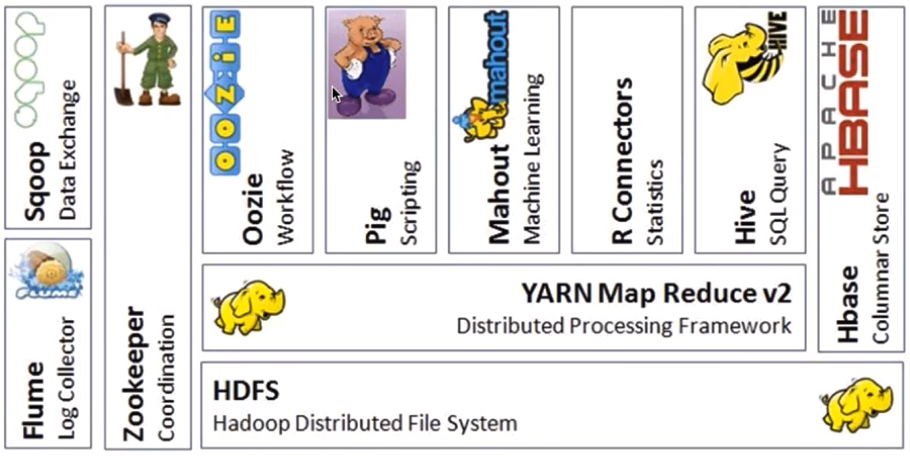
分布式计算框架MapReduce特点：扩展性，容错性，海量数据离线处理

MapReduce处理过程：Input将文件加载到内存通过Splitting按照一定策略进行分割拆分成多个文件交给Mapping进行处理，Shuffling将Mapping处理的结果按照一定策略进行分组，最后Reducing进行将分组后的结果按组进行合并统计，Final result将所有组的结果进行合并统计

Hadoop可靠性：数据多副本存储，任务作业错误会重新调度作业计算

Hadoop扩展性：存储/计算资源不够时，可以横向的线性扩展廉价的普通机器，一个集群可以包含数以千计的节点

Hadoop生态系统：



Hive：SQL Query（SQL查询）由Facebook开源，定义了类似SQL的HiveQL语言，会将HiveQL语言转换成MapReduce作业在Hadoop集群上运行，数据仓库工具，可以将Hadoop下的原始结构化数据变成Hive的表，使用一种几乎和SQL完全相同的语言HiveQL，但不支持更新、索引、事务等

R Connectors：R语言用于统计分析

Mahout：Machine Learning机器学习

Pig：Scripting脚本语言，可以将脚本转换成MapReduce作业在Hadoop集群上运行，使用类似SQL的面向数据流的语言Pig Latin，Pig Latin可以完成排序，过滤，求和，聚组，关联等操作，可以支持自定义函数，Pig会将Pig Latin映射为MapReduce作业上传到集群运行，减少Java代码的编写，Pig的三种运行方式：Grunt shell，脚本方式，嵌入式

Oozie：Workflow工作流，配置有依赖关系的作业，按照依赖关系顺序执行

Zookeeper：Google Chubby的开源实现，coordination协调和谐管理，协调管理Hadoop集群上的多个框架之间的关系，用于协调分布式系统上的各种服务，应用场景HBase、实现NameNode的自动切换

Flume：LogCollector日志收集，将日志文件收集到Hadoop集群上进行处理统计分析

Sqoop（SQL to Hadoop）：Data Exchange，传统关系型数据库数据与Hadoop集群数据之间的数据转换框架，使用JDBC接口

HBase：Columnar Store列式存储的数据库，可以在大数据里进行快速查询，列式数据库，可集群化，可以使用shell，web，api等多种方式访问，适合搞读写（insert）的场景，HQL查询语言，NoSQL的典型代表产品

Avro：数据序列化工具，用于支持大批量数据交换的应用，支持二进制序列化方式，可以便捷快速的处理大量数据，动态语言友好，Avro提供的机制是动态语言可以方便的处理Avro数据，Thrift接口

Chukwa：架构在Hadoop之上的数据采集与分析架构，主要进行日志采集和分析，通过安装在手机节点的代理采集最原始的日志数据，代理将数据发给收集器，收集器定时将数据写入Hadoop集群，指定定时启动的MapReduce作业对数据进行加工处理和分析，Hadoop基础管理中心（HICC）最终展示数据

Cassandra：NoSQL，分布式的Key-Value型数据库，由Facebook贡献，与HBase类似，也是鉴于Google BigTable的思想体系，只有顺序写，没有随机写的设计，满足高负荷情形的性能需求

Hadoop常用发行版：

Apache Hadoop：Hadoop生态圈可以单个框架使用，多个框架同时使用会出现jar包冲突等问题，常用于学习

CDH：Cloudera Distributed Hadoop，可以使用CM框架通过浏览器安装Hadoop生态系统，但CM框架不开源，在Apache Hadoop的基础上进行了改进，使用CDH相同版本基本可以免除多个框架之间的jar包冲突等问题，公司生产环境常用，官方下载地址<http://archive.cloudera.com/cdh5/cdh/5/>

HDP：Hortonworks Data Platform，使用原生的Apache Hadoop，开源

HDFS是一个M/S（master/slave）架构一个master（NameNode/NN）带领多个slave（DataNode/DN），一个文件会被拆分成多个Block（blocksize默认128M）

NameNode/NN：负责客户端请求的响应，负责 metadata元数据（文件的名称、副本系数、权限、Block存放的DataNode/DN）的管理，metadata元数据在集群启动时加载到内存，定期会持久化到fsimage磁盘文件中，edits记录对metadata的操作日志

SecondaryNameNode/SNN：不是NN的备份但可以做备份，SNN主要负责帮助NN合并edits log、fsimage文件减少NN的启动时间，配置文件中fs.checkpoint.period指定SNN合并时间间隔默认3600秒，fs.checkpoint.size指定edits log文件的最大值默认64M

