公司技术分享之Hadoop

1. hadoop的历史（可以讲创建的历史，也可以提一下于谷歌的论文的关系，这里又可以讲讲谷歌创始人求职的经历）

Hadoop起源于Apache Nutch，Nutch是一个开源的java语言来实现的搜索引擎。

2004年，Cutting和同为程序员出身的Mike Cafarella决定开发一款可以代替当时的主流搜索产品的开源搜索引擎，这个项目被命名为Nutch。它是模仿google搜索引擎创立的开源搜索引擎，后归于apache旗下

nutch主要完成抓取，提取内容等工作。和solr一样，都是搜索引擎，并且都是基于lucene的。支持全文搜索和爬虫。

但是遇到了一个难题是：nutch抓取海量数据怎么存储和处理呢？

 当时Google的一篇论文<Google File System>

所以Cutting和Mike设计了HDFS->Hadoop Distributed File System

又根据Google的另一篇论文<Google Mapreduce> 设计了Hadoop的MapReduce

所以Hadoop框架两个核心模块:①HDFS(存储数据)+②MapReduce(计算数据)

说起Google，Cutting也是它成长的见证人之一，这里有一段鲜为人知的故事。早在Cutting供职于Architext期间，有两个年轻人曾去拜访这家公司，并向他们兜售自己的搜索技术，但当时他们的Demo只检索出几百万条网页，Excite的工程师们觉得他们的技术太小儿科，于是就在心里鄙 视一番，把他们给送走了。但故事并未到此结束，这两个年轻人回去之后痛定思痛，决定自己创业。于是，他们开了一家自己的搜索公司，取名为Google。这两个年轻人就是Larry Page和Sergey Brin。在Cutting看来，Google的成功主要取决于，反向排序之后再存储的设计和对自己技术的自信。

1. Hadoop的原理，这里可以讲hdfs和mapreduce的原理；

HDFS概述

HDFS为了保证数据存储的可靠性和可读性，对数据进行切块后进行复制并存储在集群的多个节点中。

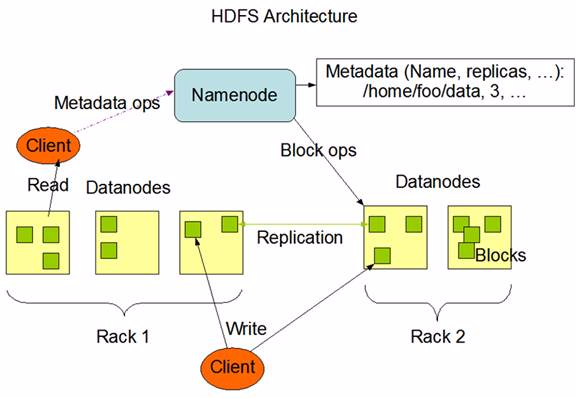
HDFS存在一个名字节点namenode和多个数据节点datanode

当客户端想要往HDFS储存文件的时候，文件会被切成很多块，并把文件块发往各个Datanode，而每个文件块存放到了哪个节点，每个文件块对应的是哪个文件。这些信息都是由namenode来存储的

