一、MapReduce

1.MapReduce计算模型介绍

- 解决海量计算

1.1 理解思想

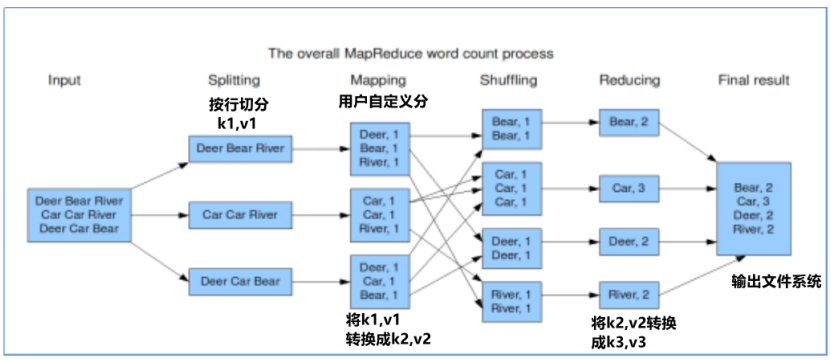
- 核心: 先分再合,分而治之
  - 把复杂的问题按一定的'分解'方法分为规模较小的若干部分,然后逐个解决,分别找出个部分的解,再把把各部分的解组成整个问题的解
- 思想分为两步:
  - map负责"分" : 将大的任务拆分成若干个小任务 -- 目的是并行处理提高效率
    - 可以拆,拆完之后没有依赖,每个都是局部的小结果
  - reduce 负责"合" : 将map阶段的结果进行全局汇总

1.2设计构思

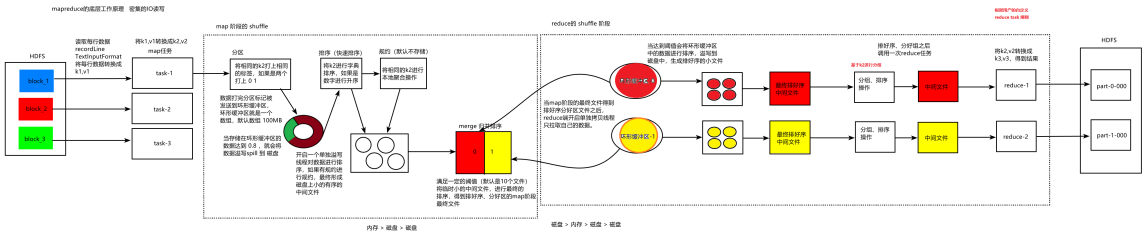
- 核心功能:
  - 将用户编写的业务逻辑代码和自带默认组件整合成一个完整的分布式运算程序
- 分而治之
- 能让用户提供业务的接口
- 统一构架,隐藏系统层细节
  - 如数据存储、划分、分发、结果收集、错误恢复等细节底层代码封装
  - 精准的把技术问题和业务问题区分,技术是通用的,业务不通用的
  - hadoop实现了底层所有的技术问题 --->80%代码 怎么做(how to do)
  - 用户实现业务问题 ---->20%代码 做什么(what need to do)

-例子:单词统计

wordcount 单词统计的案例
三大阶段八大步骤:
一: map阶段
1.1.从文件中读取每行的数据,
得到 k1,v1 => (偏移量,行数据)
(0,“Deer Bear Driver”)
1.2.将k1,v1 map转换成k2,v2
将v1这行数据通过 空格 进行Split切分
将切分出来的(值,1)作为 k2,v2
二: shuffle 阶段 (洗牌、混洗)
2.1 分区 - 默认 Hash 分区
2.2 排序 - 默认 字典排序
2.3 规约 - combine默认不存在
2.4 分组 - 基于k2 进行分组
三: reduce 阶段 - 合
3.1 根据用户自定义规则对数据进行聚合
3.2. 将k3,v3 输出到文件系统中



