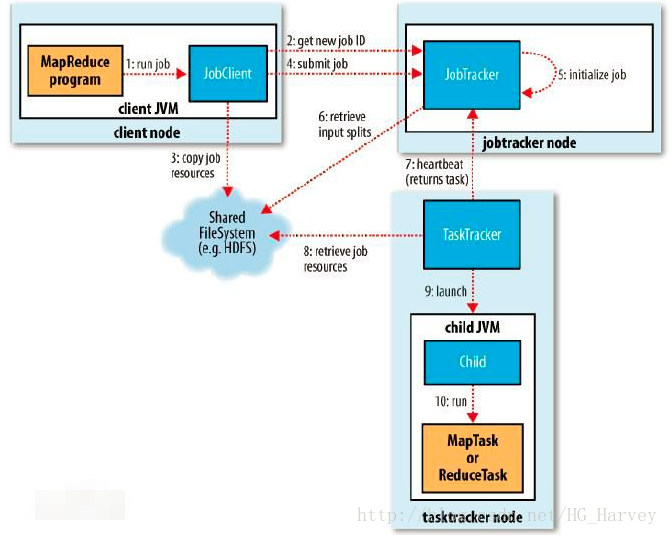
# MapReduce的原理和机制

## Mapreduce1.0工作原理

主要由三个实体组成：

* 客户端：用于提交Mapreduce作业
* JobTracker：协调作业的运行，JobTracker是一个java的应用程序，它的主类是JobTracker
* TaskTracker：运行作业划分后的任务，TaskTracker是一个java应用程序，主类是TaskTracker
* 分布式文件系统（HDFS）：用来在其他实体之间共享作业文件



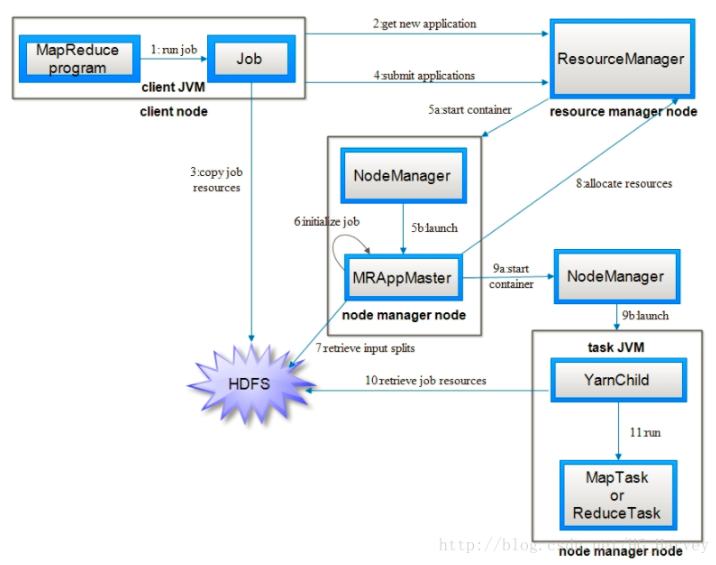
流程：

1. 客户端通过submit（）方法提交作业
2. Submit（）方法会创建一个内部的JobSummiter实例，并且调用submitJobInternal（）方法提交作业，JobSummiter向JobTracker请求一个新的作业ID，该实例会检查本次作业是否可执行（如：检查输出路径，计算作业的输入分片等），如果可执行会将运行所需要的作业资源（jar，配置文件等）上传到文件系统（HDFS）
3. JobTracker接收到提交任务请求后，会放到一个内部队列里，交由作业调度器（JobSckeduler）进行调度并初始化任务，从文件系统中获取客户端已经计算好的输入分片（每个分片计算一个map），reduce任务数量由setNumReduceTask（）方法设置
4. TaskTracker会定期向JobTracker发送“心跳”，表明TaskTracker是否存活，同时“心跳”是两者之间的消息通道，当TaskTracker空闲后，会通过“心跳”发送给JobTracker，然后JobTracker会为它分配任务
5. TaskTracker接受到JobTracker分配的任务后，从分布式文件系统中JAR文件复制到本地，开始运行任务，任务运行过程中TaskTracker会定期通过“心跳”会将任务的状态及运行情况告知JobTracker
6. JobTracker接收到最后一个任务已完成的通知后，Job会打印一条消息告知用户，然后从waitForCompletion（）方法返回，同时Job的统计信息及计数值也会打印到控制台