所以namenode也叫名字节点，存储的是元数据信息。可以把hdfs想象成仓库

元数据存储在内存中，查询速度很快。但是为了容灾，元数据也会存在磁盘中。

计算机生成了可选文字:
NameNode 
0 
1. 要 知 道 哪 些 DateNode 节 点 
2 ． 管 理 了 哪 些 大 文 件 ， 并 且 要 知 道 这 个 大 文 件 被 分 割 后 ， 都 存 储 
到 了 哪 点 上 
3 ． 分 布 式 文 件 系 统 有 吝 灾 机 制 ， 比 如 节 点 故 障 引 起 的 数 据 丢 失 ， 针 对 这 种 场 
景 ， 我 们 应 该 做 好 数 据 备 份 工 作 ， 一 般 来 说 ， 一 个 文 件 被 分 成 b k 块 后 ， 每 
个 块 都 应 该 有 备 份 ， 常 见 的 是 三 分 （ 两 备 ） ， 分 别 是 0 〕 自 己 机 器 上 有 一 份 
0 同 机 架 上 的 某 台 节 点 上 有 一 一 份 0 不 同 机 架 上 节 点 有 一 份 ， 所 以 ， 
namenode 还 要 知 道 每 个 block 备 份 信 息 ， 如 果 某 台 机 器 出 现 故 障 ， 还 能 从 其 
他 节 点 找 到 。 
4.NameNodefi 理 Date № de ， 如 何 得 知 每 个 Date № de 的 状 态 ？ Hadoop 是 通 过 
心 跳 机 制 ， 就 是 ateNode 通 过 Avro 的 Rpc, 每 隔一 段 时 间 ， 像 Name № de 发 送 
一 次 心 跳 包 ， 如 过 了 一 定 时 问 之 后 № me № de 没 有 收 到 心 跳 包 ， 则 证 明 
DateNodei$f 
机 架 1 
HDFS 分 布 式 文 件 系 统 的 特 点 ： 
上 2 、 3 、 4 特 点 已 列 举 
5 ． 简 化 的 一 一 致 性 模 型 
0 飞 0 
HDFS 的 缺 点 ： 
不 能 做 到 低 延 迟 的 数 据 访 问 ， 受 限 于 磁 盘 和 网 络 
4GB 
机 架 2 
IGB 
DateNode 
对 于 外 部 使 用 用 户 ， 不 需 要 了 解 had 灬 p 底 层 细 节 ， 比 如 文 件 的 切 块 ， 文 件 
的 存 储 ， 节 点 的 管 理 
一 个 文 件 存 储 在 HOFS 上 后 ， 适 合 一 次 写 人 ， 多 次 读 取 的 场 景 。 因 为 存 储 在 
HOFS 上 的 文 件 都 是 超 大 文 件 ， 当 上 传 完 这 个 文 件 到 hadoop 集 群 后 ， 会 进 行 
文 件 切 块 ， 分 发 ， 复 制 等 操 作 。 如 果 文 件 被 修 改 ， 会 导 致 重 新 出 发 这 个 过 程 
， 而 这 个 过 程 耗 时 是 最 长 的 。 所 以 在 hadoop 里 ， 阊 版 本 
允 许 数 据 的 追 加 ， 单 不 允 许 数 据 的 修 改 。 



1.namenode 管理元数据

2.Metadata 元数据.在namenode的内存中存储,此外,会通过Fsimage文件和Edits来存储.每次HDFS启动时,namenode都会从Fsimage文件中加载到内存中

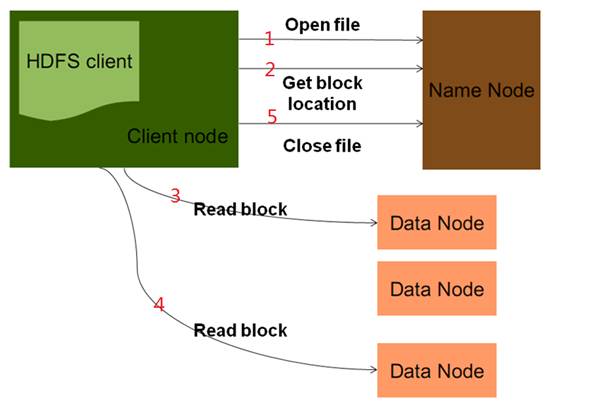
3.Blocks 文件块.HDFS文件系统的存储特点是块存储

4.Replication 副本.HDFS引入副本冗余机制,来实现数据存储的可靠性.注:副本数量不易过多,因为会使得整个集群的磁盘利用多低.比如:3副本策略,磁盘利用率就是1/3

5.datanode,存储文件块的

6.rack 机架

7.Client 凡是同时指令或通过代码操作的一端,都属于客户端.底层都是通过DFSClient这个类来操作,通过rpc接口和namnode或datanode通信



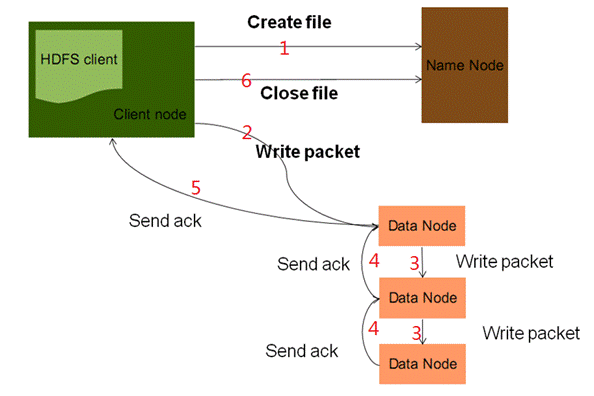
1.DFSClient 通过namenode获取指定文件的输入流.namnode会做检验.比如检验路径的合法性,以及读取的权限.如果检验没通过,直接报错,返回给客户端.