DataNode/DN：负责存储用户的文件对应的数据块（Block），定期（默认3秒，默认超过10分钟表示DataNode节点丢失）向NN发送心跳信息，回报本身及其所有的block信息，健康状况

NameNode与DataNode可以部署在同一个节点（即廉价计算机上），一般建议部署在不同节点上

NameNode（Filename，numReplicas，block-ids），文件全路径名，r:文件副本系数数值（即文件被复制成几份保存），{DataNode节点id [，DataNode节点id对应文件副本系数的数值个数]}

HDFS副本存放策略：第一个副本在存放源文件相同机架的相同节点上（若源文件在集群外部主机上，则随机挑选一台磁盘不满，CPU不忙的主机节点），第二个副本在与第一个副本不同的随机一个机架上的随机节点上，第三个副本在与第二副本相同机架的不同节点上，更多副本随机节点存放

HDFS文件权限与Linux文件权限类似，

Hadoop伪分布式安装步骤：

Jdk安装：解压tar –zxvf jdk的压缩包 –C 安装目录

添加环境变量，一般为用户环境变量：~/.bash\_profile

export JAVA\_HOME=java安装目录全路径

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

使环境变量生效：source ~/. bash\_profile

验证java是否配置成功：java -version

安装ssh：

Ubuntu Linux系统安装ssh：

sudo apt-get install ssh

sudo apt-get install rsync

CentOS Linux系统安装ssh：

sudo yum install ssh

由于NameNode与DataNode节点之间需要通信，为了方便需要设置节点通信免密码登录：

ssh-keygen -t rsa 使用rsa（基于因式分解在时间上不对称的加密算法）加密产生秘钥对（公钥和私钥，用公钥加密的文件只能通过私钥解密，用私钥加密的文件只能通过公钥解密）

cp ~/.ssh/id\_rsa.pub ~/.ssh/authorized\_keys 在本机免密码登录配置文件authorized\_keys中加入允许免密码连接本机的主机生成的公钥

ssh原理：ssh 主机名，本机连接到指定主机名的主机，本机先发送连接请求（默认附加端口22），主机服务器将ssh公钥发送给本机，本机收到公钥后使用其他加密方式生成一个随机256位的会话通信密钥，并将会话通信密钥使用主机服务器公钥加密后发送给服务器主机，服务器主机用私钥解密用公钥加密的会话通信密钥，服务器主机与本机之间使用会话通信密钥进行通信（rsa密钥的加密算法工作量大，不适合做通信），服务器主机请求本机输入用户名密码，本机发送用户名密码，服务器主机验证用户名密码，根据验证是否正确接受或拒绝本机连接

下载并解压Hadoop，下载CDH的地址<http://archive.cloudera.com/cdh5/cdh/5/>

解压tar –zxvf Hadoop的CDH版压缩包 –C 安装目录

在hadoop-env.sh文件中配置java的环境变量JAVA\_HOME

Hadoop配置文件修改（文件位置Hadoop的安装目录/etc/hadoop/）：

core-site.xml:

<configuration>

<property>

<name>fs.defaultFS</name>

<value>hdfs://主机名:端口号（单机的主机名可以为localhost，端口号Hadoop1默认为9000，Hadoop2默认为8020）</value>

</property>

<property>（配置Hadoop文件目录，防止电脑重启后删除Hadoop的数据）

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

<value>系统绝对路径</value>

</property>

</configuration>

hdfs-site.xml:

<configuration>

<property>

<name>dfs.replication</name>

<value>文件副本系数（默认为3，单机模式为1）</value>

</property>

</configuration>

slaves配置节点主机名，单机模式可以为localhost或主机名，集群模式配置IP地址

启动HDFS：

格式化文件系统：hdfs/hadoop namenode –format（第一次使用HDFS系统时使用，格式化清除数据）

Hadoop命令hdfs是Hadoop2中默认命令，hadoop是Hadoop1默认命令，hadoop虽然过时但还能使用，建议使用hdfs

启动HDFS：sbin/start-dfs.sh

验证是否启动成功：使用查看与java有关的进程命令jps会有DataNode，SecondaryNameNode，NameNode进程或者通过浏览器进入http://主机名（或者IP地址）:50070/查看启动状态

jps(Java Virtual Machine Process Status Tool)是JDK 1.5提供的一个显示当前所有java进程PID的命令，jps [-q/m/l/v] 显示当前所有Java进程[只显示PID/显示传递给进程主函数的参数/显示进程主函数完成的package全包名或进程程序的jar文件完整路径名/显示传递给JVM的参数]PID和进程主函数名

通过hadoop shell命令方式上传到集群的文件，会采用hdfs-site.xml中设置的副本系数，通过java api方式上传到集群的文件，如果本地java资源文件里没有hdfs-site.xml文件配置，则默认使用hadoop的副本系数3

HDFS系统：

Client：发送读写请求

NameNode：协调控制所有请求，节点数量为1

DataNode：负责数据存储，节点数量可以有很多

HDFS上传数据写操作：Client收到写操作命令请求后，通过解析配置文件信息中用户名（权限认证根据用户名匹配）、存储块的大小以及文件副本系数，将数据拆分成相应数据块，然后发送请求给NameNode，按顺序发送数据块，并表明文件存储块的大小和文件副本系数，NameNode接收请求后根据文件副本系数，找到对应文件副本系数数量的可用DataNode，并将DataNode的信息按照距离Client的远近从近到远进行排序后返回给Client（作业调度是优先调度到最近的DataNode上），Client开始发送数据到第一个DataNode，第一个DataNode在接收数据的同时将收到的数据发送给下一个DataNode，依次按照顺序发送给下一个DataNode直到最后一个DataNode，最后一个DataNode接收完这个数据块的所有数据后，所有DataNode发送消息给NameNode表明该数据块存储完成，NameNode返回该数据块接收完成的消息给Client，Client接收返回的消息后按照数据块顺序将剩下的数据块重复以上步骤依次发送直到所有数据块发送完成，Client发送关闭文件流的请求，然后NameNode关闭文件流并记录源数据信息（数据块的数量，id以及存储数据的DataNode信息等）