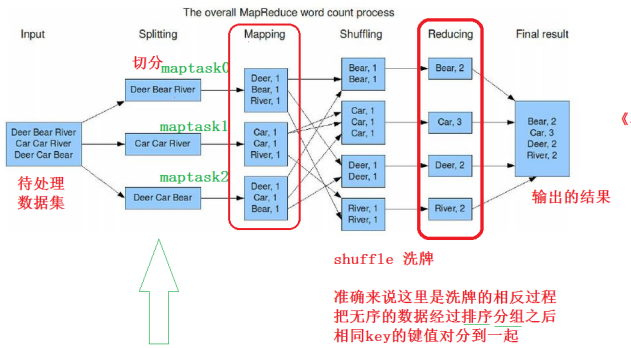
2.MapReduce基本原理



分而治之
先分
再合

map阶段的处理逻辑：
根据分隔符切割内容变成一个个单词
然后把每个单词标记1 变成kv键值对形式《单词，1》

reduce阶段的处理逻辑：
因为经过shuffle之后相同单词已经分为一组
只需要把对应的value进行累加即可
输出的结果就是该单词出现的总次数



《单词，总次数》

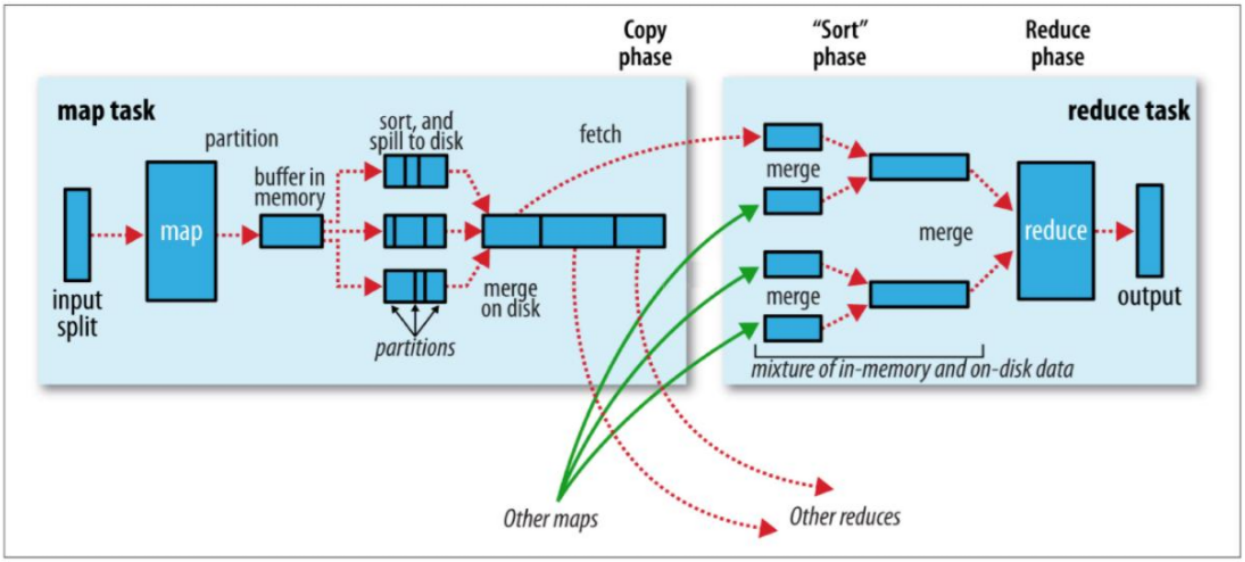
shuffle 洗牌

准确来说这里是洗牌的相反过程
把无序的数据经过排序分组之后
相同key的键值对分到一起

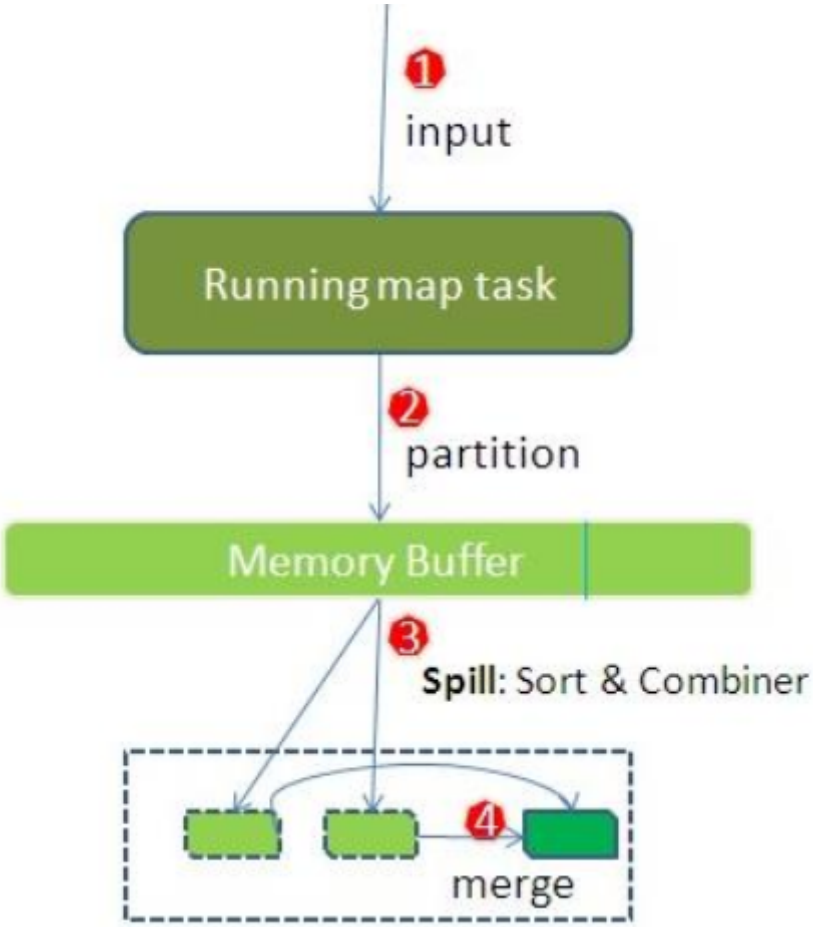
为什么进行切分拆分？
拆分之后的数据没有依赖的话
可以通过并行计算提高处理效率

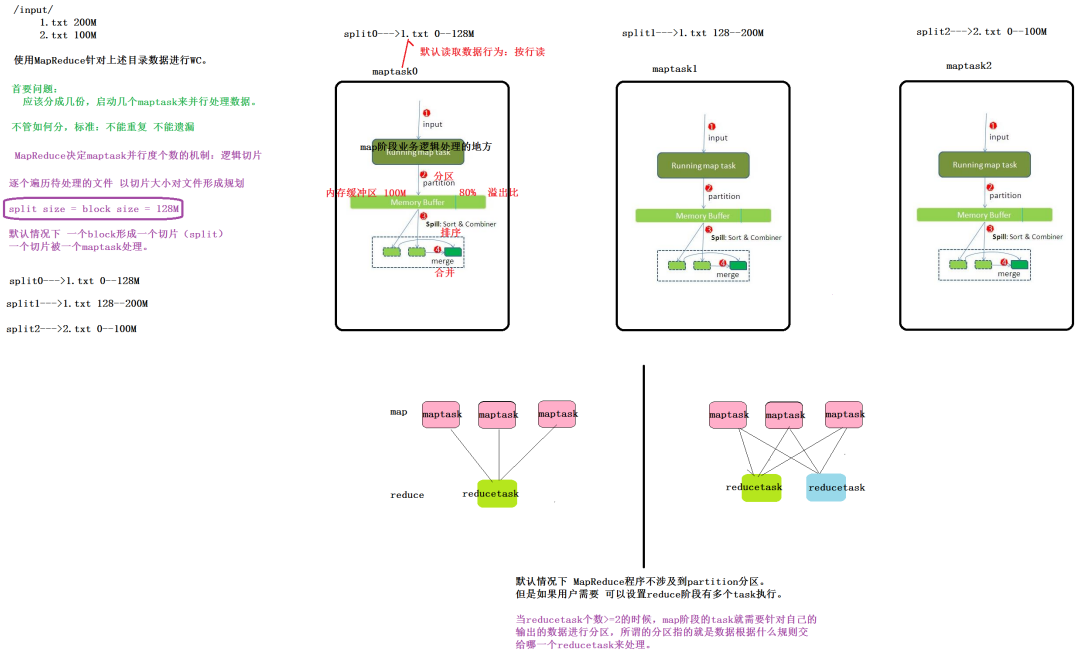
2.1MapReduce输入和输出

- 数据都是以<key,value>键值对的形式存在的,不管是输入还是输出
- 抽象函数式编程的模型:map 和 reduce
- 关于inputpath
  - 指向的是一个文件,mr就处理这一个文件