## MapReduce 2.0 工作原理

主要包括的实体：

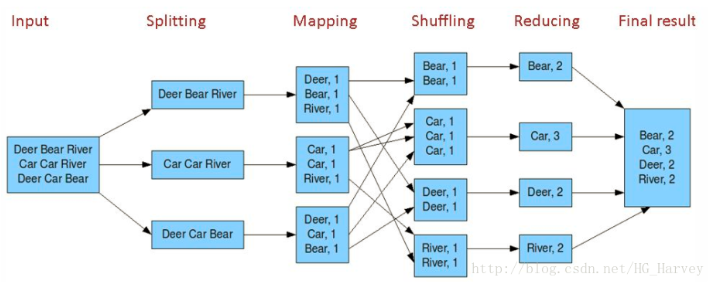
* 客户端：提交MapReduce作业
* ResourceManger：简称RM，负责协调集群上计算资源的分配
* NodeManger：简称NM，负责启动和监视集群中机器上的计算容器（Container）
* ApplicationMaster：简称AM，Yarn中每个应用都会启动一个AM，负责向RM申请资源，请求NM启动Container，并告诉Container做什么事情
* Container：资源容器，YARN中所有的应用都是在container之上运行的，AM也是在Container上运行的，不过AM的container是RM申请的



流程：

1. 步骤1,2和MapReduce1.0相似，不同的是作业ID是从资源管理器（RM）中获取，而不是jobtracker，在YARN中命名法中它是一个应用程序ID，最后调用submitApplication（）方法提交作业
2. RM收到作业请求后，便把请求传递给调度器，调度器分配一个容器，然后RM在NM（节点管理器）下的容器中启动应用程序的master进程
3. Master进程是一个java应用程序，它的主类是MRAPPMaster，它对作业进行初始化：通过创建多个薄记对象以保持对作业进度的跟踪，因为它将接受来自任务的进度和完成报告
4. Master接受来及HDFS在客户端计算的输入分片，对每个分片创建一个map任务对象以及由mapreduce.job.reduces属性确定多个reduce任务对象
5. 如果作业不适合在单个节点上运行，那么application.master就回为该作业中的所有map任务和reduce任务想RM请求容器，附着心跳信息的请求包括每个map任务的数据本地化信息，特别是输入分片所在的主机和相应机架信息，调度器使用这些信息来做调度决策，先数据本地化，再机架本地化
6. 一旦RM的调度器为任务分配了容器，application master就通过与NM（节点管理器）通信来启动容器，在运行任务之前，首先将任务所需要的资源本地化，包括作业配置，jar文件和所有来自HDFS（分布式文件系统）缓存的文件，然后执行map任务或reduce任务

## MapReduce中wordcount



Map阶段：

1. 将一个输入作为一个映射，即key：value（注：文件中的每行会执行一次map，针对words.txt文件中的内容会执行三次map，以文件中的第一行为例）



1. 处理value，拆分成独立的单词数组（按空格分隔）



1. 当单词数组中的每个元素组成一个映射，即key：value

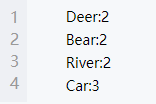


Reduce阶段：

1. 将map阶段产生所有key：value通过洗牌排序shuffle产生新的key：value



1. 遍历key，累加value值，最终结果如下



# Shuffle过程

## 基本概念

Shuffle是洗牌、混洗，把一组有一定规则的数据尽量转成一组无规则的数据，越随机越好，MapReduce中的shuffle更像是洗牌的逆序过程，把一组无规则的数据尽量转成一组具有一定规则的数据