HDFS下载数据读操作：Client收到读操作命令请求后，发送需要读取的文件名称信息（用户名，权限认证根据用户名匹配）给NameNode，NameNode根据文件名称通过计算将该文件对应的所有数据块以及每个块对应的文件副本数量和文件副本所在的DataNode信息按照距离Client的远近从近到远进行排序后返回给Client，Client收到返回的信息后去和第一个DataNode（即距离Client最近的）交互获取相应数据块

HDFS缺点：低延迟的数据访问，不适合小文件存储

MapReduce1的架构：Master/Slave架构，1个JobTracker带多个TaskTracker

JobTracker/JT：作业的管理者，负责资源管理和作业调度，将作业分解成一堆的任务Task（MapTask和ReduceTask），将作业分给TT运行，监控作业，处理容错（task作业挂了，重启task的机制），在一定的时间间隔内未收到TT的心跳信息，认为该TT可能挂了，将该TT上运行的任务分配到其他TT上执行

TaskTracker/TT：任务的执行者，在TT上执行Task（MapTask和ReduceTask）定期向JobTracker汇报本节点的健康状况、资源使用情况、作业执行情况等心跳信息，接收JobTracker的启动/结束/停止作业任务的命令

MapReduce1存在的缺点问题：由于集群里只有一个JobTracker所以有单点故障问题，由于JobTracker需要和所有Client以及TaskTracker交互信息，所以节点压力大不易扩展，只能支持MapReduce作业不支持其他作业调度

HDFS2.x使用Federation（联邦）解决HDFS1.x的单点故障等问题，通过可以水平扩展多个NameNode/Namespace将元数据的存储和管理以及负载分散到多个节点上（解决内存受限问题），可以通过多个Namespace隔离存储和管理不同类型的应用的元数据

YARN框架为了提高资源利用率减少运维成本，将所有小集群合并在一起运行调度各种任务，不同的计算框架可以共享同一个HDFS集群上的数据，使用整体的资源调度

YARN的优点：与其他计算机框架共享集群资源，按资源需要分配，进而提高集群资源利用率

YARN的架构：

ResourceManager/RM：负责集群资源的统一调度和管理，处理客户端的提交/结束作业请求，监控所有NM，当某个NM挂了，将该NM上运行的任务信息提交给对应的AM请求处理，整个集群同一时间提供服务的RM只有一个（一般有两个，另一个是备用状态）

NodeManager/NM：单个节点资源管理，负责本身节点资源管理和使用，定时向RM汇报本节点的资源使用情况，定时向RM汇报本节点的资源使用情况，接收RM的各种命令启动Container，整个集群中有多个

ApplicationMaster/AM：每一个应用程序对应一个AM，负责应用程序的管理，为应用程序向RM申请资源，分配给内部task，需要与NM交互启动/停止task，task与AM都是运行在Container里

Container：封装了集群里的资源的一个容器，是一个任务运行环境的抽象

Client：通过命令提交/结束作业，查看作业的运行进度

YARN执行流程：

Client向RM提交作业，RM为该作业分配一个NodeManager启动一个Container运行AM，AM与RM注册交互（注册完成后Client可以查看任务作业运行情况）申请该作业需要的资源，AM得到资源后到对应的NodeManager上启动对应的Container执行作业任务

YARN环境的搭建：

修改配置文件：

mapred-site.xml:

<configuration>

<property>（配置指定MapReduce的框架为YARN）

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

<value>yarn</value>

</property>

<property>（配置JobHistory的服务IPC地址以及端口）

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

<value>ubuntu:10020</value>

<description>MapReduce JobHistory Server IPC host:port</description>

</property>

<property>（配置JobHistory的服务web访问地址以及端口）

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

<value>ubuntu:19888</value>

<description>MapReduce JobHistory Server Web UI host:port</description>

</property>

<property>（配置JobHistory作业完成时的存放目录）

<name>mapreduce.jobhistory.done-dir</name>

<value>/history/done</value>

</property>

<property>（配置JobHistory作业正在运行的存放目录）

<name>mapreduce.jobhistory.intermediate-done-dir</name>

<value>/history/done\_intermediate</value>

</property>

</configuration>

yarn-site.xml:

<configuration>

<property>（配置指定YARN的NodeManager的服务）

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

<value>mapreduce\_shuffle</value>

</property>

<property>（配置YARN的日志收集聚合功能）

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

<value>true</value>

</property>

</configuration>

启动YARN相关的进程：sbin/start-yarn.sh

验证是否启动成功：jps查看进程是否有ResourceManager和NodeManager或者通过浏览器进入http://主机名（或者IP地址）:8088/查看应用状态

停止YARN相关进程sbin/stop-yarn.sh

YARN命令式提交作业：hadoop jar MapReduce作业的jar包 执行类 执行类相应的参数

MapReduce作业的案例：hadoop安装目录/share/hadoop/mapreduce/ MapReduce作业jar包

Split：交由MapReduce作业处理的数据块，是MapReduce中最小的计算单元

Block：HDFS中最小的存储单元，默认blocksize为128M

默认情况下Split与Block是相对应的，可以配置进行修改（不建议）

InputFormat：将输入的数据进行分片（Split），常用的是处理文本格式的数据TextInputFormat

OutputFormat：数据的输出

Combiner：在reduce之前进行合并，减少Map Tasks输出的数据量及数据网络传输量，使用场景适合相加操作，不适合求平均数操作

Partitioner：

MapReduce框架处理数据是将数据文件拆分成键值对<key,value>方式并按照键值对<key,value>方式进行计算处理

Map Task：解析每条记录的数据，交给自己的map方法处理，将map的输出结果写到本地磁盘（有些作业只要Map操作没有Reduce操作）

Reduce Task：将Map Task输出的数据进行读取，按照数据进行分组传给reduce方法处理，输出结果到HDFS

数据文件通过InputFormat类读取并拆分成多个Split<key,value>（默认Split个数与Block个数对应），通过RecordReader将每个Split中的数据按照一定规则读取到Map，一个Split对应一个Mapper Task，Map处理完成后将数据按照相同或者一定规则的key原则排序后发送给不同的Shuffle，Shuffle进行分组合并后发送给Reduce，Reduce Task进行value计算统计处理，最后交给OutputFormat将数据存储到HDFS中，文件输出个数与Reduce Task个数对应