  - 指向的是一个目录,mr就处理该目录下所有的文件,整体当做数据集处理
- 关于outputpath
  - 要求制定的目录为空目录,不能够存储,否则执行效验失败
  - FileAlreadyExistsException: Output directory file:/D:/datasets/wordcount/output already exists
- 整体流程

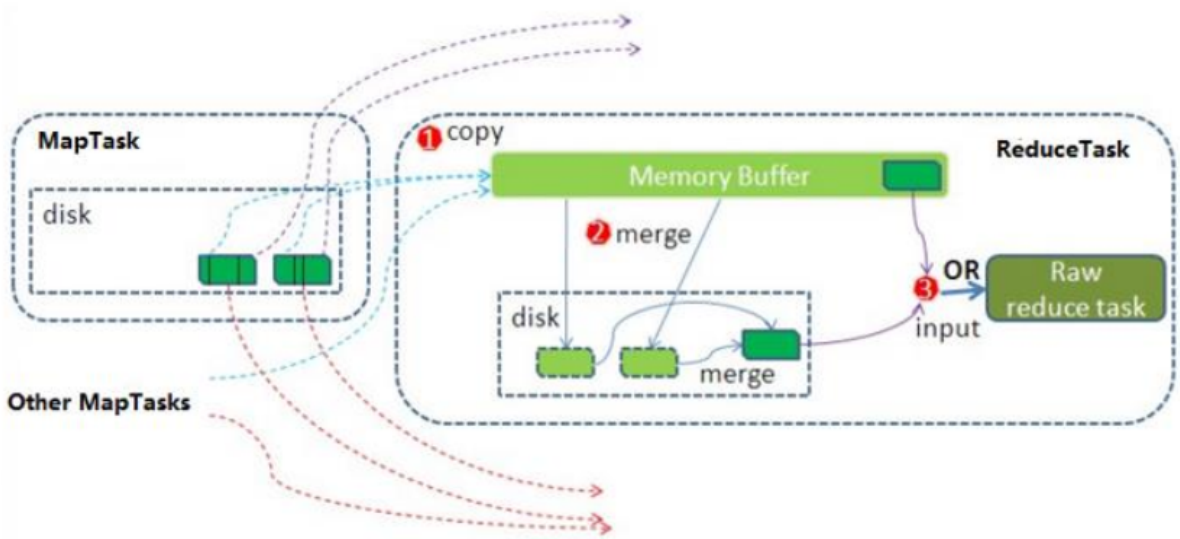

Copy phase
"Sort" phase
Reduce phase
map task
reduce task
partition
sort, and spill to disk
fetch
buffer in memory
map
input split
merge on disk
partitions
merge
merge
reduce
output
mixture of in-memory and on-disk data
Other maps
Other reduces

map阶段流程


1
input
Running map task
2
partition
Memory Buffer
3
Spill: Sort & Combiner
4
merge



