MapReduce

MapReduce 是一个分布式运算程序的编程框架.

MapReduce的优缺点

优点:

1. Mapreduce易于编程
2. 良好的扩展性
3. 高容错性(如果一个JOB失败,可以把这个JOB转移到另一个节点上去进行计算,不至于使得任务失败)
4. 适合超大文件的离线处理

缺点:

1. 不擅长实时计算,无法在秒级内返回结果
2. 不擅长流计算:MapReduce的输入数据是静态的,不能是动态变化的.
3. 不擅长DAG有向图的计算(例如多个应用程序存在依赖关系,后一个应用的输入作为前一个的输出,这个MapReduce是做不了的)
4. **每个MapReduce作业的输出结果都会写入磁盘,会导致存在大量I/O**

MapReduce 核心思想

1. MapReduce运算程序一般需要分为两个阶段: Map阶段和Reduce阶段
2. Map执行并发的MapTask,完全并行运行,互不相干
   Maptask的工作
   1. 读数据,按行处理,按空格切分行内单词
   2. 切割成KV(单词,1)建值对
   3. 将所有的KV对中的单词分区写到磁盘中(相当于Shuffle过程,用来做Map和Reduce的衔接)
3. Reduce阶段的并发ReduceTask, 完全互不相干,但是他们的数据依赖于上一个阶段的所有MapTask并发实例的输出,随后每一个Reduce Task 各自输出

4. MapReduce编程模型只能包含一个Map阶段和一个Reduce阶段,如果用户的业务逻辑非常复杂,那么多个MapReduce将会串行运行,但是由于每一个任务都会落盘,所以其I/O将会非常之高

Mapreduce的架构:

Mapreduce 运行时的三类实例进程:

1. MrAppMaster: 负责整个程序的过程代都和状态协调 (就是上图架构中的TaskTracker 和 JobTracker,)
2. MapTask: 整个Map阶段数据集的处理
3. ReduceTask: 负责整个Reduce阶段数据整合

#3 JobTracker 和 TaskTracker功能:

JobTarcker:

1. 负责集群资源的分配:监控TaskTracker状况、资源使用量等信息
2. 集群作业管理:将Job拆分成Task,跟踪job和Task的执行进度

TaskTracker:

1. 执行命令: 接收JobTracker 发送过来的命令并执行
2. 资源划分: 使用Slot 等量划分节点上的资源量(CPU,内存等), 一个Task 获取到了Slot之后才有机会运行
3. 汇报信息: 通过"heart beat"将本节点上的资源使用情况和任务进度汇报给JobTracker

#2 Mapreduce的工作流程:

1.整体流程

1 Client将用户编写的MapReduce作业的配置信息、jar包等信息上传到共享的文件系统，通常是HDFS。

2 Client提交作业给JobTracker，同时Job Tracker更糊Job Client 提交的文件从HDFS中获知文件的具体位置。

3 JobTracker读取作业的信息，生成一系列Map和Reduce任务，**调度给拥有空闲slot的TaskTracker**。

4 TaskTracker根据**JobTacker的指令启动Child进程执行Map任务**，Map任务将从HDFS读取输入数据。

5 JobTracker从TaskTracker处获得Map任务进度信息。

6 一旦Map任务完成后，JobTacker将Reduce任务分发给TaskTracker。

7 TaskTracker根据JobTacker的指令**启动Child进程执行Reduce任务**，Reduce任务将从Map端读取数据。

8 JobTracker从TaskTracker处获得Reduce任务进度信息。

9 当Reduce任务完成计算并将结果**分别存储在不同文件**中写入HDFS，则意味着整个作业执行完毕。

(shuffle过程实际上就是一个分区,告诉你要把这个K-V对分给哪一个ReduceTask)

2.数据输入的细节

1. 数据逻辑划分: InputFormat机制根据与定义格式将输入数据在逻辑上划分为若干个Split, **所以Jobtracker 分配给Map任务的数据的单位是Split 而并非 Block**
2. **Split的数量往往决定了Map任务的个数, 一个Split 一般由一个Map来解决**

3.Map and Shuffle

shuffle其实是Map过程中的一部分.