2.DFSClient会通过namenode获取文件的元数据信息,如果第一步的检验通过的话,则namenode会将文件的元数据信息封装到输入流返回客户端.

3.4 客户端通过输入流,去对应的Datanode读取文件块

注:如果文件块数量过多,DFSClient会多次调用GetBlockLocation去获取文件块信息.

1. 读取完成后,close file,关流.



.DFSClient获取HDFS的文件输出流,namenode收到请求后,会做校验,比如路径的合法性以及是否具有写权限.如果未通过,直接报错返回.

如果校验通过,namenode会根据文件的信息,做文件切块的分配,比如每个文件块存到哪个datanode上.然后把分配的信息封装输出流返回给客户端.

2.3.4.5 DFSClient 通过输出流,将文件块打散成一个一个的packet进行发送方.每个packet是64kb大小.

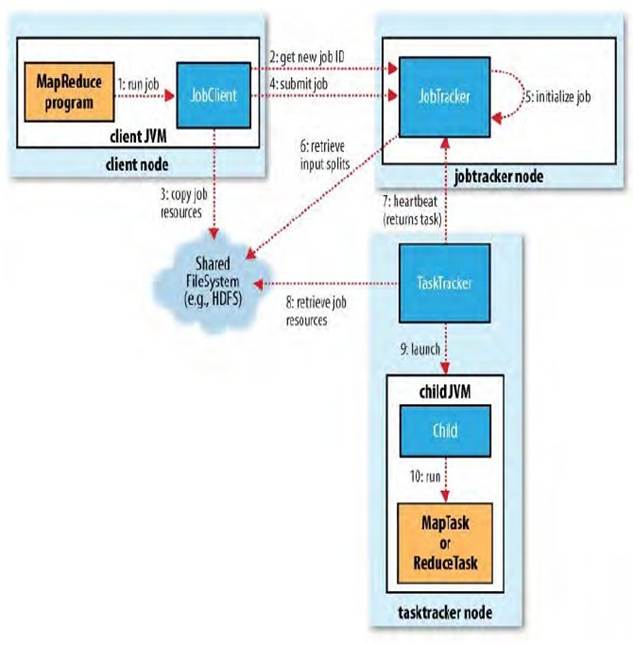
把每个packet通过精心选择的服务器进行写出.

写出到指定的Datanode上,Datanode上会将数据发送数据链上给其他的Datanode.

此外,每个Datanode收到数据后,会向上游反馈ack确认机制.直到最上游的Datanode向DFSClient发送ACK之后,再次发送下一个文件块.

注:上述过程中,有一个专业的术语:数据流管道(Pipeline),目的是为了充分利用每台服务器的带宽,最小化数据传输的延迟.

此外,在底层,会用到全双工通信来做数据的传输.



1.run job 启动job。收集job环境信息，比如输入路径、输出结果路径等信息。此外也会检查路径是否正确，如果有问题，就会抛异常，并终止job的后续提交。

2.get new job ID。向jobTracker申请一个新的jobid。jobid的作用一是起到唯一标识作用，二是程序员可以根据jobid 查询job,以及kill job。

指令:Hadoop job -kill jobid

3.JobTracker分配给一个jobid后，告诉JobClient，把job的运算资源（简单理解为jar包）上传到HDFS上。

4.JobClient 执行submit job，在这个过程中，JobClient把jobid，以及jar包在HDFS上的存储路径提交给jobTracker。

5.JobTracker收到相关任务信息后，开始进行Job的初始化。

6.根据分片数量以及用户设置的分区数量，计算出有多少个map任务和reduce任务。

7.TaskTracker通过RCP心跳领到JobTracker分配的任务。

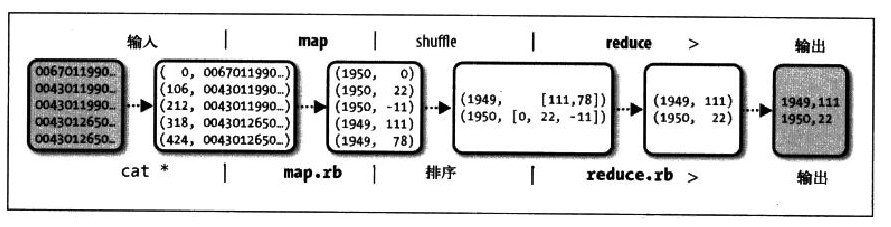
8.TaskTracker根据任务信息，去HDFS上找到相关的jar包，并下载到自己的节点上。