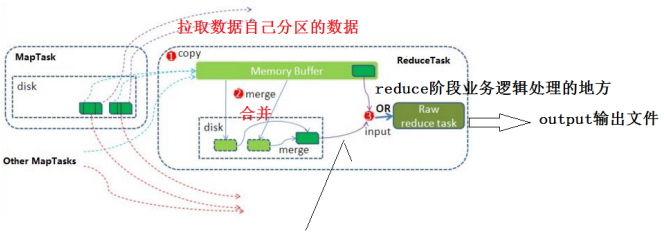
- maptaskbk并行读度个数机制:逻辑切片机制
- 影响maptask个数的因素有
  - 文件的个数
  - 文件的大小
  - split size=block size 切片的大小受数据块的大小控制
- reduce阶段流程



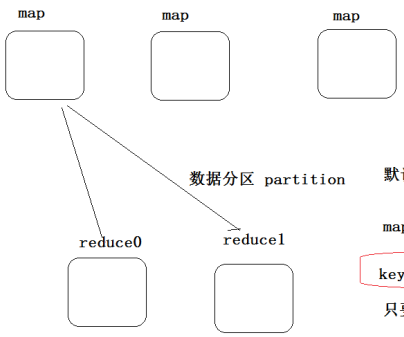
map阶段执行完毕之后，每个结果都是局部的结果。
因此需要进行reduce全局聚合得出最终的结果。

默认情况下：全局只有一个reducetask进行数据聚合。
当然也可以根据需求设置为多个。（多个就意味着分区）

1、因为maptask处理完数据之后并不会将数据结果推送给reducetask，使得实际上reducetask工作的第一个步骤：
就是去各个maptask那里把属于自己分区的数据拉取回来。



在MapReduce框架的内部
会对数据进行分组和排序的动作
保证的相同的数据在一组去处理。



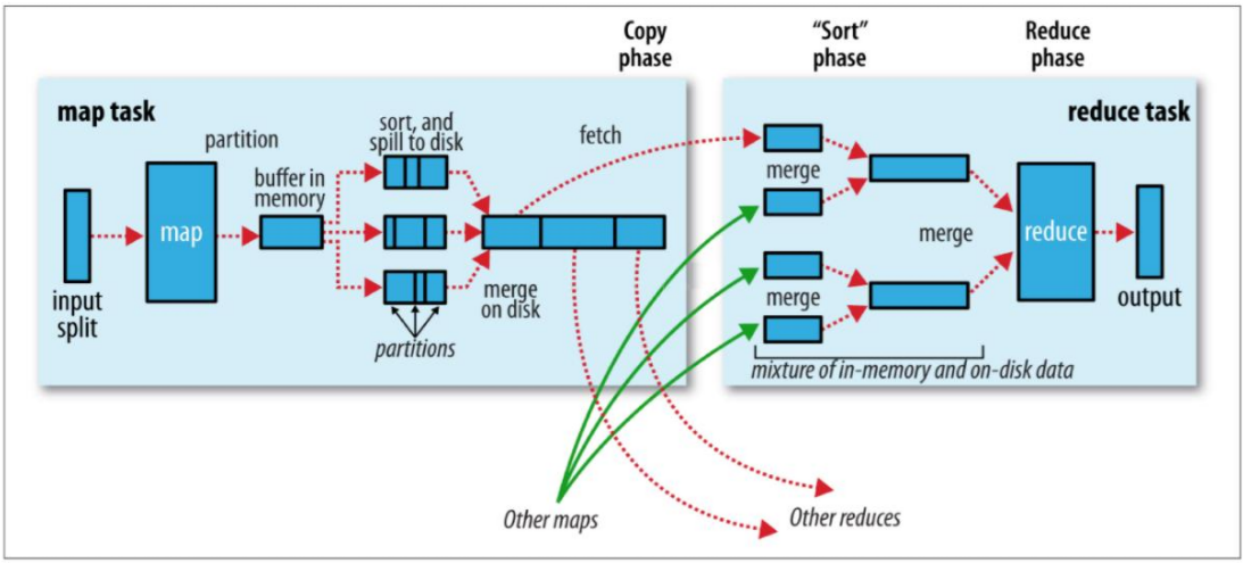
默认的分区规则 HashPartitioner

maptask输出的结果 《key,value》

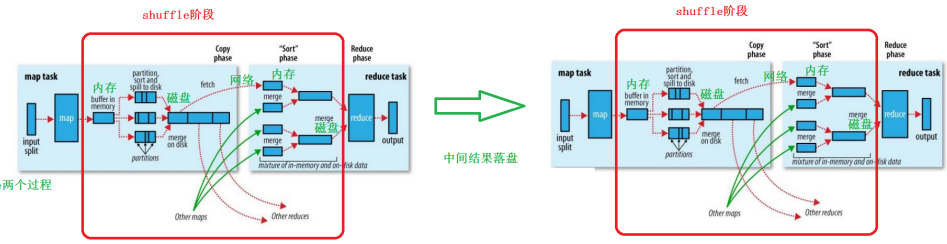
key.hashcode % reducetask个数 = 余数（分区编号数）

只要输出的key一样 数据就会到同一个分区

- 影响reducetask个数的因素
- 只要用户不设置永远默认1个
- 用户也可以通过代码进行设置,设置为几就是几
  - 当reducetask>=2,数据就会分区了
- shuffle机制