Maven编译：mvn clean package –DskipTests

上传到服务器：scp 源文件路径 服务器路径

运行命令：hadoop jar 应用程序包 运行的主类全类名 相关执行参数

相同的代码和脚本再次执行会报文件已经存在的错误，在MapReduce中，输出的文件是不能事先存在的，可以输入命令删除或者在代码中完成自动检测文件是否存在删除

jobhistory：记录已经运行完的MapReduce信息到指定的HDFS目录下，默认未开启，需要在mapred-site.xml中配置开启，配置完成后需要启动JobHistory的服务：./mr-jobhistory-daemon.sh start historyserver

验证JobHistory的服务是否启动成功：jps查看是否有JobHistoryServer进程

数据处理流程：

1、数据采集

2、数据清洗

3、数据处理

4、处理结果入库

5、数据的可视化

Ubuntu系统修改用户名：

sudo –i su root 切换到root用户

修改/etc/passwd，/etc/shadow，/etc/group文件中用户名为指定用户名

mv /home/用户名 /home/指定用户名 修改用户家目录

su 用户名或者重启系统 切换用户或者重启系统

Hadoop分布式集群环境搭建：

前置安装配置：

集群所有主机时间同步：

安装ntp，使用时间服务器（通过网络请求时间服务器返回时间）同步时间，使用命令ntpdate -u 时间服务器域名或IP地址 设置时间服务器为指定的域名或IP地址

网络互通检查，ping网络是否相通

修改集群主机名：

Ubuntu系统修改：

sudo –i su root 切换到root用户

修改/etc/hostname文件中主机名为指定主机名重启系统

CentOS系统修改：

使用root权限修改/etc/sysconfig/network文件中主机名为指定主机名，CentOS7主机名配置文件/etc/hostname

修改集群hosts文件中IP地址以及主机名

对每个主机节点合理分配角色即NameNode/SecondaryNameNode/DataNode（一般SecondaryNameNode与NameNode不分配到同一台主机上）和ResourceManager/NodeManager

配置主机节点之间免密码登录：

在每台主机上运行ssh-keygen -t rsa命令生成免密码的秘钥

在集群主节点即NameNode与ResourceManager所在的主机节点上运行ssh-copy-id -i ~/.ssh/id\_rsa.pub 所有主机节点的主机名，将主节点的公共秘钥分发到所有主机节点上，有可能会因为ssh-agent中未添加秘钥导致ssh免密码登录失效，使用eval "(ssh-agent -s)"和ssh-add命令添加秘钥

jdk安装：

tar -zxvf jdk安装包 –C 安装目录

在~/.bash\_profile中配置环境变量：

export JAVA\_HOME=java安装目录全路径

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

Hadoop集群安装

tar –zxvf Hadoop的CDH版压缩包 –C 安装目录

添加Hadoop的环境变量：

export HADOOP\_HOME=hadoop安装目录全路径

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

source ~/.bash\_profile使环境变量生效

Hadoop配置文件修改（文件位置Hadoop的安装目录/etc/hadoop/）：

创建masters文件，加入SecondaryNameNode主机名或IP地址，不指定默认与NameNode同一主机节点

hadoop-env.sh:

export JAVA\_HOME=java安装目录全路径

core-site.xml:

<configuration>

<property>

<name>fs.defaultFS</name>

<value>hdfs://主机名:端口号（单机的主机名可以为localhost，端口号Hadoop1默认为9000，Hadoop2默认为8020）</value>

</property>

<property>（配置Hadoop文件目录，防止电脑重启后删除Hadoop的数据）

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

<value>系统绝对路径</value>

</property>

</configuration>

hdfs-site.xml:

<configuration>

<property>配置文件系统存放的目录

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

<value>NameNode节点存放目录的绝对路径</value>

</property>

<property>

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

<value>DataNode节点存放目录的绝对路径</value>

</property>

<property>

<name>dfs.replication</name>

<value>文件副本系数（默认为3，单机模式为1）</value>

</property>

<property>

<name>dfs.namenode.secondary.http-address</name>（配置SecondaryNameNode的Web访问地址）

<value>IP地址或主机名:端口号（默认50090）</value>

</property>

<property>（配置Federation允许多个NameNode）

<name>dfs.nameservices</name>（配置自定义名称服务即命名空间，命名规则是以字母开头的数字和字符组合）

<value>NameNode名称[,NameNode名称]</value>

</property>

[<property>（配置Federation的多个NameNode）

<name>dfs.namenode.rpc-address.NameNode名称</name>

<value>主机名:端口号</value>

</property>

<property>

<name> dfs.namenode.http-address.NameNode名称</name>

<value>主机名:端口号</value>

</property>

<property>

<name> dfs.namenode.secondary.http-address.NameNode名称</name>

<value>主机名:端口号</value>

</property>]

</configuration>

yarn-site.xml:

<configuration>

<property>（配置指定YARN的NodeManager的服务）

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

<value>mapreduce\_shuffle</value>

</property>

<property>（配置YARN的ResourceManager）

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

<value>指定角色为ResourceManager的主机节点主机名</value>

</property>

<property>（配置YARN的日志收集聚合功能）

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

<value>true</value>

</property>

</configuration>

mapred-site.xml:

<configuration>

<property>（配置指定MapReduce的框架为YARN）

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

<value>yarn</value>

</property>

<property>（配置JobHistory的服务IPC地址以及端口）

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

<value>ubuntu:10020</value>

<description>MapReduce JobHistory Server IPC host:port</description>

</property>

<property>（配置JobHistory的服务web访问地址以及端口）

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

<value>ubuntu:19888</value>

<description>MapReduce JobHistory Server Web UI host:port</description>

</property>

<property>（配置JobHistory作业完成时的存放目录）

<name>mapreduce.jobhistory.done-dir</name>

<value>/history/done</value>

</property>

<property>（配置JobHistory作业正在运行的存放目录）

<name>mapreduce.jobhistory.intermediate-done-dir</name>

<value>/history/done\_intermediate</value>

</property>

</configuration>

slaves配置所有DataNode节点主机名，指定集群中所有的DataNode节点主机

分发主节点上的安装包到其他主机节点上：

scp –r 安装位置的文件夹 其他所有主机节点用户名@主机名:安装位置的文件夹，-r表示源文件目录下所有文件

scp ~/.bash\_profile其他所有主机节点用户名@主机名:安装位置的文件夹，分发环境变量文件，并在其他所有主机节点上使用source命令使环境变量生效

在主节点上Hadoop安装目录的bin目录下格式化文件系统：./（hdfs/hadoop）namenode –format

在主节点上启动Hadoop集群：sbin/start-all.sh

验证Hadoop集群启动成功：

jps验证：主节点即NameNode和ResourceManager节点主机有SecondaryNameNode，NameNode，ResourceManager进程，其他主机节点即DataNode和NodeManager节点主机有DataNode，NodeManager进程

WEBUI验证：http://主节点主机名（或者主节点IP地址）:50070/查看Hadoop启动状态，http://主节点主机名（或者主节点IP地址）:8088/查看YARN启动状态

集群停止：sbin/stop-all.sh

Hadoop项目在集群上运行：

上传运行数据到集群机器上后转到hdfs上，上传开发的jar包到主节点的lib目录下，使用hadoop jar MapReduce作业的jar包 执行类 执行类相应的参数命令运行

NameNode启动时先经过一个安全模式阶段，该阶段不会产生数据写，会将持久化的磁盘文件fsimage和edits文件进行合并生产并持久化新的fsimage文件和空的edits文件（由NameNode进行合并不是SecondaryNameNode操作，SecondaryNameNode在集群运行时进行合并操作），然后NameNode在此阶段是收集各个DataNode的报告，当数据块达到最小副本数以上认为是安全的，检测到副本系数不足的数据块时会复制该数据块直到达到最小副本数，在一定比例（可设置）的数据被认为安全后，经过一段时间，安全模式会自动结束

hdfs dfsadmin –safemode enter/leave 开启/关闭安全模式

HDFS的可靠性机制包括：冗余副本策略，机架策略，心跳机制，安全模式，校验和，回收站，元数据保护，快照机制

校验和：在文件建立时，每个数据块都会产生校验和，校验和会作为一个单独隐藏文件保存在命名空间下，客户端获取数据时可以检查校验和是否相同来确定数据块是否损坏，如果损坏可以读取其他副本

回收站：删除文件时，其实是放入回收站/trash中，回收站中的文件可以恢复，回收站可以设置时间阈值，当回收站中的文件存放时间超过阈值会被彻底删除释放占用的数据库

元数据保护：映像文件和事务日志是NameNode的核心数据，可以设置拥有多个副本，增加副本会降低NameNode的处理速度，但会增加安全性，NameNode虽有副本，但还是单节点，需要手动切换副本

快照：支持存储某个时间点的映像，需要时可以使数据重返这个时间点的状态

/etc/resolv.conf 文件中保存着DNS服务器的域名

nslookup 查看DNS服务器里域名对应的IP地址

Hadoop大集群搭建即节点很多（成百上千）：

密钥分发可以使用NFS（网络文件系统）将密钥文件共享整个集群

hosts文件配置可以使用DNS代替（一般Linux的DNS服务器配置软件使用bind）

hadoop目录广播分发可以使用cat ./slave | awk '{print "scp -rp 文件路径 用户名@"$1":文件路径"}' > 文件名 生成分发主节点上的安装包到所有主机节点上脚本命令到一个文件里，chmod a+x 文件名 赋予文件执行权限，sh ./文件名 执行文件里所有命令进行广播分发hadoop目录

在已经启动的hadoop集群里添加节点：

需要在新节点上安装hadoop以及修改相应配置文件，或者从集群里的节点上复制文件到新节点，修改slave文件增加新节点，设置ssh免密码登录，最好是重启集群，如果集群不能停，可以单独启动新节点上的相应进程即使用命令hadoop-daemon.sh start 进程名，然后运行start-balancer.sh进行数据负载均衡，当节点出现故障或增加新节点，数据块分布可能不均匀，负载均衡可以重新平衡各个DataNode上数据块的分布

性能调优：reducer数量配置在mapred-site.xml文件中mapred.reduce.tasks（默认1）设置，数量少数据大的文件处理速度比数量多数据小的文件快，减少网络传输即压缩map的输出，通过参数配置优化每个节点能运行的任务数量

任务执行优化：

推测式执行即主节点发现有节点任务执行比其他节点慢，会另启一个节点执行相同的备份任务，然后节点任务和备份节点任务有一个先执行完成会把另一个kill，因此在监控页面上会看到正常执行的任务被kill，推测式执行默认打开，在mapred-site.xml文件中可以配置关闭，如果是程序本身问题，推测式执行会让集群更慢

重用JVM，可以省去启动新的JVM消耗的时间，在mapred-site.xml文件中可以配置hadoop集群启动的单个JVM运行的最大任务数量（默认1，-1表示无限制）

忽略模式，任务再读取数据失败2次后，会将数据位置告诉主节点，主节点重新启动该任务并且在遇到所记录的坏数据时直接跳过（默认关闭状态，可用SkipBadRecord方法打开）

主节点会根据心跳（周期1分钟）去检测任务节点是否故障，当任务节点在执行map任务并且未完成发生故障时，主节点会在其他任务节点重新启动这个map任务，当任务节点在执行reduce任务并且未完成发生故障时，主节点会在其他任务节点继续执行为完成的reduce任务

