## 1) MapReduce的基本介绍: 分布式计算框架

思想: 分而治之

比如说: 搬砖, 图书馆数书, 数学中计算从1~100和

MapReduce整个计算程序一般可以分为二个部分:

map: 负责 分 的 过程

reduce: 负责 合 的过程

MapReduce是一个分布式运算程序的编程框架，

核心功能是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，并发运行在Hadoop集群上。

MR既然是一个分布式计算的程序, 那么必然要有输入 和 输出

实现MapReduce的时候, 需要编写 map 和 reduce

map和reduce在进行数据传输的时候, 都是采用 kv键值对的方式来传输的

2) MapReduce的编程规范步骤 :

map阶段: 2步

1) 读取数据, 将读取过来的数据转换为 k1 和 v1

2) 自定义map逻辑, 将 k1 和 v1 转换为 k2 和 v2

shuffle阶段: 4步 接收 map阶段的k2和 v2 转换为 new k2和 v2

3) 分区 : 将相同的k2的数据发往同一个reduce

4) 排序 : 针对 k2 的数据 进行排序工作

5) 规约 : mr中优化步骤, 用于进行局部聚合功能, 一般是可以省略的

6) 分组 : 将相同k2的数据分成一组, 将v2的数据合并形成一个集合

reduce阶段: 2步

7) 自定义reduce逻辑, 接收k2 和 v2的数据, 将其转换为 k3 和 v3

8) 输出结果, 将k3 和 v3 输出到目的地

## 3) mapreduce的combiner:

combiner是MapReduce中优化步骤, 主要利用局部聚合的方式,来提升性能:

combiner主要是在map端实施局部聚合的工作, 将map的输出结果先进行一次局部合并, 然后发到reduce进行最终合并的操作

通过此操作, 可以减少 从map端 将数据发往reduce端的中间数据传输量, 从而提升MR的执行效率

请问 所有的业务 都可以利用这个局部聚合来优化嘛?

比如说: 求平均数

1,2,3,4,5,6,7 --> 正确: 4

比如现在有三个map, 每个map读取两个数据 正常逻辑, 每个map将数据都发送给reduce, 由reduce进行求平均数

如果想使用combiner, 在每一个map先求出一个平均数, 然后发送到reduce, 在统一求这个平均数

map1: 1,2 --> 1.5

map2: 3,4 --> 3.5

map3: 5,6,7 --> 6

reduce:

1.5,3.5,6 -->3...

并不是所有的业务都可以进行局部聚合的操作, 在实施局部聚合时候, 要校验是否会对最终结果有影响, 如果有, 那么不允许使用局部聚合

通过观察发现 大多数的combiner局部聚合过程, 与reduce执行一模一样的, 只不过, combiner是对每一个map进行聚合的操作

而reduce, 是对多个map的结果, 统一进行聚合的操作

如何实现一个combiner组件呢? 实现方式与 reduce是一模一样的

区别:

在驱动类中, 将combinner组件, 设置combinner位置:

job.setCombinerClass( combiner的组件)

## 4) MapReduce的并行度机制:

何为并行度: 在执行MR的时候, map运行的数量 以及 reduce运行数量 有谁来决定的问题

map运行的数量 取决于 读取文件的大小, 默认情况下, 是按照 128M 一个Map来处理, 同时每一个文件至少会有一个map来读取

思考: 如果一个文件300M 存储在HDFS中, 会被分成三个Block块(物理划分), 如果通过MR 读取这个文件, 请问, 是否是一个block对应一个map呢?

