第七章 MapReduce

第七章考点

- 传统并行计算框架与MapReduce的比较
- MapReduce模型、设计理念，Map函数，Reduce函数
- MapReduce工作流程
- 执行阶段
- WordCount运行实例

一、MapReduce概述

1. 分布式并行编程
   MapReduce是一种分布式并行编程框架
   基于大数据的摩尔定律，需要处理的数据量快速增加，人们开始借助于分布式并行编程来提高程序性能
   分布式程序运行在大规模计算机集群上，可以并行执行大规模数据处理任务，从而获得海量的计算能力
2. MapReduce相较于传统的并行计算框架有什么优势

	集群的架构和容错性	**硬件价格及拓展性**	**编程和学习难度**	**适用场景**
传统的并行编程框架	性能计算模型HPC，通常采用共享式框架（共享内存，共享底层的存储空间）底层都是采用统一的存储区域网络SAN（共享底层的逻辑） 扩展比较难其中一个硬件发生故障以后容易导致整个集群不可工作	使用刀片服务器，共享网络以及存储区域网络SAN，共享式集群架构的扩展性比较差	编程难度高，编程原理和多线程遍历的逻辑比较类似，需要借助很多的互斥信号量锁等机制，要实现不同人物之间的同步	用于实时的细粒度计算，尤其适用于计算密集型的应用
MapReduce	采用典型的非共享式架构，在整个集群当中每个节点都拥有自己的内存任何一个节点出现问题不会影响到其他节点的正常运行，整个集群中又设计了冗余和容错机制	整个集群可以随意增加减少相关的计算节点，不需要很高端的机器，只需要很廉价的PC机即可，因此PC机要比刀片服务器廉价得多，所以很便宜，而且扩展性好	自动帮你实现分布式部署到集群上面各个机器上面去运行	一般适用于非实时的批处理以及数据密集型应用

3. MapReduce模型简介
   MapReduce将复杂的，运行于大规模集群上的并行计算过程高度地抽象到了两个函数，Map和Reduce
   编程容易，不需要掌握分布式并行编程细节，也可以很容易把自己的程序运行在分布式系统上，完成海量的数据运算

- **设计理念**
  MapReduce采用分而治之的策略，一个存储在分布式文件系统中的大规模数据集会被切分为许多独立的分片，然后为每一个分片单独地启动一个Map任务，最终通过多个Map任务，并行地在多个机器上去处理
  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，因为移动数据需要大量的网络传输开销
- 实现方式
  要完成一次数据分析时，选择一个计算节点，把运行数据分析的程序放在计算节点上运行，然后把它所涉及的数据，全部从各个不同节点上面拉过来，传输到计算发生的地方
  计算向数据靠拢，MapReduce框架会将Map程序就近地在HDFS数据所在的节点运行，即将计算节点和存储节点放在一起运行，从而减少节点间的数据移动开销
  因为移动数据需要大量的网络传输开销，尤其是在大规模数据环境下，这种开销尤为惊人，所以移动计算要比移动数据要更加经济
- MapReduce模型主从框架
  MapReduce框架采用了Master/Slave框架，包括一个Master和若干Slave，Master上运行的JobTracker，Slave上运行TaskTracker
  Hadoop框架是用java实现的，但是MapReduce应用程序则不一定要用Java来写

3. Map和Reduce函数

函数	输入	输出	说明
Map	(key,value)如<行号，“a,b,c”>	如： <"a",1> <"b",1> <"c",1>	将小数据集进一步解析为一批<key,value>对，输入Map函数中进行处理