## 为什么需要shuffle过程

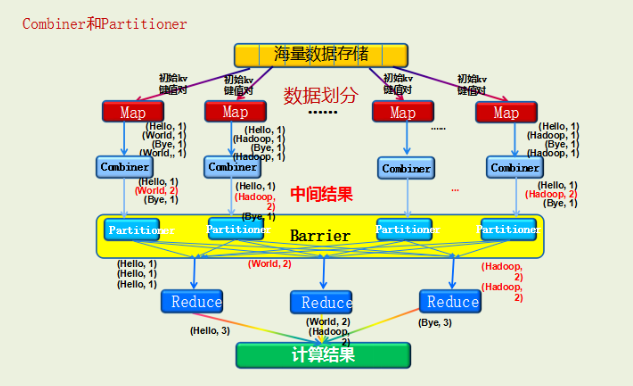
MapReduce计算模型包括两个重要的阶段：

Map映射：负责数据的过滤分发

Reduce规约：负责数据的计算归并

Reduce的数据来源于map，map的输出是reduce的输入，reduce需要从shuffle来获取数据。Map和reduce输出的格式都是一致，但是reduce的输入可能和map的输出不一样，这时候需要shuffle聚集处理，完成reduce的输入格式

广义来说：map的输出到reduce输入这段称shuffle过程，都hadoop默认执行，中间执行步骤combiner和partitioner，sort和merge



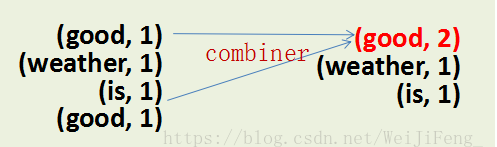
**Combiner**

* 一个可选的本地reducer，可以在map阶段聚合数据
* 可以明显减少通过网络传输的数据量
* 使用combiner可以产生特别大的性能提升，并没有副作用
* 不能保证执行，不能作为整个算法的一部分
* 并不是所有情况下都能使用combiner，适用于对记录汇总的场景（求和、但是求平均数的场景就不能使用了）

**Combiner对系统的优化：**

问题：大量的键值对数据在传送给reduce节点时会引起较大的通信带宽开销

解决方案：每个map节点处理完成的中间键值对将由combiner做一个合并压缩，即把那些键名相同的键值对归并为一个键名下的一组数值



**Partitioner：**

将mapper（如果使用了combiner就是combiner）输出的键/值对拆分为分片（shard），每个reduce对应一个分片

1. Partitioner组件可以让Map对Key进行分区，从而可以根据不同的key来分发到不同的reduce中去处理
2. 可以自定义key的一个分发规则，如数据文件包含不同的省份，而输出的要求是每个省份输出一个文件
3. 提供了一个默认的HashPartitioner

用数据分区解决数据相关性问题：

问题：

一个reduce节点上的计算数据可能会来自多个map节点，因此，为了在进入reduce节点计算之前，需要把属于一个reduce节点的数据归并到一起

解决方案：

在map阶段进行了combining以后，可以根据一定的策略对map输出的中间结果进行分区（partitioning），这样既可解决以上数据相关性问题避免reduce计算过程中的数据通信

例如：

有一个巨大的数组,其最终结果需要排序,每个Map节点数据处理好后,为了避免在每个Reduce节点本地排序完成后还需要进行全局排序,我们可以使用一个分区策略如:(d%R),d为数据大小，R为Reduce节点的个数，则可根据数据的大小将其划分到指定数据范围的Reduce节点上,每个Reduce将本地数据排好序后即为最终结果。

**Partitioner主要作用：**

将map的结果发送到相应的reduce，这就对partitioner有两个要求：

1. 均衡负载，尽量的将工作均匀的分配给不同的reduce
2. 效率，分配速度一定要快

## Shuffle过程的期望

* 完整地从map task端拉取数据到reduce端
* 在跨节点拉取数据时，尽可能地减少对贷款的不必要消耗
* 减少磁盘IO对task执行的影响

Shuffle描述着数据从map task输出到reduce task输入的这段过程

每个map task都有一个内存缓冲区，存储着map的输出结果，当缓冲区快满的时候需要将缓冲区的数据以一个临时文件的方式存到磁盘，当整个map task结束后在对磁盘中这个map task产生的所有临时文件做合并，生成最终的正式输出文件，然后等待reduce task来拉取数据