审计日志，可以在log4j.properties配置文件中的log4j.logger.org.apache.hadoop.fs.FSNamesystem.aduit=WARN改为INFO，即不仅仅是发生WARN事件才记录日志，任何INFO信息事件都会记录日志

Hadoop使用stream流的形式执行，可以运行其他语言的MapReduce作业任务：hadoop jar hadoop的stream流的jar包 –input 文件输入全路径 –output 文件输出全路径 –mapper 指定map函数的文件全路径名 –reducer 指定reduce函数的文件全路径名

HDFS可以使用NFS实现HA，但NFS本身也存在单点故障问题，一般使用QJM实现HA

HDFS使用QJM实现高可用HA原理：多台NameNode并存，仅有一台NameNode处于Active状态，其他NameNode处于Standby，SecondaryNameNode合并文件的操作交由处于standby状态的NameNode操作，多台JournalNode组成的集群存储管理共享元数据文件给多台NameNode，多台Zookeeper组成的集群管理整个集群节点

HDFS使用QJM实现高可用HA配置：

给集群所有主机节点分配好角色：NameNode，DataNode，JournalNode，Zookeeper，ZKFailoverController

时间同步，测试网络连通性，配置主机之间免秘钥登录，修改hosts文件，检查防火墙，解压安装文件（JDK，Zookeeper，Hadoop）

删除masters文件修改配置文件：

core-site.xml

<configuration>

<property>

<name>fs.defaultFS</name>（指定HDFS集群访问地址）

<value>hdfs://名称服务名</value>

</property>

<property>

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

<value>/home/kaku/tmp/hadoop</value>

</property>

</configuration>

hdfs-site.xml

<configuration>

<property>

<name>dfs.nameservices</name>（配置名称服务）

<value>自定义名称服务名</value>

</property>

<property>

<name>dfs.ha.namenodes.名称服务名</name>（多个NameNode节点的名称）

<value>NameNode节点名称[,NameNode节点名称]</value>

</property>

[<property>

<name>dfs.namenode.rpc-address.名称服务名. NameNode节点名称</name>

<value>主机名:端口号</value>（指定名称服务名的NameNode节点名称的Remote Procedure Call 远程过程调用RPC地址，默认端口8020）

</property>

<property>

<name>dfs.namenode.http-address.名称服务名. NameNode节点名称</name>

<value>主机名:端口号</value>（指定名称服务名的NameNode节点名称的Web访问地址，默认端口50070）

</property>]

<property>

<name>dfs.namenode.shared.edits.dir</name>（指定用于管理NameNode节点的元数据fsimage和edits的JournalNode集群主机节点）

<value>qjournal://主机名:8485（默认端口8485）[;主机名:8485]/名称服务名</value>

</property>

<property>

<name>dfs.client.failover.proxy.provider.名称服务名</name>（指定自动故障迁移，）

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

</property>

<property>

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

<value>sshfence</value>（指定集群免秘钥方式）

</property>

<property>

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

<value>私钥文件全路径名</value>（指定免秘钥的私钥文件地址）

</property>

<property>

<name>dfs.journalnode.edits.dir</name>（指定JournalNode集群存放元数据的目录）

<value>文件全路径名</value>

</property>

</configuration>

配置Zookeeper：

修改Zookeeper安装目录/conf/zoo.cfg配置文件

tickTime=2000 Zookeeper请求时间间隔

dataDir=文件路径名 Zookeeper数据保存数据目录，默认/tmp目录下开启重启数据会被清除

clientPort=2181 Zookeeper端口

initLimit=5

syncLimit=2

[server.数值=主机名:2888:3888] 配置指定数值为ID的Zookeeper集群服务器，默认两个端口2888，3888（Zookeeper集群服务器数量一般是大于等于3的奇数）

在dataDir指定的Zookeeper数据保存路径下创建myid文件保存主机的服务ID，每台Zookeeper集群节点使用zkServer.sh start/status/stop/restart 启动/查看状态/停止/重启 Zookeeper命令，在执行命令的当前路径下会生产Zookeeper日志文件zookeeper.out

修改Hadoop的配置文件hdfs-site.xml添加：

<property>

<name>dfs.ha.automatic-failover.enabled</name>（启用Zookeeper管理高可用，默认false不启用）

<value>true</value>

</property>

core-site.xml添加：

<property>

<name>ha.zookeeper.quorum</name>（指定Zookeeper集群）

<value>主机名:2181（默认端口2181）[,主机名:2181]</value>

</property>

在每台JournalNode节点上使用hadoop-daemon.sh start journalnode命令启动JournalNode

在一台NameNode主机节点上使用hdfs/hadoop namenode –format格式化NameNode并使用hadoop-daemon.sh start namenode命令启动NameNode，在其他没有格式化的NameNode节点上使用hdfs namenode -bootstrapStandby命令同步格式化的NameNode节点（也可手动复制指定文件完成同步）

使用hdfs zkfc -formatZK命令格式化Zookeeper集群

使用start-dfs.sh启动使用QJM配置的高可用HA的HDFS

使用QJM的高可用HA的HDFS搭建完成后可以在每台Zookeeper集群节点上使用命令启动Zookeeper集群，使用使用start-dfs.sh启动使用QJM配置的高可用HA的HDFS，JournalNode和ZKFailoverController在使用QJM配置的高可用HA的HDFS搭建完成后会跟随自动启动和关闭不需要手动启动和关闭

MapReduce的Split大小由计算框架自动完成分片也可通过max.split(单位M数值)/min.spilt(单位M数值)/block(单位M数值)进行指定Split最大/Split最小/存储块大小，计算框架自动Split的规则max(min.split,min(max.split,block))