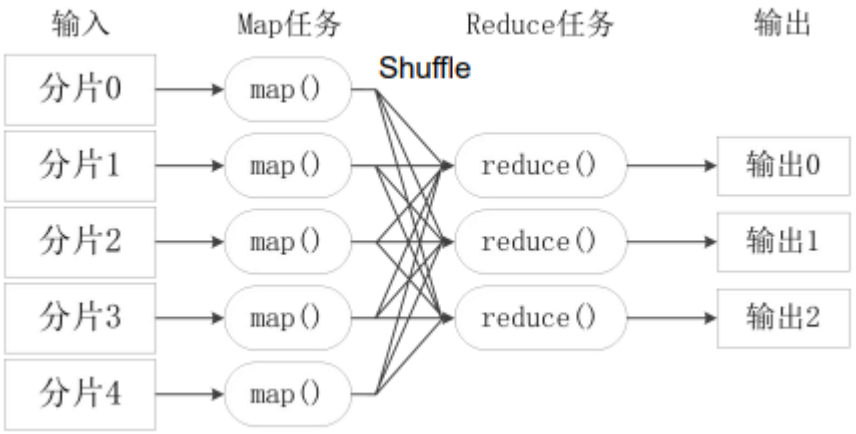
函数	输入	输出	说明
			每一个输入的 会输出一批 ，其中 是计算的中间结果
Reduce	如<"a",3>	输出 (<key,value>) 如<"a",3>	输入的中间结果 中的表示的是一批属于同一个的value

4. MapReduce的体系架构
   MapReduce的体系架构主要由四个部分组成Client，JobTracker，TaskTracker以及Task
   - Client
     用户通过编写MapReduce程序通过Client提交到JobTracker端
     用户可以通过Client提供的一些接口查看作业的运行状态
   - JobTracker
     JobTracker负责资源监控和作业调度
     JobTracker监听所有TaskTracker和Job的健康状况，一旦发现失败，就将其相应的任务转移到其他节点
     JobTracker会跟踪任务的执行进度，资源使用等信息，并将这些信息告诉任务调度器（TaskScheduler），而调度器会在资源出现空闲的时候选择合适的任务去使用这些资源
   - TaskTracker
     TaskTracker会周期性地通过“心跳”将本节点上资源使用情况和任务的运行进度汇报给JobTracker，同时接收JobTracker发送过来的命令，并执行相应的操作（如启动新任务，杀死任务等）
     TaskTracker使用“slot”等量划分本节点上的资源量，一个Task获取到一个Slot之后才有机会运行，而Hadoop调度器的作用就是将每个TaskTracker上的空闲slot分配给Task使用，slot分为Map slot和Reduce slot分别为Map Task和Reduce Task使用
   - Task
     Task分为Map Task和Reduce Task两种，均由TaskTracker启动，同一台机器上可以同时运行两种Task

