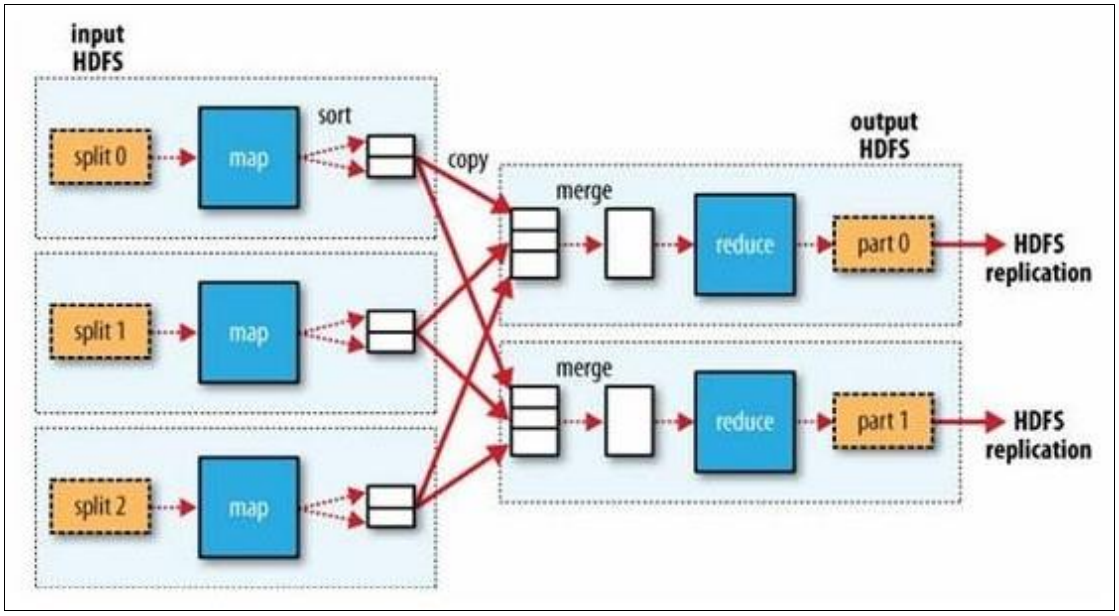
目录

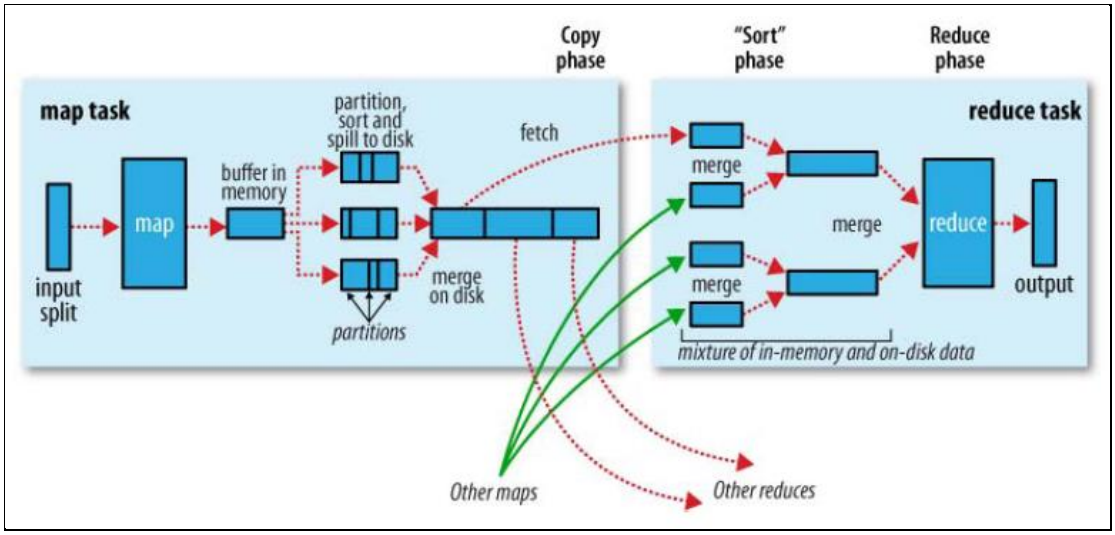
1、MapReduce 的 Shuffle 机制

1.1、概述

1、MapReduce 中，mapper 阶段处理的数据如何传递给 reducer 阶段，是 MapReduce 框架中最关键的一个流程，这个流程就叫 Shuffle
2、Shuffle: 数据混洗 ——（核心机制：数据分区，排序，局部聚合，缓存，拉取，再合并排序）
3、具体来说：就是将 MapTask 输出的处理结果数据，按照 Partitioner 组件制定的规则分发给 ReduceTask，并在分发的过程中，对数据按 key 进行了分区和排序