Shuffler对Map的数据结果进行Partition分区分组（默认对Reduce数量取模%）指定处理该分组数据的Reduce，每一个Map任务都有一个内存缓冲区（默认100M）存储Map输出的结果，当缓冲区到达指定阈值（spill.percent默认0.8）时会缓冲区的数据由单独线程（不影响往缓冲区写Map结果数据的线程）以临时文件的形式进行Sort排序（默认字典排序）和Combiner（简单合并，可以减少数据网络传输和Reduce的合并工作量，默认不进行Combiner操作）并溢写（Spill）到磁盘以及临时文件数据合并，Reduce会从磁盘上获取通过分区分给自己处理的数据进行排序（默认字典排序）合并等处理

计算框架Shuffler中Partition和Sort有默认计算方式可以重写修改，Combiner、Map和Reduce无默认计算方式，需要自己编写计算方式

Hadoop集群高可用一般都交由Zookeeper集群管理，Zookeeper使用自己管理的文件系统数据库管理Hadoop集群高可用信息（例如名称服务名）

Hadoop集群使用Zookeeper集群管理实现高可用的配置：

在高可用HA的HDFS基础上修改配置文件：

mapred-site.xml:

<configuration>

<property>（配置指定MapReduce的框架为YARN）

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

<value>yarn</value>

</property>

<property>（配置JobHistory的服务IPC地址以及端口）

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

<value>主机名:10020</value>

<description>MapReduce JobHistory Server IPC host:port</description>

</property>

<property>（配置JobHistory的服务web访问地址以及端口）

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

<value>主机名:19888</value>

<description>MapReduce JobHistory Server Web UI host:port</description>

</property>

<property>（配置JobHistory作业完成时的存放目录）

<name>mapreduce.jobhistory.done-dir</name>

<value>/history/done</value>

</property>

<property>（配置JobHistory作业正在运行的存放目录）

<name>mapreduce.jobhistory.intermediate-done-dir</name>

<value>/history/done\_intermediate</value>

</property>

</configuration>

yarn-site.xml:

<configuration>

<property>（配置指定YARN的NodeManager的服务）

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

<value>mapreduce\_shuffle</value>

</property>

<property>（指定ResourceManager开启高可用）

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

<value>true</value>

</property>

<property>（指定ResourceManager的集群id，类似名称服务名，由于Zookeeper集群是根据名称服务名管理Hadoop集群高可用信息，所以为避免与HDFS高可用的名称服务名冲突，一般与HDFS高可用的名称服务名不一致）

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

<value>集群ID名称</value>

</property>

<property>（指定ResourceManager集群服务器的名称）

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

<value>ResourceManager名称[,ResourceManager名称]</value>

</property>

[<property>（指定ResourceManager名称的主机节点）

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

<value>主机名</value>

</property>

<property>（指定ResourceManager名称的Web UI访问地址）

<name>yarn.resourcemanager.webapp.address.ResourceManager名称</name>

<value>主机名:端口号（默认8088）</value>

</property>]

<property>（指定管理ResourceManager高可用信息的Zookeeper集群）

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

<value>主机名:2181[,主机名:2181] </value>

</property>

</configuration>

Hadoop集群使用Zookeeper集群管理的高可用配置完成后启动Zookeeper集群后可以使用start-all.sh脚本启动高可用的Hadoop集群，可以使用yarn-daemon.sh start/stop resourcemanager命令启动/停止ResourceManager

Hadoop的依赖jar包文件hadoop安装目录/share/hadoop

Hadoop的官方离线文档hadoop安装目录/share/doc/hadoop

Hadoop的官方离线API文档hadoop安装目录/share/doc/hadoop/api

Configuration 配置名=new Configuration();//创建配置，默认加载src目录下的配置文件（\*-site.xml文件）

FileSystem 文件系统名= FileSystem.get(new URI(HDFS的RPC地址),配置名,"用户名");通过配置文件获取指定用户名和RPC地址的文件系统对象

System.setProperty("HADOOP\_USER\_NAME", "用户名"); 指定程序执行时的用户名

FileSystem.mkdirs(new Path(HDFS路径)); 创建目录

FSDataInputStream HDFS数据输入流名=文件系统名.open(new Path(HDFS路径名)); 获取HDFS数据输入流对象

IOUtils.copyBytes(输入对象,输出对象,输出大小); 以指定大小复制数据到指定对象

HDFS数据输入流名.close(); 关闭HDFS数据输入流

FSDataOutputStream HDFS输出流名=文件系统名.create(new Path(HDFS路径名)); 获取HDFS输出流对象

HDFS输出流名.write(字符串.getBytes()); 以字节流形式写入字符串

HDFS输出流名.flush(); 刷新HDFS输出流

HDFS输出流名.close(); 关闭HDFS输出流

文件系统名.rename(旧HDFS路径名,新HDFS路径名); 重命名文件

文件系统名.copyFromLocalFile(本机路径名,HDFS路径名); 复制本机文件到HDFS

文件系统名.copyToLocalFile(HDFS路径名, 本机路径名); 复制HDFS文件到本机

FileStatus[] 文件状态组名= 文件系统名.listStatus(new Path(路径名)); 获取文件状态对象组

文件状态名

文件系统名.delete(new Path(路径名)); 删除文件

SequenceFile.Writer 文件写入流名= SequenceFile.createWriter(文件系统名,配置名,路径, 键的类,值的类); 获取序列文件（小文件合并成大文件，小文件以key/value保存，小文件名为key，小文件内容为value）的写入流

File 目录文件名=new File(目录文件路径名);

for (File 文件名: 目录文件名.listFiles()) {

writer.append(new Text(文件名.getName()), new Text(FileUtils.readFileToString(文件名)));

}

SequenceFile.Reader 文件写入流名=SequenceFile.create Reader(文件系统名, 路径,配置名); 获取序列文件输出流

public class WordCountApp { Hadoop的MapReduce任务入口类

public class WordCountMapper extends Mapper<LongWritable,Text,Text,IntWritable>{

protected void map(LongWritable key, Text value, Context context) throws IOException, InterruptedException { 执行Mapper任务的类需要继承Mapper类重写map方法

super.map(key, value, context);