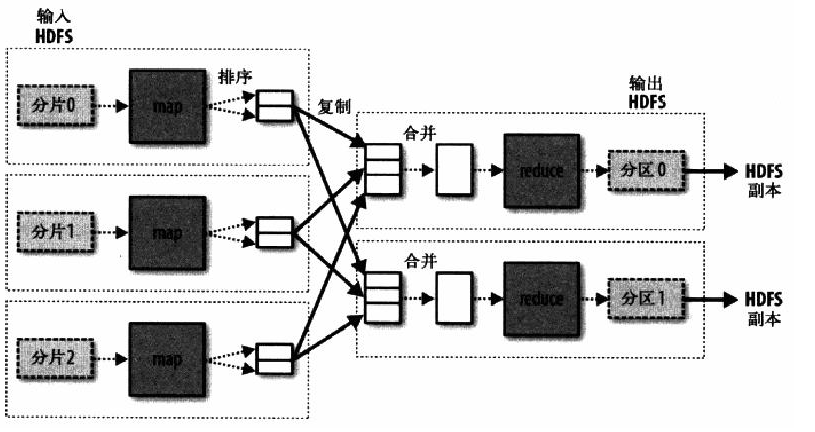
9.启动JVM

10.执行map任务或reduce任务

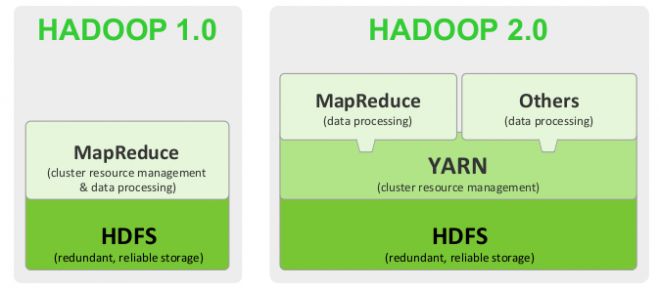
注意：Hadoop任务分派的策略：数据的本地化策略。

Hadoop里有个思想：移动的是运算，而不是数据（jar包存储在某一个datanode节点上，就相当于存储在HDFS上）





1. 讲一下hadoop1和hadoop2的区别，并且讲解一下hadoop较慢的原因。



1. 实际操作一下hadoop的环境搭建以及做一个单词demo；

案例1：

（需求：查找文件中单词出现次数。

    文件（word.txt）)

hello world

hello hadoop

hello 1606

hello world

代码（Mapper）：

public class WordCountMapper2 extends Mapper<LongWritable,Text, Text,IntWritable>{

@Override

protected void map(LongWritable key, Text value, Mapper<LongWritable, Text, Text, IntWritable>.Context context)

throws IOException, InterruptedException {

//--做Text的类型转换,目的是使用String提供的API,比如切分等操作

String line=value.toString();

String[] words=line.split(" ");

for(String word:words){

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

}

}

}

代码（Reducer）：

public class WordCountReducer2 extends Reducer<Text,IntWritable,Text,IntWritable>{

@Override

protected void reduce(Text key, Iterable<IntWritable> values,

Reducer<Text, IntWritable, Text, IntWritable>.Context context) throws IOException, InterruptedException {

int count=0;

for(IntWritable value:values){

count=count+value.get();

//count++

}

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

}

}

代码（Driver）：

public class Driver {

​

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

//--Hadoop的环境参数对象

Configuration conf=new Configuration();

//--获取MR job任务对象

Job job=Job.getInstance(conf);

//--设置运行主类

job.setJarByClass(Driver.class);

//--设置Mapper组件类,底层是通过反射获取的

job.setMapperClass(WordCountMapper2.class);

//--设置Mapper输出key的类型,一定要对应上

job.setMapOutputKeyClass(Text.class);

//--设置Mapper输出Value的类型.注意:导包导的是hadoop.io包

job.setMapOutputValueClass(IntWritable.class);

//--设置Reducer组件

job.setReducerClass(WordCountReducer2.class);

//--设置Reduce输出key类型

job.setOutputKeyClass(Text.class);

//--设置Reduce输出value类型

job.setOutputValueClass(IntWritable.class);

//--设置处理的文件的HDFS路径.

