**学习日志**

姓名：李奥 学号： 1812998 日期： 6.25

**学习内容安排：**

**MapReduce工作原理，MapReduce的类型与格式**

**学习反馈：**

MapReduce是一种编程模型，用于大规模数据集（大于1TB）的并行运算。它们的主要思想，都是从函数式编程语言里借来的。每次一个步骤方法会产生一个状态，这个状态会直接当参数传进下一步中。而不是使用全局变量。

MapReduce框架

MapReduce将复杂的，运行大规模集群上的并行计算过程高度地抽象两个函数：Map和Reduce

MapReduce采用“分而治之”策略，将一个分布式文件系统中的大规模数据集，分成许多独立的分片。这些分片可以被多个Map任务并行处理。

MapReduce设计的一个理念就是“计算向数据靠拢”，而不是“数据向计算靠拢”，原因是，移动数据需要大量的网络传输开销

MapReduce框架采用了Master/Slave架构，包括一个Master和若干个Slave，Master上运行JobTracker，Slave运行TaskTracker

Hadoop框架是用JAVA来写的，但是,MapReduce应用程序则不一定要用Java来写。

JobTracker：初始化作业，分配作业，TaskTracker与其进行通信，协调监控整个作业

TaskTracker：定期与JobTracker通信，执行Map和Reduce任务

HDFS：保存作业的数据、配置、jar包、结果

作业调度算法：

FIFO调度器（默认）、公平调度器、容量调度器

TaskTracker和JobTracker之间的通信与任务的分配是通过心跳机制完成的；

TaskTracker会主动向JobTracker询问是否有作业要做，如果自己可以做，那么就会申请到作业任务，这个任务 可以使Map也可能是Reduce任务；

TaskTraker将代码和配置信息到本地；

分别为每一个Task启动JVM运行任务

任务在运行过程中，首先会将自己的状态汇报给TaskTracker，然后由TaskTracker汇总告之JobTracker；

任务进度是通过计数器来实现的；

JobTracker是在接受到最后一个任务运行完成后，才会将作业标志为成功。

MapReduce编程模型

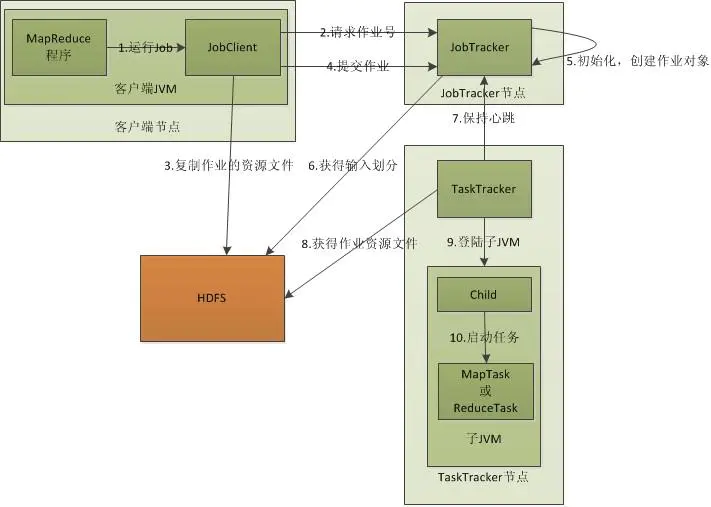
MapReduce 由 两 个 阶 段 组 成 ：Map 和 Reduce。

map() 函数以 key/value 对作为输入，产生另外一系列 key/value 对作为中间输出写入本地 磁盘。MapReduce 框架会自动将这些中间数据按照 key 值进行聚集，且 key 值相同（用户可 设定聚集策略，默认情况下是对 key 值进行哈希取模）的数据被统一交给 reduce() 函数处理。

reduce() 函数以 key 及对应的 value 列表作为输入，经合并 key 相同的 value 值后，产 生另外一系列 key/value 对作为最终输出写入 HDFS。