1. 数据在进行Map函数之后会进入一个缓冲区进行Partition操作,不同批次的数据会被分到不同的part中,在part中对这些分区进行排序,然后再进行归并排序.最后merge到一起, 这么做可以提高并发和效率.
2. 分区决定了会被Shuffle到哪一个Reduce任务来处理
3. 归并时间k-v中的value 拼接成一个新的list,比如<"a",1>,<"a",1>merge之后会得到<"a",<1,1>>
4. merge得到的文件会被压入本地磁盘,JobTracker会监视MapReduce的任务执行,并通知ReducerTask来领取数据
5. ReduceTask 并不是说一定要等待Map任务结束之后才可以进行,而是可以通过手动设置一个阈值,比如阈值为0.8,那么80%的maptask完成时就会进行一次落盘,,然后JobTracker就会侦查到这个过程,通知ReduceTask来领取数据(但是这里的ReduceTask拉取的必然是已经完成的Maptask的数据)
6. shuffle的流程概括(这里我发现有CSDN有一个写的特别好的解释,拿来引用, 信息源:https://blog.csdn.net/asn_forever/article/details/81233547)
   因为频繁的磁盘I/O操作会严重的降低效率，因此“中间结果”不会立马写入磁盘，而是优先存储到map节点的“环形内存缓冲区”，在写入的过程中进行分区（partition），也就是对于每个键值对来说，都增加了一个partition属性值，然后连同键值对一起序列化成字节数组写入到缓冲区（缓冲区采用的就是字节数组，默认大小为100M）。当写入的数据量达到预先设置的阈值后（mapreduce.map.io.sort.spill.percent,默认0.80，或者80%）便会**启动溢写出线程将缓冲区中的那部分数据溢出写（spill）到磁盘的临时文件中**，并在写入前根据key进行排序（sort）和合并（combine，可选操作）。溢出写过程按轮询方式将缓冲区中的内容写到mapreduce.cluster.local.dir属性指定的目录中。**当整个map任务完成溢出写后**，会对磁盘中这个map任务产生的所有临时文件（spill文件）进行归并（merge）操作生成最终的正式输出文件，此时的归并是将所有spill文件中的相同partition合并到一起，并对各个partition中的数据再进行一次排序（sort），生成key和对应的value-list，文件归并时，如果溢写文件数量超过参数min.num.spills.for.combine的值（默认为3）时，可以再次进行合并。至此，map端shuffle过程结束，接下来等待reduce task来拉取数据。对于reduce端的shuffle过程来说，reduce task在执行之前的工作就是不断地拉取当前job里每个map task的最终结果，然后对从不同地方拉取过来的数据不断地做merge最后合并成一个分区相同的大文件，然后对这个文件中的键值对按照key进行sort排序，排好序之后紧接着进行分组，分组完成后才将整个文件交给reduce task处理。

4.Reduce

Reducetask总体可以分为4个阶段:

1.copy阶段 2. merge阶段 3.sort阶段 4.Reduce阶段

1. copy 阶段从Map task所在的机器上的磁盘上获取数据
2. merge 和sort阶段把多个从不同机器上获得的文件归并写入磁盘(如果内存足够用的话就放在内存),形成一个大的文件(文件中的键是排过序的)
3. 执行Reduce操作

Reduce的任务个数取决于集群中可用的Reduce任务槽数目,通常设置比Reduce任务槽数目稍微小一点任务数量,预留的资源可以用来容错

5.数据输出的细节

每一个ReduceTask都会输出一个文件到指定目录

输出的格式需要提前被定义

#2 Mapreduce的的容错机制

1. Jobtracker故障:
   整个作业重新启动
2. TaskTracker故障:
   表现为Jobtracker无法收到来自TaskTracker的心跳,此时Jobtracker会将这个TaskTracker的作业重新分配给一个新的节点执行,**这个过程对于用户来说是透明的**,用户只会感觉执行时间变长
3. Task故障:
    1. MapTask故障:
       重新执行MapTask
    2. ReduceTask故障:
       重新执行ReduceTask,从MapTask本地磁盘读取数据