Map产生输出开始到Reduce取得数据作为输入之前的过程称作shuffle。



1、shuffle阶段是一个横跨map、reduce两个过程

又细分为

map端的shuffle

reduce端shuffle

2、shuffle是MapReduce核心精髓所在

也是MapReduce被诟病最多的地方所在

步骤繁琐、频繁的内存和磁盘之间的数据IO

3、当业务需求需要经过多个MapReduce程序才能完成计算的时候

中间的结果会落盘操作 使得计算效率低

比如spark把中间结果放在内存中 不落盘

MR程序组成、提交执行

- 组成
  - 从运行的角度看MR程序 -- MR中Task都是进程级别的(Java进程)
    - maptask
    - reducetask
    - MrAppMaster

MR执行流程

- 三大阶段八大步骤
  - map阶段
    - 读取文件数据,将其转换成k1,v1
    - 通过用户自定义map任务,将k1,v1转换成k2,v2
  - shuffle机制
    - 分区-默认是HashPartition
    - 排序-默认是字典排序
    - 规约-默认没有,优化步骤combiner
    - 分组-相同的k2的v2放到集合中
  - reduce阶段
    - 通过用户自定义的reduceTask将k2v2转换成k3v3
    - 将结果k3,v3输出到文件系统中
- Mapper阶段
  - 分区数==reduce的任务数
  - 发牌shuffle的规则,Hash分区
  - 环形缓冲区 buffer将分玩区保存到数组(内存)

MapReduce并行度机制

- 所谓的并行度是指每个阶段有多少个task并行执行
- maptask并行度-- 逻辑切片 逻辑规划
  - split size =block size
  - 自适应能力 -- 根据文件大小 个数 block size自己决定个数
- reducetask并行度
  - 默认情况下是一个
    - 输出结果也是一个
  - 手动设置
    - 输出结果文件有多个
    - reduce阶段数据倾斜

MR数据分区 partition

- 当reducetask>1时,对应maptask思考输出结果传递给哪一个reducetask
- 默认分区规则
  - key.hashcode % NumReduceTask
  - 只要key一样 一定会到同一个分区,被同一个reducetask处理

mr压缩

```
1
2  1、压缩：
3      压缩是指以某种格式对文件进行归档操作
4      好处：
5          节省空间、减少I/O（磁盘I/O和网络I/O）、加快传输
6      缺点：
7          增加CPU负载，集群负载较高（会造成job堵塞）时不建议使用压缩
8
9      分类1：
10         Lossless无损压缩：日志数据
11         Lossy有损压缩：图片、视频
12     分类2：
13         对称压缩、非对称压缩
14
15 2、压缩格式对比(原始文件1403MB)
16     格式     压缩大小     压缩效率     压缩时间     解压时间     支持分割
17     snappy   701MB       50%左右       6秒         20秒        不支持
18     lz4      693MB       40%~50%       6秒         2秒         不支持
19     lzo      684MB       40%~50%       8秒         11秒        支持(建立索引)
20     gzip     447MB       30%~40%       86秒        23秒        不支持
```