指定三个组件分别是 InputFormat、Partitioner 和 OutputFormat， 它们均需要用户根据自己的应用需求配置①指定输入 文件格式。将输入数据切分成若干个 split，且将每个 split 中的数据解析成一个个 map() 函数 要求的 key/value 对。②确定 map() 函数产生的每个 key/value 对发给哪个 Reduce Task 函数处 理。③指定输出文件格式，即每个 key/value 对以何种形式保存到输出文件中。

MapReduce作业运行流程



1.在客户端启动一个作业。

2.向JobTracker请求一个Job ID。

3.将运行作业所需要的资源文件复制到HDFS上，包括MapReduce程序打包的JAR文件、配置文件和客户端计算所得的输入划分信息。这些文件都存放在JobTracker专门为该作业创建的文件夹中。文件夹名为该作业的Job ID。JAR文件默认会有10个副本（mapred.submit.replication属性控制）；输入划分信息告诉了JobTracker应该为这个作业启动多少个map任务等信息。

4.JobTracker接收到作业后，将其放在一个作业队列里，等待作业调度器对其进行调度（这里是不是很像微机中的进程调度呢，呵呵），当作业调度器根据自己的调度算法调度到该作业时，会根据输入划分信息为每个划分创建一个map任务，并将map任务分配给TaskTracker执行。对于map和reduce任务，TaskTracker根据主机核的数量和内存的大小有固定数量的map槽和reduce槽。这里需要强调的是：map任务不是随随便便地分配给某个TaskTracker的，这里有个概念叫：数据本地化（Data-Local）。意思是：将map任务分配给含有该map处理的数据块的TaskTracker上，同时将程序JAR包复制到该TaskTracker上来运行，这叫“运算移动，数据不移动”。而分配reduce任务时并不考虑数据本地化。

5.TaskTracker每隔一段时间会给JobTracker发送一个心跳，告诉JobTracker它依然在运行，同时心跳中还携带着很多的信息，比如当前map任务完成的进度等信息。当JobTracker收到作业的最后一个任务完成信息时，便把该作业设置成“成功”。当JobClient查询状态时，它将得知任务已完成，便显示一条消息给用户。

以上是在客户端、JobTracker、TaskTracker的层次来分析MapReduce的工作原理的

Map、Reduce任务中Shuffle和排序的过程

**输入分片（input split）：**

在进行map计算之前，mapreduce会根据输入文件计算输入分片（input split），每个输入分片（input split）针对一个map任务，输入分片（input split）存储的并非数据本身，而是一个分片长度和一个记录数据的位置的数组，输入分片（input split）往往和hdfs的block（块）关系很密切，假如我们设定hdfs的块的大小是64mb，如果我们输入有三个文件，大小分别是3mb、65mb和127mb，那么mapreduce会把3mb文件分为一个输入分片（input split），65mb则是两个输入分片（input split）而127mb也是两个输入分片（input split），换句话说我们如果在map计算前做输入分片调整，例如合并小文件，那么就会有5个map任务将执行，而且每个map执行的数据大小不均，这个也是mapreduce优化计算的一个关键点。

**map阶段：**

就是编写好的map函数了，因此map函数效率相对好控制，而且一般map操作都是本地化操作也就是在数据存储节点上进行；

**combiner阶段：**

combiner阶段是可以选择的，combiner其实也是一种reduce操作，因此我们看见WordCount类里是用reduce进行加载的。Combiner是一个本地化的reduce操作，它是map运算的后续操作，主要是在map计算出中间文件前做一个简单的合并重复key值的操作，例如我们对文件里的单词频率做统计，map计算时候如果碰到一个hadoop的单词就会记录为1，但是这篇文章里hadoop可能会出现n多次，那么map输出文件冗余就会很多，因此在reduce计算前对相同的key做一个合并操作，那么文件会变小，这样就提高了宽带的传输效率，毕竟hadoop计算力宽带资源往往是计算的瓶颈也是最为宝贵的资源，但是combiner操作是有风险的，使用它的原则是combiner的输入不会影响到reduce计算的最终输入，例如：如果计算只是求总数，最大值，最小值可以使用combiner，但是做平均值计算使用combiner的话，最终的reduce计算结果就会出错。