---

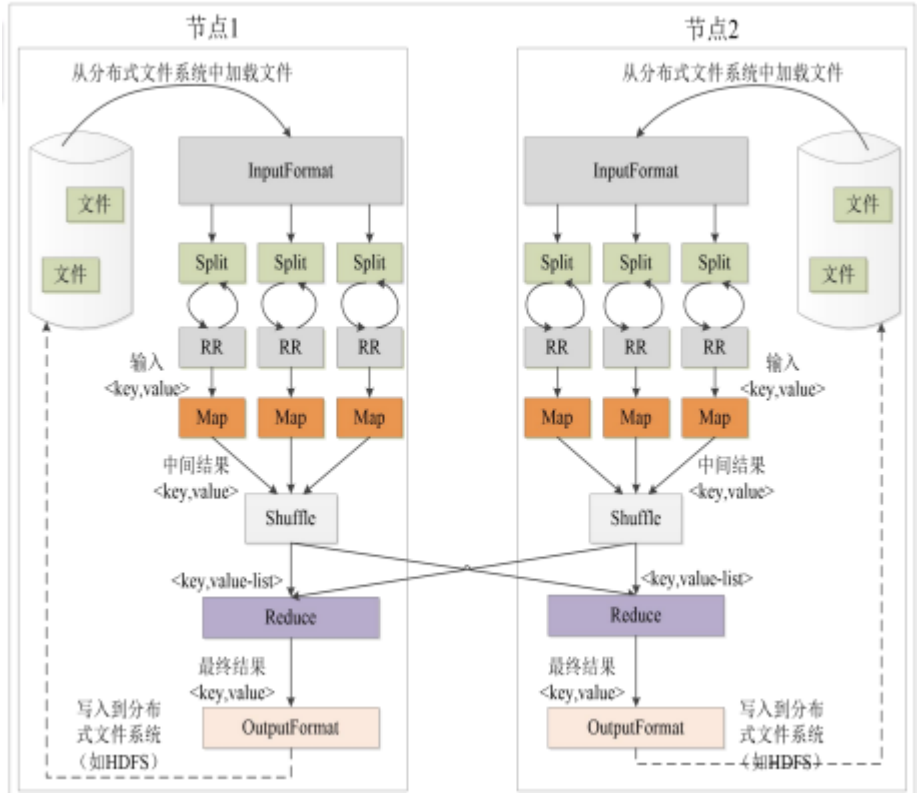
三、MapReduce的工作流程

1. **工作流程**



- 一个大的MapReduce作业会首先被拆成许多个Map任务在多台机器上并行运行，每个Map任务通常运行在数据存储的节点上
- 当Map任务结束后，会生成以 `<key,value>` 形式存储的许多中间结果，这些中间结果会被分发到多个Reduce任务在多台机器上并行执行，具有相同的key的<key,value>会被发送到同一个Reduce任务，Reduce任务会对中间的结果进行汇总计算得到最后结果，并输出到分布式文件系统
  在这个过程中：
- 不同的Map任务之间不会通信
- 不同的Reduce任务之间也不会发生任何信息交换
- 用户不能显式从一台机器到另外一台机器发送消息
- 所有的数据交换都是通过MapReduce框架自身去实现的

2. MapReduce的多个**执行阶段**



- 加载文件，InputFormat验证输入的格式是否符合定义，将输入的文件切分为逻辑上的多个InputSplit(它是MapReduce对文件进行处理和运算的输入单位，实际上并没有对文件进行实际切分，只记录了处理数据的位置和长度)

- 由于InputSplit是逻辑切分而非物理切分，还需要通过RecordReader(RR)来处理InputSplit中具体的记录，加载数据并将其转化为适合Map任务读取的键值对，输入给Map任务
- Map任务会根据用户自定义的映射规则，输出一系列<key,value>作为中间结果
- 为了让Reduce可以并行处理Map的结果，需要对Map的输出进行一定的分区，排序，合并，归并等操作，得到<key,value_list>形式的中间结果，再交给对应的Reduce进行处理，这个过程称为shuffle
- Reduce以一系列<key,value_list>中间结果作为输出，执行用户定义的逻辑，输出结果为OutputFormat模块
- OutputFormat模块会验证输出目录是否存在，以及输出结果类型是否已经符合配置文件里面的配置类型，如果都满足就输出Reduce的结果到分布式文件系统

3. 相关描述
   - 关于Split：
     HDFS 以固定大小的block（块）为基本单位存储数据，而对于MapReduce 而言，其处理单位是 split。split 是一个逻辑概念，它只包含一些元数据信息，比如数据起始位置、数据长度、数据所在 节点等。它的划分方法完全由用户自己决定
   - Map任务的数量
     Hadoop为每个split创建一个Map任务，split的多少决定了Map任务的数目，大多数情况下，理想的分片大小是一个HDFS块
     分片大小与块大小不同情况下比较，分别从Map任务数量以及数据传输开销两个方面比较：
     - 大分片（Split > Block）—— 弊端：费网（慢）
     - **情况**：一个任务要处理的数据分散在不同的机器上。
     - **后果**：为了凑齐数据，必须跨机器通过**网络传输**拉取数据。
     - **总结：破坏了“数据本地化”，导致大量网络拥堵，处理速度变慢。**
     - 小分片（Split < Block）—— 弊端：费事（乱）
     - **情况**：把一大块数据切得太碎，导致任务数量暴增（比如 1000 个小任务）。
     - **后果**：系统忙于**启动和关闭**这些小任务（这一过程很耗时），真正干活的时间反而少了，且调度压力极大。
     - **总结：管理开销太大，“启动任务的时间”超过了“实际计算的时间”，浪费资源。**
     - 默认（Split = Block）—— 优势：完美平衡（快）
     - **情况**：一个任务正好处理本地磁盘上的一个完整数据块。
     - **后果**：既不需要跨网络拉数据（0 网络开销），任务大小也适中（不会因为太碎而浪费启动时间）。
     - **总结：实现了“计算向数据靠拢”，效率最高。**
   - Reduce任务的数量
     最优的Reduce任务个数取决于集群中可用的reduce任务槽(slot)的数目。
     通常设置比reduce任务槽数目稍微小一些的Reduce任务个数（这样可以预留一些系统资源处理 可能发生的错误）。