**Sort**

先把缓冲区中的数据按照partition值和key两个关键字升序排序，移动的知识索引数据，排序结果是Kvmeta中数据按照partition为单位聚集在一起，同一partition内的按照key有序

**Merge**

对于数据量非常大，map task的内存缓冲区不够，达到一定容量时候，就要把内存缓冲区的内容写入out文件。

对于很多out文件，此时merge就闪亮登场了

然后为merge过程创建一个交file.out的文件和一个叫file.out.Index的文件用来存储最终的输出和索引

根据file.out.Index文件和file.out文件提供out文件的路径和索引，支出之前内存缓冲区写出的out文件结果进行扫描

最后一个partition一个partition的进行合并输出，对于某个partition来说，从索引列表中查询这个partition对应的所有索引信息，每个对应一个段插入到段列表中。也就是这个partition对饮一个段列表，记录所有的spill文件中对应的这个partition那段数据的文件名、起始位置、长度等等。

# HDFS

## HDFS的存储机制

分为HDFS的**写入过程和读取过程**两部分：

写入过程：

1. 客户端向namenode请求上传文件，namenode检查目标文件是否已存在，父目录是否存在
2. Namenode返回是否可以上传
3. 客户端请求第一个block上传到哪几个datanode服务器上
4. Namenode返回3个datanode节点，分别为dn1，dn2，dn3
5. 客户端请求dn1上传数据，dn1收到请求会继续调用dn2，然后dn2调用dn3，将这个通信管道建立完成
6. Dn1，dn2，dn3逐级应答客户端
7. 客户端开始往dn1上传第一个block（先从磁盘读取数据放到一个本地内存缓存），以packet为单位，dn1收到一个packet就回传给dn2，dn2传给dn3；dn1每出传一个packet会放入一个应答队列等待应答
8. 当一个block传数完成后，客户端再次请求namenode上传第二个block的服务器（重复执行3-7步）

读取过程：

1. 客户端向namenode请求下载文件，namenode通过查询元数据，找到文件块所在的datanode地址。
2. 挑选一台datanode（就近原则，然后随机）服务器，请求读取数据。
3. datanode开始传输数据给客户端（从磁盘里面读取数据放入流，以packet为单位来做校验）。
4. 客户端以packet为单位接收，先在本地缓存，然后写入目标文件

## 什么是DFS，允许文件通过

分布式存储数据，通过网络将数据写到远程端数据块中，每一块数据都有多个副本

**特点**：通透性强：client，只注重数据写入读取方式，不关心实现原理

**优点**：

处理PB级及其以上数据量

注重数据吞吐量

处理非结构化数据

一次写入，多次读取

**缺点**：

不处理小文件

不支持结构化数据

不能对数据进行实时修改（可追加）

**常见的DFS**：HDFS、GFS、TFS（淘宝）

## HDFS的shell

Hadoop fs –ls 列出hdfs文件

Hadoop fs -ls imput 列出hdfs目录下某个文档中的文件

Hadoop fs –put ~/file test 上传文件到HDFS

Hadoop fs –get output getout将HDFS中文件复制到本地系统中

Hadoop fs –rmr newoutput 删除HDFS下的文档

Hadoop fs –cat input/\* 查看HDFS下某个文件

Hadoop dfsadmin –report 报告HDFS的基本统计情况

Hadoop dfsadmin –safemode leave退出安全模式

Hadoop dfsadmin –safemode enter 进入安全模式

**添加节点**

首先在新加节点上安装好Hadoop，要和NameNode使用相同的配置（可以直接从NameNode复制），修改**"/usr/hadoop/conf/master**"文件，加入NameNode主机名。然后在NameNode节点上修改"**/usr/hadoop/conf/slaves**"文件，加入新节点主机名，再建立到新加点无密码的SSH连接，运行启动命令：**start-all.sh**