1.2、主要流程





Shuffle 是 MapReduce 处理流程中的一个核心过程，它的每一个处理步骤是分散在各个 map task 和 reduce task 节点上完成的，整体来看，分为核心 3 个操作：

1、分区 partition（如果 reduceTask 只有一个或者没有，那么 partition 将不起作用。设置没设置都相当于没有）

2、Sort 根据 key 排序（MapReduce 编程中的 sort 是一定会做的，并且只能按照 key 排序，当然如果没有 reducer 阶段，那么就不会对 key 排序）

3、Combiner 进行局部 value 的合并（Combiner 是可选的组件，作用只是为了提高任务的执行效率）

1.3、详细流程

1、mapTask 收集我们的 map()方法输出的 kv 对，放到内存缓冲区 kvbuffer（环形缓冲区：内存中的一种首尾相连的数据结构，kvbuffer 包含数据区和索引区）中

2、从内存缓冲区中的数据区的数据不断溢出本地磁盘文件 file.out，可能会溢出多次，则会有多个文件，相应的内存缓冲区中的索引区数据溢出为磁盘索引文件 file.out.index

3、多个溢出文件会被合并成大的溢出文件

4、在溢出过程中，及合并的过程中，都要调用 partitoner 进行分区和针对 key 进行排序

5、在数据量大的时候，可以对 mapTask 结果启用压缩，将 mapreduce.map.output.compress 设为 true，并使用 mapreduce.map.output.compress.codec 设置使用的压缩算法，可以提高数据传输到 reducer 端的效率

6、reduceTask 根据自己的分区号，去各个 mapTask 机器上取相应的结果分区数据

7、reduceTask 会取到同一个分区的来自不同 mapTask 的结果文件，reduceTask 会将这些文件再进行合并（归并排序）

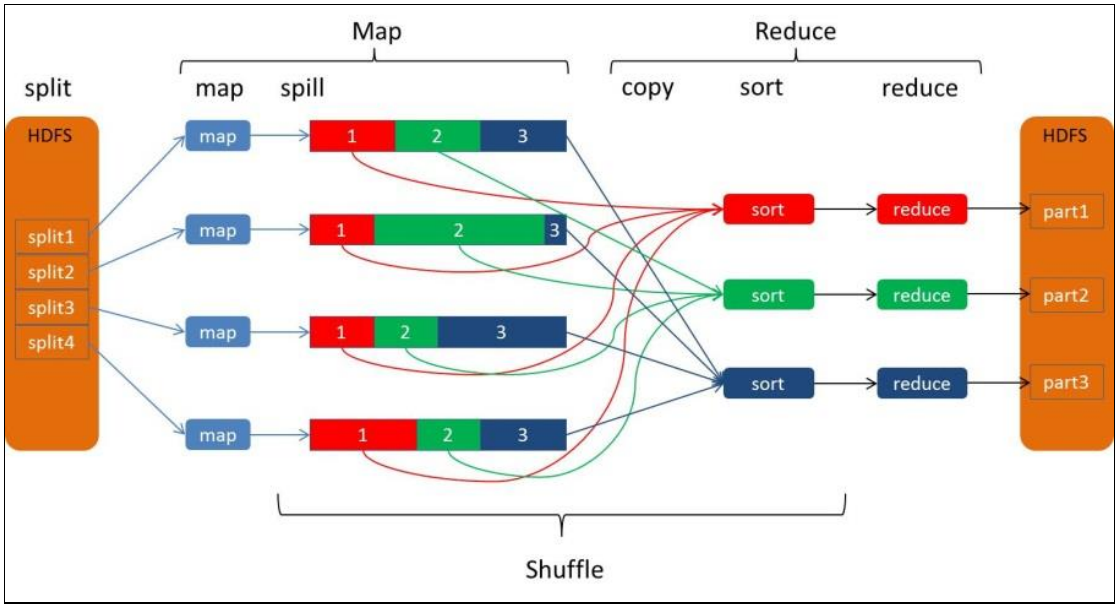
8、合并成大文件后，shuffle 的过程也就结束了，后面进入 reduceTask 的逻辑运算过程（从文件中取出一个一个的键值对 group，调用用户自定义的 reduce()方法）