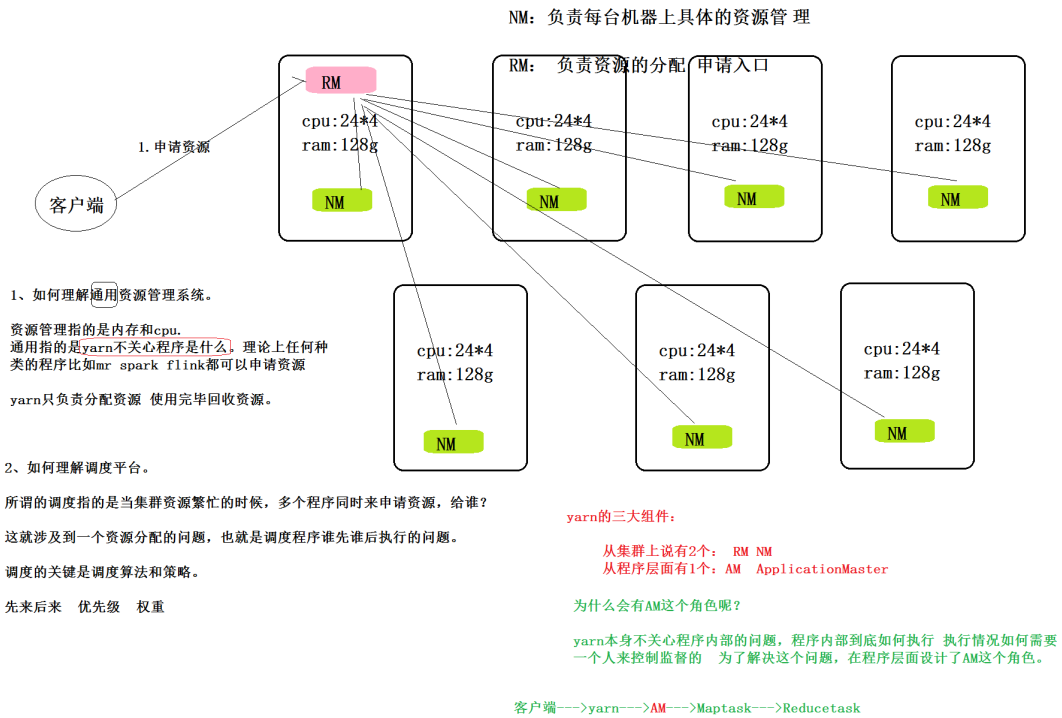
```
    bzip2    390MB       20%~30%       142秒      63秒         支持

#####lzo创建索引：com.hadoop.compression.lzo.LzoIndexer参数

问题：
    A、若Flume采集的数据要进行分布式计算，那Flume用什么压缩格式？
        Flume采集数据执行的是一个MapReduce程序，压缩格式只有支持分割，效率才会高，故可以采用BZip2和Lzo两种压缩格式

    B、冷数据是用什么压缩格式？
        冷数据用到的次数较少，我们更追求让其占用更小的空间，如果该数据是基于HDFS的，则可使用BZip2格式。



3、压缩的应用场景：
    输入：
        在input split前进行的压缩，压缩格式要支持分割(lzo或者bzip2)。
        如果不支持文件分割，则2G大小的解压缩文件会交由一个map任务处理，影响任务执行效率。
        如果支持分割，则该文件就会被分割成16个128mb的分片，每个分片有一个map任务处理。
        一般这儿的压缩要配合响应的数据存储格式使用。
    中间：
        map输出进行的压缩，压缩和解压缩要尽可能地快(snappy/lz4/lzo)。
    输出：
        redeuce输出前进行的压缩。
        如果reduce输出为最终输出，选用高压缩比的压缩格式(bzip2/gzip)。
        如果reduce输出非最终输出，选用支持分割的压缩格式(bzip2/lzo)。

#####执行hadoop checknative即可查看hadoop支持哪些压缩格式
#####但是要使用压缩，必须手动编译hadoop源码，并安装对应的压缩类才行。



4、hadoop压缩配置

vi core-site.xml

    <!--配置hadoop可能要使用到的压缩格式-->
    <property>
        <name>io.compression.codecs</name>
```

```
        <value>
            org.apache.hadoop.io.compress.SnappyCodec,
            org.apache.hadoop.io.compress.Lz4Codec,
            org.apache.hadoop.io.compress.LzoCodec,
            org.apache.hadoop.io.compress.GzipCodec,
            org.apache.hadoop.io.compress.BZip2Codec,
            org.apache.hadoop.io.compress.DefaultCodec    #zlip
        </value>
    </property>



vi mapred-site.xml

    <!--是否开启rmap输出端的压缩-->
    <property>
        <name>mapreduce.map.output.fileoutputformat.compress</name>
        <value>true</value>
    </property>
    <!--map输出采用lz4的压缩格式-->
    <property>
        <name>mapreduce.map.output.fileoutputformat.compress.codec</name>
        <value>org.apache.hadoop.io.compress.Lz4Codec</value>
    </property>
    <!--是否开启reduce输出端的压缩-->
    <property>
        <name>mapreduce.output.fileoutputformat.compress</name>
        <value>true</value>
    </property>
    <!--reduce输出采用bzip2的压缩格式-->
    <property>
        <name>mapreduce.output.fileoutputformat.compress.codec</name>
        <value>org.apache.hadoop.io.compress.BZip2Codec</value>
    </property>


5、hive压缩配置
    hive>set hive.exec.compress.output=true;
    hive>set mapreduce.output.fileoutputformat.compress.codec
```

```
99          >=org.apache.hadoop.io.compress.BZip2Codec;
100
101
102
```

MR的Uber模式

- Uber模式下，程序只申请一个AM Container：所有Map Task和Reduce Task，均在这个Container中顺序执行
- MR程序运行在YARN上时，有一些轻量级的作业要频繁的申请资源再运行，性能比较差怎么办？
  - Uber模式



- 默认不开启
- 配置：${HADOOP_HOME}/etc/hadoop/mapred-site.xml

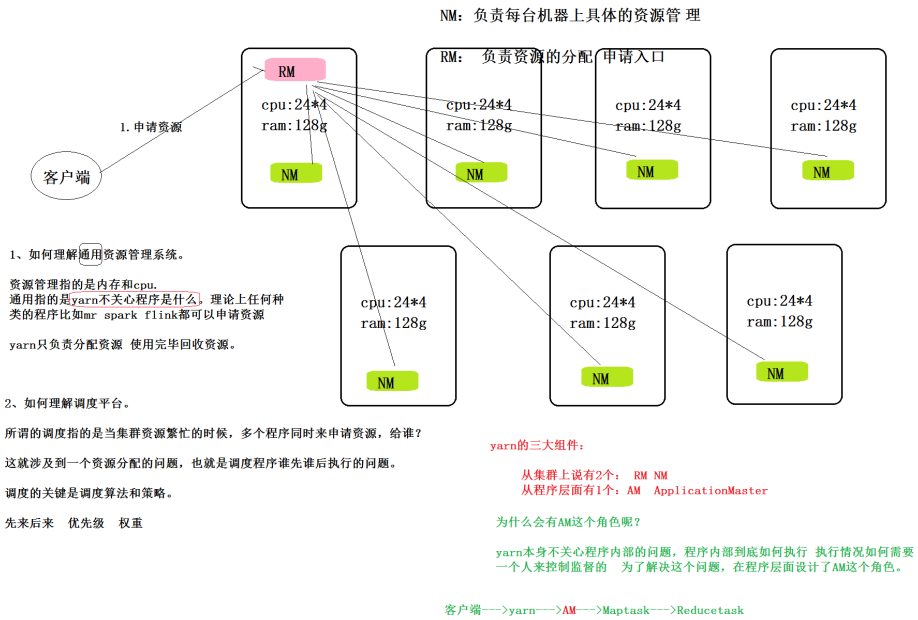
```
mapreduce.job.ubertask.enable=true
#必须满足以下条件
mapreduce.job.ubertask.maxmaps=9 --一个程序map数不超过9个
mapreduce.job.ubertask.maxreduces=1 --一个程序reduce数不超过1个
mapreduce.job.ubertask.maxbytes=128M --处理的数据量大小
yarn.app.mapreduce.am.resource.cpu-vcores=1 --APPMater的资源
yarn.app.mapreduce.am.resource.mb=1536M
```