**负载均衡**

HDFS的数据在各个DataNode中的分布肯能很不均匀，尤其是在DataNode节点出现故障或新增DataNode节点时。新增数据块时NameNode对DataNode节点的选择策略也有可能导致数据块分布的不均匀。

用户可以使用命令重新平衡DataNode上的数据块的分布：**start-balancer.sh**

## NameNode

整个文件系统的管理节点，接受用户操作请求，和datanode心跳机制

Namenode**主要构成元素**：

1. fsimage：元数据的镜像文件 是内存中的元数（保存在磁盘中），据序列化的到磁盘之后的文件名字
2. edits：记录操作过程
3. fstime：记录最近一次做还原点的时间

**工作特点**：

NameNode始终在内存中保存metedata,用于处理用户的读请求.写请求namenode会首先在写editlog日志,记录写入数据的位置,数量等信息,然后返回到客户端,同时hadoop会维护一个fsiamge文件,即是namenode中matedata的镜像,然后Secondary namenode会合并fisimage和edits文件来更新NameNode的metedata产生新的fsimage。

## SecondaryNameNode工作机制

**工作流程：**

**切换 下载 合并 发送**

**执行以上过程的时机有两点:**

**一是fistime中checkpoint大于3600秒**

**二是edits中数据量大于64MB**

第一阶段：**NameNode启动**

1. 第一次启动namenode格式化后，创建fsimage和edits文件。如果不是第一次启动，直接加载编辑日志和镜像文件到内存
2. 客户端对元数据进行增删改的请求
3. Namenode记录操作日志、更新滚动日志
4. Namenode在内存中对数据进行增删改查

第二阶段：Secondary NameNode工作

1. SecondaryNameNode询问namenode是否需要checkpoint。直接带回namenode是否检查结果。
2. SecondaryNameNode请求执行checkpoint。
3. namenode滚动正在写的edits日志
4. 将滚动前的编辑日志和镜像文件拷贝到Secondary NameNode
5. SecondaryNameNode加载编辑日志和镜像文件到内存，并合并。
6. 生成新的镜像文件fsimage.chkpoint
7. 拷贝fsimage.chkpoint到namenode
8. namenode将fsimage.chkpoint重新命名成fsimage

## NameNode与SecondaryNameNode的区别与联系

1. 机制流程同上：
2. 区别：
3. NameNode负责管理整个文件系统的元数据，以及每一个路径（文件）所对应的数据块信息
4. SecondaryNameNode主要用于定期合并命名空间镜像和命名空间镜像的编辑日志
5. 联系：
6. SecondaryNameNode中保存了一份和namenode一致的镜像文件（fsimage）和编辑日志（edits）
7. 在主namenode发生故障时（假设没有及时备份数据），可以从SecondaryNameNode恢复数据

## Datanode

提供真实文件数据的存储服务

主要元素：文件块（block）基本存储单位，hadoop2.0 默认大小128mb

## 元数据存储细节

HDFS为保证数据的安全性，会将元数据保存两份一份位于内存中的namenode节点中，另一份namenode的metadata存在磁盘中

在读取时，根据crc32校验和机制来判断该block中数据是否损坏

NameNode(FileName,replicas,block-ids,id2host....),

/test/a.log,3,{blk\_1,blk\_2},[{blk\_1:[h0,h1,h3]},{blk\_2:[h0,h2,h4]}])

存放路径，副本数量，切分文件，存放的具体的block数据块

## RPC=remote procedure Call 远程调用协议

不同进程之间方法调用，一个项目组中的两个方法，可能是一个进程在上海另一个在北京，可以通过RPC调用

## 常见题

* HDFS中的block默认保存：**3份**
* HDFS默认blocksize是：**128mb（2.7.2版本，分布式模式）**
* Hadoop1.x都是64M，hadoop2.x开始都是128M
* Client端上传文件的时候下列哪项正确？ **BC**