4. shuffle过程详解

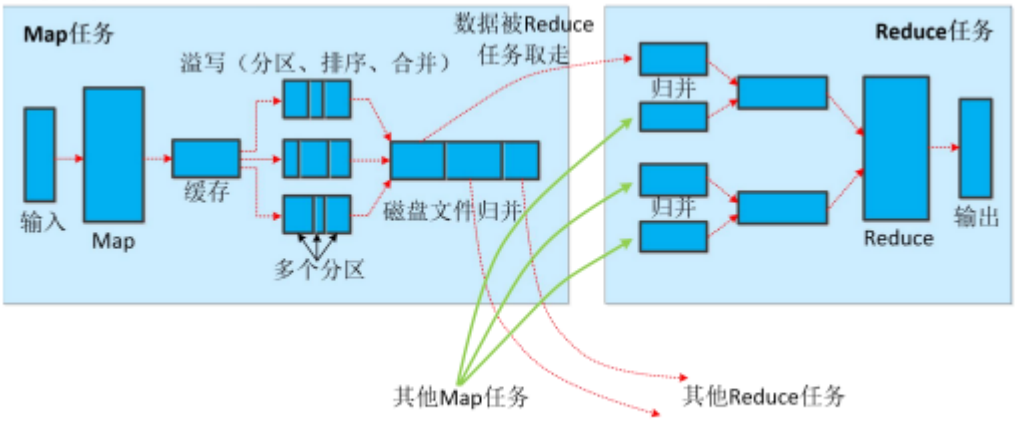
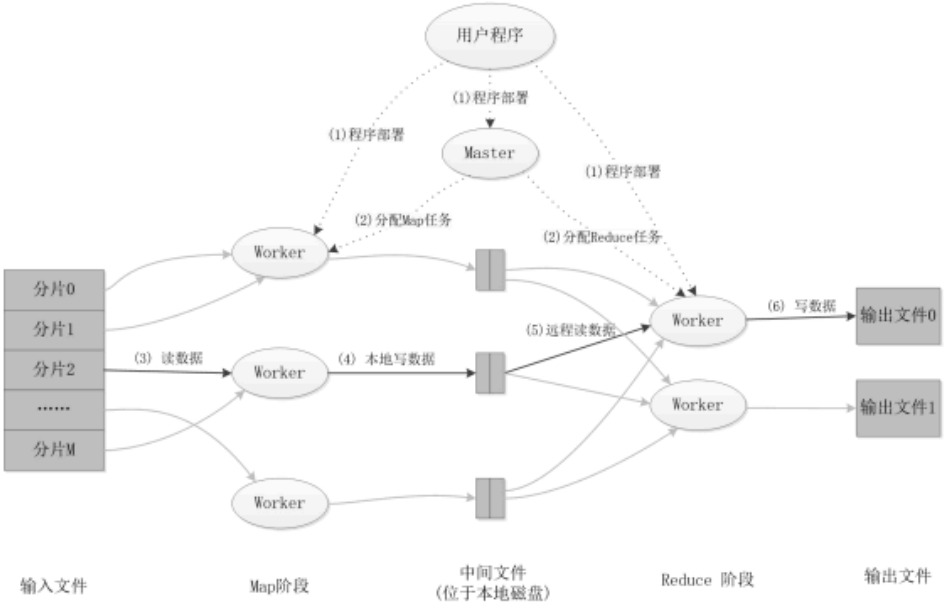