特点

- Uber模式的进程为AM，所有资源的使用必须小于AM进程的资源
- Uber模式条件不满足，不执行Uber模式

- Uber模式，会禁用推测执行机制

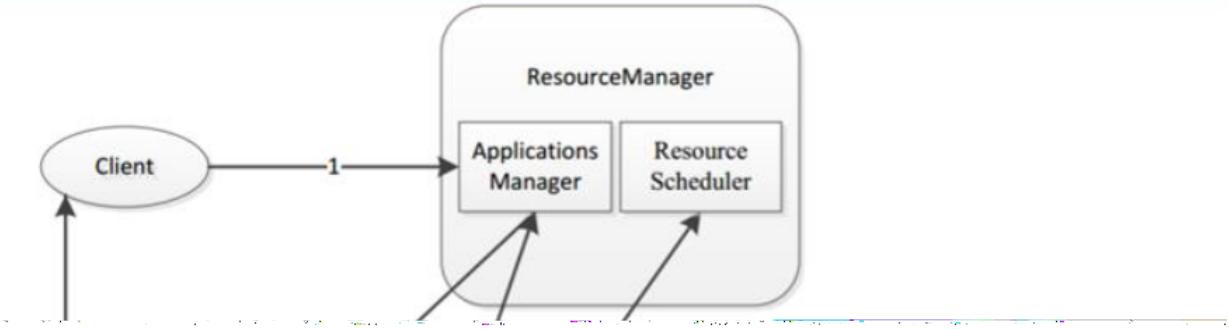
3.Apache Hadoop Yarn

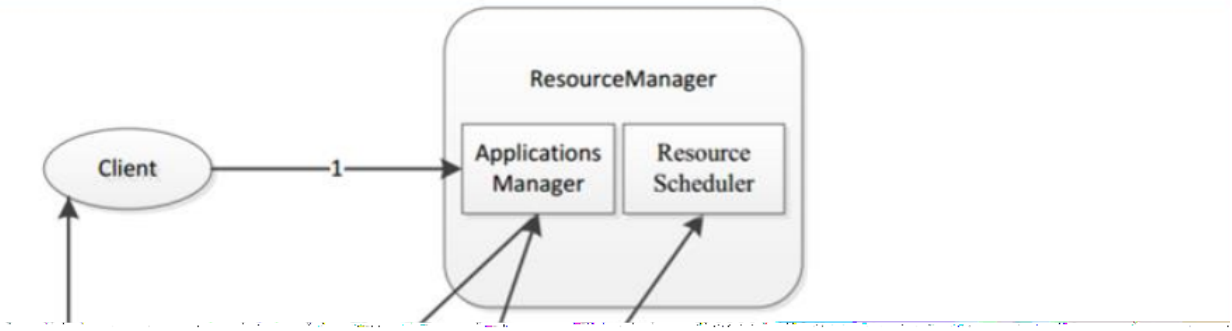


YARN的概述

- yarn是一个通用资源管理系统和调度平台
  - 资源指的跟程序运行相关的硬件资源 比如:CPU ram
- YARM组件 3大组件
  - 主角色 resourcemanager RM
    - 负责整个集群的资源管理和分配,是一个全局的资源管理系统
    - 是程序申请资源的唯一入口
  - 从角色 nodemanager NM
    - 负责每台机器上具体的资源管理,负责启动关闭container容器
- 程序内部-1个组件
  - ApplicationMaster AM
    - yarn作为通用资源管理系统,不关心程序的种类和程序内部执行情况
    - 谁来关心程序内部执行情况
      - 比如MapReduce程序来说,先maptask在运行reducetask
    - 需要一个组件来管理程序执行情况,程序内部的资源申请给阶段执行情况的监督
    - 为了解决这个问题yarn提供了第三个组件 applicationmaster
      - 主人,雇主;主宰;主人;有控制里的人;能手;擅长...者
    - 把applicationmaster称之为程序内部的老大角色,负责程序内部的执行情况
    - AM针对不同类型的程序有不同的具体实现
      - yarn默认实现了MapReduce的AM 名字叫做MrAppMaster