A 数据经过NameNode传递DataNode

B Client端将文件且分为Block，依次上传

C Client只上传数据到一台DataNode，然后由NameNode负责Block复制工作

* **DataNode** 负责HDFS数据存储
* 关于SecondaryNameNode哪项是正确的（C）

A 它是NameNode的热备

B 它对内存没有要求

C 他的目的是帮助NameNode合并编辑日志，减少NameNode启动时间

D SecondaryNameNode应与NameNode部署到一个节点

* **JobTracker** 通常与NameNode在一个节点启动
* hadoop的集群是基于master/slave模式，namenode和jobtracker属于master，datanode和tasktracker属于slave，master只有一个，而slave有多个。
* SecondaryNameNode内存需求和NameNode在一个数量级上，所以通常secondary NameNode（运行在单独的物理机器上）和 NameNode 运行在不同的机器上。
* JobTracker对应于NameNode,TaskTracker对应于DataNode。
* **DataNode和NameNode是针对数据存放来而言的。JobTracker和TaskTracker是对于MapReduce执行而言的。**

**mapreduce中几个主要概念**

mapreduce 整体上可以分为这么几条执行线索：

**jobclient，JobTracker与TaskTracker**。

1）JobClient会在用户端通过JobClient类将已经配置参数打包成jar文件的应用存储到hdfs，并把路径提交到Jobtracker,然后由JobTracker创建每一个Task（即 MapTask 和 ReduceTask）并将它们分发到各个TaskTracker服务中去执行。

2）JobTracker是一master服务，软件启动之后JobTracker接收Job，负责调度Job的每一个子任务。task运行于TaskTracker上，并监控它们，如果发现有失败的task就重新运行它。一般情况应该把JobTracker 部署在单独的机器上。

3）TaskTracker是运行在多个节点上的slaver服务。TaskTracker主动与JobTracker通信，接收作业，并负责直接执行每一个任务。TaskTracker 都需要运行在HDFS的DataNode上

* 文件大小默认为64M，改为128M有什么影响？

答：增加文件块大小，需要增加磁盘的传输速率

# Hive

## Hive的主要作用

Hive是基于hadoop的数据仓库工具，可以将结构化的数据文件映射成一张数据表，并且提供sql查询，相当于MapReduce的客户端

## 常见问题

1. Maven的离线仓库的默认位置是什么

检查离线仓库：导入这里需要的hbase和hadoop依赖的包

* 创建maven默认的离线仓库文件夹.m2（当前用户的home目录下）

$ mkdir ~/.m2/

* 解压离线仓库到默认位置

$ tar -zxf /opt/softwares/hbase+hadoop\_repository.tar.gz -C ~/.m2/

1. 配置hive-env.sh都设计到哪些属性

* 添加JAVA\_HOME路径     JAVA\_HOME=/opt/modules/jdk1.7.0\_67
* 添加HADOOP\_HOME路径  HADOOP\_HOME=/opt/modules/hadoop-2.5.0-cdh5.3.6/
* 添加HIVE\_COF路径   export HIVE\_CONF\_DIR=/opt/modules/hive-0.13.1-cdh5.3.6/conf

1. 配置hive-site.xml都修改了哪些属性，请写出属性名称并解释该属性

* 连接数据库的URL：javax.jdo.option.ConnectionURL
* 数据库驱动名：javax.jdo.option.ConnectionDriverName
* 数据库账户：javax.jdo.option.ConnectionUserName
* 数据库密码：javax.jdo.option.ConnectionPassword

1. 配置Mysql时，在CentOS6以及CentOS7中开启Mysql服务的命令分别是什么？

* centos7 1.systemctl start mysqld.service
* centos6 2.service mysql start/stop

1. Mysql中如何修改root密码？

不用先登录mysql：

mysqladmin -uroot -p123456 password 123

首先登录MySQL：

格式：mysql> set password for 用户名@localhost = password('新密码');

mysql> set password for root@localhost = password('123');

1. Hive的两个重要参数是什么？

* hive -e ‘’ 从命令行执行指定的HQL
* hive -f \*.hql 执行hive脚本命令