MR 对整个文件进行读取工作, 然后对整个文件进行逻辑划分(FileSplit)---文件切片

计算切片的规则:

Math.max(minSize, Math.min(maxSize, blockSize)); --默认大小 等于 blockSize的大小 而blockSize大小 等于 128M

minSize : 默认值为 0

maxSize : 默认值为 Long最大值

如果想要调整切片的大小: 可以通过 inputFormat 来进行调整:

TextInputFormat.setMinInputSplitSize(job,值); 调大切片的大小

TextInputFormat.setMaxInputSplitSize(job,值); 调小切片的大小

reduce运行数量: 一般有人为来设定的, 当业务需求要求结果是多个文件的时候, 这个时候一般就要设置reduce的数量

一个reduce就会输出一个聚合结果, 多个reduce就会输出多个聚合结果

如果只有一个reduce, 一般聚合结果, 最终聚合结果, 而如何使用多个reduce, 将聚合结果, 拆分多半

同时reduce数量 还取决于分区的数量, 如果进行自定义分区, 分区数量要和reduce是相等的

## 5) map阶段的工作机制:

1) 通过FileInputFormat 读取目标地址的数据, 通过FileInputFormat对目标数据进行FileSplit的逻辑分片数据, 最终会启动多少个mapTask取决于

FileSplit的逻辑分片的数量, 读取后, 将数据转换为K1和 v1

2) 自定义map逻辑, 接收k1和v1, 将k1和v1 转换为 k2和v2: 前序读取一行, 调用一次map逻辑, 输出k2和v2

3) 进行分区操作: map端输出一次k2和v2, 那么分区逻辑就会对k2进行分区操作, 默认采用HASH取模计算法, 计算后, 对k2进行分区标记

然后将标记好的k2和v2写入到 环形缓冲区

4) 随着不断写入到环形缓冲区, 环形缓冲区默认的大小为100M, 当达到缓冲区的80%的时候, MR会启动一个溢写线程, 将80%数据写出到磁盘中, 形成一个

临时文件, 在执行溢写过程中, 还会对k2数据进行排序操作, 如果设置了规约, 那么也就会在这个时候执行了

5) 不断的写入, 不断的溢写 , 产生多个临时文件, 当mapTask执行完成后, 会将最后的缓冲区的数据一并写出磁盘上, 然后开始进行合并操作. 保证一个

MapTask 只会产生一个最终的结果文件, 在合并的过程中, 同样也会进行排序, 并且有规约依然还会执行操作

6) 当mapTask的结果文件形成后, 静静等待reduceTask的拉取即可

shuffle中: 分区 排序 规约 属于 map端的shuffle过程

## 6) reduce阶段的工作流程:

1) 当map执行完成后, reduce开始执行拉取工作, 每一个reduce到mapTask执行的结果文件中, 拉取属于自己分区的数据

2) 将数据首先拉取到内存中, 当内存写满后, 将数据溢写到磁盘上, 形成多个临时文件, 在溢写的时候, 依然会对数据进行排序

3) 当整个拉取工作结束后, 对多个临时文件数据执行merge(合并)操作 , 将多个临时文件合并为一个结果文件, 同时合并过程依然排序

4) 对合并后的结果文件, 开始执行分组操作, 分好一组数据后, 就会调用一次reduce的逻辑, reduce接收k2和v2的数据, 将其转换为k3和v3

5) reduce每输出一次 k3和v3, 就会被FileOutPutFormat 接收到, 然后将数据追加到结果文件上

6) 当整个reducer将所有的分组数据都处理完成, 结果文件, 也就产生了...

shuffle中: 分组 属于 reduce的shuffle过程

发现不管是 map阶段 还是reduce阶段, 大量进行 磁盘 到 内存 内存到磁盘 ... 相关的IO操作, 主要目的能够解决处理海量数据计算问题

带来好处: 能够处理海量的数据

带来的弊端: 造成大量的磁盘IO工作 导致效率比较低

相关的配置:

配置 默认值 解释

mapreduce.task.io.sort.mb 100 设置环型缓冲区的内存值大小

mapreduce.map.sort.spill.percent 0.8 设置溢写的比例

mapreduce.cluster.local.dir ${hadoop.tmp.dir}/mapred/local 溢写数据目录

mapreduce.task.io.sort.factor 10 设置一次合并多少个溢写文件

## 7) 自定义分组:

如何自定义分组:

1) 创建一个类, 让其继承 WritableComparator

2) 重写空参构造方法 在方法调用父类中构建:

super(k2.class ,true)

3) 重写 compare方法: 方法中有两个参数 分别代表分区内相邻的两个数据

4) 在compare方法中, 实现自定义的比较是否相等的逻辑即可

5) 告知给驱动类