String string = value.toString();

String[] strings = StringUtils.split(string,' ');

for (String s : strings) {

context.write(new Text(s),new IntWritable(1));

}

}

}

public class WordCountReducer extends Reducer<Text,IntWritable,Text,IntWritable>{

protected void reduce(Text key, Iterable<IntWritable> values, Context context) throws IOException, InterruptedException { 执行Reduce任务的类需要继承Reduce类重写reduce方法

super.reduce(key, values, context);

int sum = 0;

for (IntWritable i : values) {

sum += i.get();

}

context.write(key,new IntWritable(sum));

}

}

public static void main(String[] args) throws Exception{

Configuration configuration=new Configuration();

Job job=Job.getInstance(configuration); 获取任务Job实例对象

job.setJarByClass(WordCountApp.class); 指定任务的入口类

job.setMapperClass(WordCountMapper.class); 指定任务的Mapper类

job.setMapOutputKeyClass(Text.class); 指定Mapper输出的key类型

job.setMapOutputValueClass(IntWritable.class); 指定Mapper输出的value类型

job.setReducerClass(WordCountReducer.class); 指定任务Reducer类

job.setOutputKeyClass(Text.class); 指定Reducer输出的key类型

job.setOutputValueClass(NullWritable.class); 指定Reducer输出的value类型

job.setCombinerClass(WordCountReducer.class); 指定Combiner类

job.setPartitionerClass(TqPartition.class); 指定Partition类

job.setSortComparatorClass(TqSort.class); 指定排序类

job.setGroupingComparatorClass(TqGroup.class); 指定分组类

job.setNumReduceTasks(3); 指定Reduce的数量

FileInputFormat.addInputPath(job,new Path("")); 指定任务的输入数据文件

Path outPath=new Path("");

FileSystem fileSystem=FileSystem.get(configuration);

if (fileSystem.exists(outPath)){ 判断文件是否存在，已存在删除

fileSystem.delete(outPath,true);

}

FileOutputFormat.setOutputPath(job,new Path("")); 指定任务的输出文件，输出文件已存在会出现错误

boolean flag = job.waitForCompletion(true); 等待判断任务是否执行完成

if (flag) {

System.out.println("job success!");

}

}

}

MapReduce任务执行分为本地测试环境（使用Java多线程模拟任务执行）和服务器环境（程序提交至Hadoop集群执行）

本地测试环境（Window）：

Windows下配置Hadoop的环境变量，复制debug工具（winutils.exe）到hadoop安装目录/bin目录下，修改相应的Hadoop源码，src目录下删除hadoop的配置文件，手动指定配置

configuration.set("fs.defaultFS","HDFS活跃主机的RPC地址"); 手动指定HDFS活跃主机的RPC地址

configuration.set("yarn.resourcemanager.hostname","ResourceManager活跃的主机名或IP地址"); 手动指定ResourceManager活跃的主机名或IP地址

服务器环境：

本地提交服务器hadoop集群执行任务：

src目录下放置服务器的hadoop配置文件（\*-site.xml），将MapReduce任务程序打jar包，修改hadoop源码，手动指定jar包位置configuration.set("mapred.jar","路径名");

服务器命令执行任务：

src目录下放置服务器的hadoop配置文件（\*-site.xml），将MapReduce任务程序打jar包上传至服务器，使用hadoop jar jar包路径名 MapReduce任务入口类的全类名执行任务

在Maven配置中指定MapReduce任务入口类的全类名后可以直接使用hadoop jar jar包路径名执行MapReduce任务：

<build>

<pluginManagement>

<plugins>

<plugin>

<groupId>org.apache.maven.plugins</groupId>

<artifactId>maven-jar-plugin</artifactId>

<configuration>

<archive>

<manifest>

<mainClass>MapReduce任务入口类的全类名</mainClass>

<addClasspath>true</addClasspath>

<classpathPrefix>lib/</classpathPrefix>

</manifest>

</archive>

</configuration>

</plugin>

</plugins>

</pluginManagement>

</build>

MapReduce程序中使用自定义数据类型的类进行处理，自定义数据类型的类需要实现WritableComparable<泛型>接口，一般需要重写序列化的write方法和反序列化readFields方法以及compareTo比较方法，MapReduce计算框架默认调用compareTo方法实现比较排序

自定义Partition类需要继承HashPartitioner重写getPartition方法，方法处理内容尽量简单，因为每个Mapper与Reducer都会执行分区方法，分区方法影响MapReduce程序的效率

自定义Sort排序类需要继承WritableComparator类，重写构造方法super(数据类型的类,true);和比较方法compare

自定义Group分组类需要继承WritableComparator类，重写构造方法super(数据类型的类,true);和比较方法compare

PageRank是Google的算法，用于计算衡量特定网页相对搜索引擎索引中的其他网页而言的重要程度，由Google创始人拉里佩奇和谢尔盖布林于1997年创造，PageRank实现了将链接价值概念作为排名因素

PageRank可以通过站长工具进行查询

PageRank算法原理：

入链（投票）：其他网页指向本网页的链接，到达指定网页的链接相当于为指定网页投一票

出链：本网页指定其他网页的链接

入链数量：网页收到其他网页链接的数量

入链质量：质量高的网页通过链接向其他页面传递更多的权重

每个页面的PageRank初始值相同（Google的PageRank算法初始值为1），不断重复的迭代递归计算每个页面的PageRank值会趋于一个稳定的值（收敛），最终理想状态每个页面的PageRank值与上次计算的PageRank值相等，由于无法达到理想状态，一般设定一个差值标准（即每个页面的PageRank值与上次计算的PageRank值相差小于设定的差值标准则认为收敛），设定一个百分比（即百分比的页面满足条件则认为收敛）

由于存在出链为0即存在不链接其他网页的页面（孤立网页）但入链不为0，因此修改PageRank计算公式，加入阻尼系数（Damping Factor）q，q一般取值为0.85

修正后的PageRank计算公式：网页总数N