Shuffle 中的缓冲区大小会影响到 mapreduce 程序的执行效率，原则上说，缓冲区越大，磁盘 io 的次数越少，执行速度就越快

缓冲区的大小可以通过参数调整，参数：mapreduce.task.io.sort.mb　默认 100M

缓冲区的溢写比也可以通过参数调整，参数：mapreduce.map.sort.spill.percent 默认 0.8

1.4、流程图



1.5、MapReduce 超详细执行流程解读

1、一个大文件需要处理，它在在 HDFS 上是以 block 块形式存放，每个 block 默认为 128M 存 3 份，运行时每个 map 任务会处理一个 split，如果 block 大和 split 相同（默认情况下确实相同），有多少个 block 就有多少个 map 任务，所以对整个文件处理时会有很多 map 任务进行并行计算

2、每个 map 任务处理完输入的 split 后会把结果写入到内存的一个环形缓冲区，写入过程中会进行简单排序，它的默认大小为 100M,当缓冲区的大小使用超过一定的阀值，一个后台的线程就会启动把缓冲区中的数据溢写(spill)到本地磁盘中(mapred-site.xml:mapreduce.cluster.local.dir)，同 Mapper 继续时向环形缓冲区中写入数据；

3、数据溢写入到磁盘之前，首先会根据 reducer 的数量划分成同数量的分区(partition)，每个分区中的都数据会有后台线程根据 map 任务的输出结果 key2 进行内排序(字典顺序、自然数顺序或自定义顺序 comparator)，如果有 combiner，它会在溢写到磁盘之前排好序的输出上运行(combiner 的作用是使 map 输出更紧凑，写到本地磁盘和传给 reducer 的数据更少)，最后在本地生成分好区且排好序的小文件； 如果 map 向环形缓冲区写入数据的速度大于向本地写入数据的速度，环形缓冲区被写满，向环形缓冲区写入数据的线程会阻塞直至缓冲区中的内容全部溢写到磁盘后再次启动，到阀值后会向本地磁盘新建一个溢写文件；

4、map 任务完成之前，会把本地磁盘溢写的所有文件不停地合并成得到一个结果文件，合并得到的结果文件会根据小溢写文件的分区而分区，每个分区的数据会再次根据 key2 进行排序，得到的结果文件是分好区且排好序的，可以合并成一个文件的溢写文件数量默认为 10(mapred-site.xml:mapreduce.task.io.sort.factor)；这个结果文件的分区存在一个映射关系，

比如 0~1024 字节内容为 0 号分区内容，1025~4096 字节内容为 1 号分区内容等等；

5、reduce 任务启动，Reducer 个数由 mapred-site.xml 的 mapreduce.job.reduces 配置决定，或者初始化 job 时调用 Job.setNumReduceTasks(int)；Reducer 中的一个线程定期向 MRAppMaster 询问 Mapper 输出结果文件位置，mapper 结束后会向 MRAppMaster 汇报信息；从而 Reducer 得知 Mapper 状态，得到 map 结果文件目录；

6、当有一个 Mapper 结束时，reduce 任务进入复制阶段，reduce 任务通过 http 协议(hadoop 内置了 netty 容器)把所有 Mapper 结果文件的对应的分区数据复制过来，比如，编号为 0 的 reducer 复制 map 结果文件中 0 号分区数据，1 号 reduce 复制 map 结果文件中 1 号分区的数据等等，Reducer 可以并行复制 Mapper 的结果，默认线程数为 5(mapred-site.xml:mapreduce.reduce.shuffle.parallelcopies)；
所有 Reducer 复制完成 map 结果文件后，由于 Reducer 会失败，NodeManager 并没有在第一个 map 结果文件复制完成后删除它，直到作业完成后 MRAppMaster 通知 NodeManager 进行删除；
另外：如果 map 结果文件相当小，则会被直接复制到 reduce NodeManager 的内存中(缓冲区大小由 mapred-site.xml:mapreduce.reduce.shuffle.input.buffer.percent 指定，默认 0.7)；一旦缓冲区达到 reduce 的阈值大小 0.66(mapred-site.xml:mapreduce.reduce.shuffle.merge.percent)或写入到 reduce NodeManager 内存中文件个数达到 map 输出阈值 1000(mapred-site.xml:mapreduce.reduce.merge.inmem.threshold)，reduce 就会把 map 结果文件合并溢写到本地；

7、复制阶段完成后，Reducer 进入 Merge 阶段，循环地合并 map 结果文件，维持其顺序排序，合并因子默认为 10(mapred-site.xml:mapreduce.task.io.sort.factor)，经过不断地 Merge 后得到一个“最终文件”，可能存储在磁盘也可能存在内存中；

8、“最终文件”输入到 reduce 进行计算，计算结果输入到 HDFS。