**shuffle阶段：**

将map的输出作为reduce的输入的过程就是shuffle了，这个是mapreduce优化的重点地方。

reduce阶段：

和map函数一样也是自己编写的，最终结果是存储在hdfs上的。

输入 --> map --> shuffle --> reduce -->输出

HDFS block和MapReduce split之间的联系？

Block：HDFS中最小的数据存储单位，默认是164M；Split：MapReduce中最小的计算单元，默认与Block一一对应。

两者的对应关系是任意的，可有用户控制。

Partitioner:数据分组 决定了Map task输出的每条数据交给哪个Reduce Task处理。默认实现：hash(key) mod R R是Reduce Task数目，允许用户自定义，很多情况下需要自定义Partitioner ，比如“hash(hostname(URL)) mod R”确保相同域名的网页交给同一个Reduce Task处理 属于（map）阶段。

Combiner：可以看做local reduce 合并相同的key对应的value，通常与reducer逻辑一样 ，好处是减少map task输出 数量（磁盘IO），见少Reduce-map网络传输数据量(I网络IO) 结果叠加属于（map）阶段。

Shuffle：Shuffle描述着数据从map task输出到reduce task输入的这段过程 (完整地从map task端拉取数据到reduce 端。

在跨节点拉取数据时，尽可能地减少对带宽的不必要消耗。减少磁盘IO对task执行的影响。

) 属于(reduce)阶段。

1. 默认的mapper是Ｍapper类，它将输入的键和值原封不动地写到输出中
2. 默认的partitioner是HashPartitioner，它对每条记录的键进行哈希操作以决定该记录应该属于哪个分区（每个分区对应于一个reduce任务）
3. 默认的reducer是Reducer类，它将所有的输入写到输出中
4. map任务的数量等于输入文件被划分成的块数
5. reduce任务的个数的选择： 一个经验法则是目标reducer保持在每个运行5分钟左右且产生至少一个HDFS块的输出比较合适
6. 默认的输入格式是TexInputFormat，输出是TextOutpFormat

### **MapReduce的类型与格式输入分片与记录**

* 一个输入分片就是由单个map操作来处理的数据块，并且每一个map只处理一个分片、
* 每个输入分片分为若干个记录，每条记录就是 一个键值对，map将一个接一个地处理记录
* 输入分片和记录都是逻辑概念，不一定对应着文件，也可能对应其他数据形式，如对于数据库，输入分片就是对应于一个表上的若干行，一条记录对应着其中的一行

输入分片只是指向数据的引用，不包含数据本身

1、InputSpilt接口（Java中的实现），包含以字节为单位的长度，表示分片的大小，用以排序分片，以便优先处理最大的分片，从而最小化作业运行时间一组存储位置，供MR系统使用一边将map任务尽可能放在分片数据附近该接口由InputFormat创建

2、InputFormat运行作业的客户端使用getSplits方法计算分片，并将结果告知application master，后者使用其存储信息来调度map任务从而在集群集群上处理这些分片数据

3、map任务将输入分片传给createRecordReader方法来获取这个分片的RecordReader（就像是记录上的迭代器），map任务用这个RecordReader来生成记录的键值对，然后再将键值对传递给map函数（参见run方法）

/////InputFormat接口public abstract class InputFormat<K, V> {

public abstract List<InputSplit> getSplits(JobContext context) throws IOException, InterruptedException;

public abstract RecordReader<K, V> createRecordReader(InputSplit split, TaskAttemptContext context) throws IOException, InterruptedException;}

/////Mapper的run方法public void run(Context context) throws IOException, InterruptedException {

setup(context);

while (context.nextKeyValue()/\*委托给RecorReader的同名方法，为mapper产生键值对\*/) {

map(context.getCurrentKey(), context.getCurrentValue(), context);//从RecordReader中检索出并传递给map方法

}

cleanup(context);}