图7-3 Shuffle过程

- shuffle过程的输入是从分布式文件系统输入数据，输出也是到分布式文件系统
- Map端的shuffle过程：
  - 输入数据和执行Map任务：Map任务输入数据一般保存在分布式文件系统（HDFS）中，这些文件的格式是任意的，也可以是二进制格式，Map任务接收<key,value>作为输入后，按一定的映射会泽将其转化为多个<key,value>输出
  - 写入缓存：每个Map任务都会被分配一个缓存，Map任务输出结果不是立即写入磁盘，而是首先写入缓存。 在缓存中积累一定数量的Map任务输出结果后， 再一次性批量写入磁盘，这样可以大大减少对磁盘I/O的影响。
  - 溢写（分区、排序和合并） 提供给MapReduce的缓存的容量是有限的，默认 大小100MB。随着Map任务的执行，缓存中Map任 务结果的数量会不断增加，很快占满整个缓存。这 时，就必须启动溢写操作，把缓存中的内容一次性 写入磁盘，并清空缓存。一般来说会设置一个溢写比例，当缓存中写入数据占比超过溢写比例时候就启动溢写程序
  - 文件归并：每次溢写操作都会在磁盘中生成出一个新的溢写文件，随着MapReduce任务的进行，磁盘中的溢写文件数量会越来越多，最终在Map任务全部结束之前，系统会对所有溢写文件中的数据进行归并，生成一个大的溢写文件，归并是指具有相同的key的键值对归并成一个新的键值对，得到一个<k1,<v1,v2,...,vn>>
  - 经过上述步骤后，Map端的Shuffle过程全部完成，最终生成一个会被存放在本地磁盘上的大文件，这个大文件的数据是被分区的，不同分区会被发送到不同的Reduce任务进行并行处理。在此过程中JobTracker会一直检测Map任务的执行，当Map任务完成的时候，会立即通知相关的Reduce任务来领取对应的数据，并且开始Reduce端的Shuffle过程
- Reduce端的shuffle过程：
  - 领取数据：Reduce任务通过RPC向JobTracker询问Map任务是否已经完 成，若完成，则领取数据存放到自己所在机器的本次磁盘上。因此，在每个 Reduce任务真正开始之前，它大部分时间都在Map端“领取”属于自己处理的 那些分区的数据
  - 归并数据：从Map端领取的数据会被存放在Reduce任务所在机器的缓存中，如果缓存被占满， 就会像Map端一样被溢写到磁盘中。Reduce任务会从多个Map机器领取数据，因此缓存中的数据是 来自不同的Map机器，一般会存在很多可以合

并的键值对。当溢写过程启动时，具有相同key的键值 对会被归并，如果用户定义了Combiner，则归并后的数据还可以执行合并操作，减少写入磁盘的数 据量。每个溢写结束后，会生成一个溢写文件，类似地，文件会进行归并，生成若干个大文件

- 把数据输入给Reduce任务：归并后得到的若干个大文件直接输入Reduce任务，这样可以 减少磁盘读写开销。 由此整个Shuffle过程顺利结束，接下来，Reduce任务会执行Reduce函数中定义的各种映射， 输出最终结果，并将其保存到分布式文件系统中



四、WordCount案例分析

该部分要求基于WordCount案例实现对MapReduce的完整操作，要求掌握每一步骤数据变化的书写

注意：MapReduce在进行排序的时候是按照字典序进行排序，所谓的字典序排序是按照阿斯克码（ASCII）进行排序，ASCII的排序规则是大写字母在小写字母前面，例如比较小写a和大写Z，大写Z在小写a的签名，此外MapReduce会把数字当做字符来看，对1,2,10，进行排序，排序结果为1,10,2，如果使用的是内置的数字类型如（IntWritable, LongWritable），那么就直接按照数值大小排序

五、MapReduce的具体应用

MapReduce可以很好地应用于各种计算问题

- 关系代数运算（选择、投影、并、交、差、连接）
- 分组与聚合运算
- 矩阵-向量乘法
- 矩阵乘法

1. 用MapReduce来实现关系的自然连接
   假设有关系R(a，b)和S(b，c)，对二者进行自然连接操作。
   - 使用Map过程，把来自R的每个元组<a,b>转换成一个键值对<b,<R,a>>，其中的键就是属性b的值。把关系 R包含到值中，这样做使得我们可以在Reduce阶段，只把那些来自R的元组和来自S的元组进行匹配。类似地，使用Map过程，把来自S的每个元组<b,c>，转换成一个键值对<b,<S,c>>。
   - 所有具有相同b值的元组被发送到同一个Reduce进程中，Reduce进程的任务是，把来自关系R和S的、具有相同属性b值的元组进行合并。
   - Reduce进程的输出则是连接后的元组，输出被写到一个单独的输出文件中

六、MapReduce的编程实践

此部分不要求掌握代码，但是需要看懂代码，此外请区分MapReduce过程中定义combiner和没定义combiner的区别

特性	无 Combiner	有 Combiner
网络传输量	大（原始数据）	小（局部聚合数据）
磁盘 I/O	高	低
Reducer 压力	大	小
计算速度	较慢	通常较快
适用性	所有逻辑	仅限满足结合律/交换律的逻辑 (Sum, Max, Min)