//--路径可以写到目录级别 或文件级别

//--如果路径目录,MR会处理指定目录下的所有文件.所需需要注意的是:格式的问题

FileInputFormat.setInputPaths(job,

new Path("hdfs://192.168.195.141:9000/wordcount"));

//--设置输出的结果目录.

FileOutputFormat.setOutputPath(job,

new Path("hdfs://192.168.195.141:9000/wordcount/result"));

//--提交job

job.waitForCompletion(true);

}

}

​

安装步骤

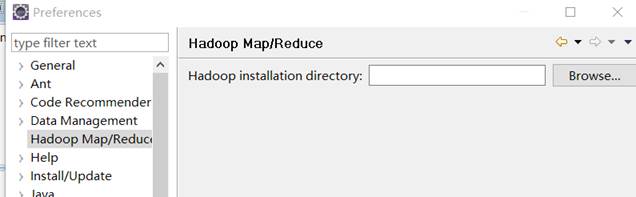
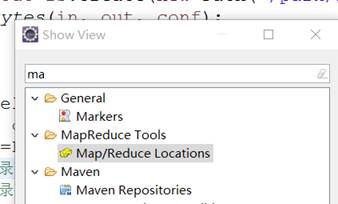
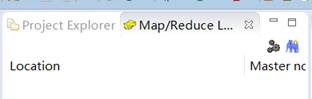
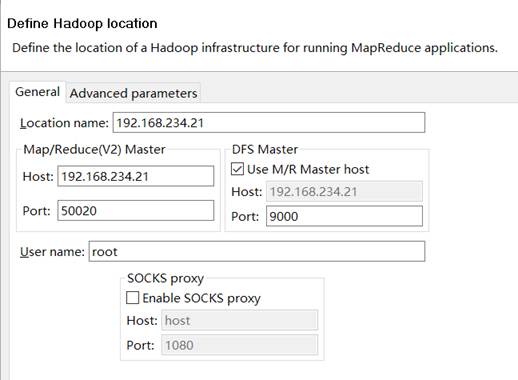
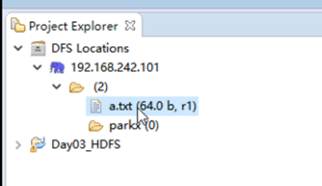
1.下载hadoop插件，注意:插件的版本要和用的hadoop版本保持一致



2.将插件jar包放在eclipse安装目录的plugins目录下

3.将hadoop安装包解压到指定的一个目录（后面要用这个安装目录）



1. 重启eclipse，windows=>下发现多出Map/Reduce选项卡，点击=》选择hadoop的安装目录，然后点击apply，点击确定
2. 
3. 点击open perspective，调出map/reduce视图，或者show view ，调出map/reduce 视图
4. 在map/reduce视图下，点击蓝色的大象，新建hadoop客户端连接
5. 
6. 在下面的选项卡里，填好namenode节点的ip地址，及相应的端口号
7. 
8. 
9. hadoop的权限修改
10. 每次更改文件，可能都需要以root用户登录，或伪装成root用户，这样比较麻烦
11. 这个可以配置，在hdfs-site.xml来配置：
12. 属性值改成false即可
13. 
14. Hadoop的应用，缺点，优点，并且和spark做一个比较。

spark 运行流程：

Spark架构采用了分布式计算中的Master-Slave模型。Master是对应集群中的含有Master进程的节点，Slave是集群中含有Worker进程的节点。

Master作为整个集群的控制器，负责整个集群的正常运行；

Worker相当于计算节点，接收主节点命令与进行状态汇报；

Executor负责任务的执行；

Client作为用户的客户端负责提交应用；

Driver负责控制一个应用的执行。

Spark集群部署后，需要在主节点和从节点分别启动Master进程和Worker进程，对整个集群进行控制。在一个Spark应用的执行过程中，Driver和Worker是两个重要角色。Driver 程序是应用逻辑执行的起点，负责作业的调度，即Task任务的分发，而多个Worker用来管理计算节点和创建Executor并行处理任务。在执行阶段，Driver会将Task和Task所依赖的file和jar序列化后传递给对应的Worker机器，同时Executor对相应数据分区的任务进行处理。

Excecutor /Task 每个程序自有，不同程序互相隔离，task多线程并行

集群对Spark透明，Spark只要能获取相关节点和进程

Driver 与Executor保持通信，协作处理