        - 其他软件比如spark flink 需要实现自己的AM ,才能在yarn运行
    - 结论:在上述设计模式下,人黑种类程序在yarn运行,首先都是申请资源运行AM角色,然后由AM控制程序内部具体的执行
- client提交程序到yarn运行流程
    - 以MapReduce程序为例





Yarn scheduler

- 所谓的调度器指的是当集群繁忙的时候,如何给申请资源的程序分配资源
- scheduler属于ResourceManager功能
- Yarn 3大调度策略
  - FIFO Scheduler 先进先出策略 先来先处理
  - capacity Scheduler 容量调度策略. 资源分两个位置,一大一小,大的处理紧急的任务,小的处理没有那么紧急的任务
  - Fair Sheduler 公平调度策略 来一个任务分平分资源
- Apache Hadoop版本默认策略是capacity.CDH商业版本默认策略是Fair
  - 默认情况下,整个yarn集群在capacity策略下,划分为一个队列,名字叫做default,占整个聚群资源的100
- 决定调度策略的参数

```
#yarn-site.xml

yarn.resourcemanager.scheduler.class=org.apache.hadoop.yarn.server.resourcemanager.scheduler.capacity.CapacityScheduler

#还可以在yarn 8088页面查看
```

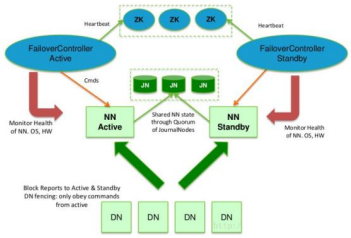
- prod生产环境 线上环境
- dev开发环境

4.Hadoop HA集群

问题1：如何避免脑裂问题的？

zkfc=zookeeper failover controller
zk容错控制器

A：时刻监督NN的健康状态
B：维持和zk集群的联系



a）两个zkfc同时去zk集群注册节点znode
（临时、非序列化）
谁抢注成功 谁对应的nn就是active。
用户也可以手动指定第一次谁是active.

b）没有注册需要注册监听 判断节点是否存在
对应的nn就是standby

c）Active NN故障-->Active zkfc--->断开和zk集群的连接
-->临时节点消失--->触发监听--->通知standby zkfc

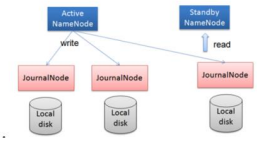
d）standby zkfc接到通知，不能立即把自己机器的nn切换成为active

standby 远程登录ssh到active那台机器上 补刀动作

"kill -9 active nn"

e)补刀结束（隔离机制），standby去注册节点 切换成为active

问题2：如何实现主备之间数据同步的？



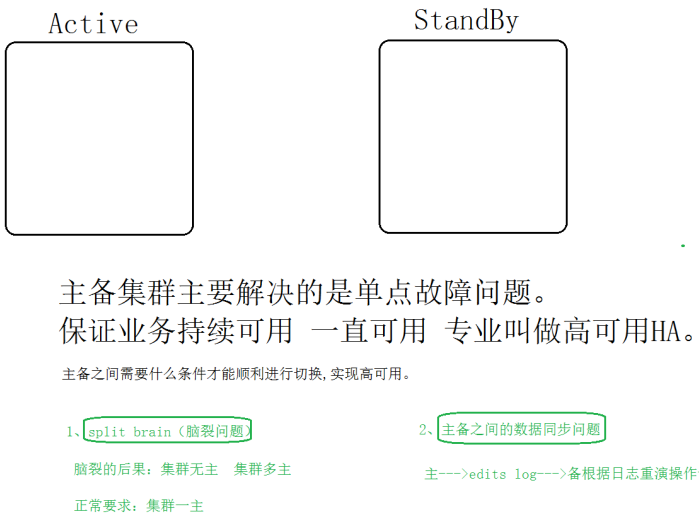
JN理解为 USB5.0高速存储盘。
其读写速度极快。

通常也是部署2N+1台

Active 把edits log写入到JN集群中。过半即为成功。

standby感知数据变化之后 读取editslog 根据日志重演操作记录 以此达到数据同步的目的。

- HA叫做高可用模式,逐一解决的是单点故障问题===SPOF.
  - SPOF无法避免 追求在出现SPOF可以容忍错误的发生(容错 failover)
- Hadoop中单点故障
  - NameNode
  - Resourcemanager
- NameNode HA QJM共享日志集群方案
  - zkfc实现主备切换避免脑裂--担心集群无主,集群多主
  - jn集群共享 edits log实现数据同步
- Resourcemanager HA 基于zk实现
  - RM需要维护的数据量很少,不像NN需要同步文件系统大量的元数据.直接基于zk即可完成
  - 别忘了 zk也是一个分布式小文件存储系统
- Hadoop HA集群搭建,难点就是配置文件的编写
- 听懂原理 可以不搭建,后面使用非HA集群
  - 针对当下集群进行逻辑删除 ,备份
  - 使用新的安装包进行编辑



- HA 主备切换错误,缺少依赖

```
2021-05-30 17:02:40,372 WARN org.apache.hadoop.ha.SshFenceByTcpPort: PATH=$PATH:/sbin:/usr/sbin fuser -v -k -n tcp 8020 via ssh: bash: fuser: command not found

#yum install -y fuser  猜想行为

yum install -y psmisc
```