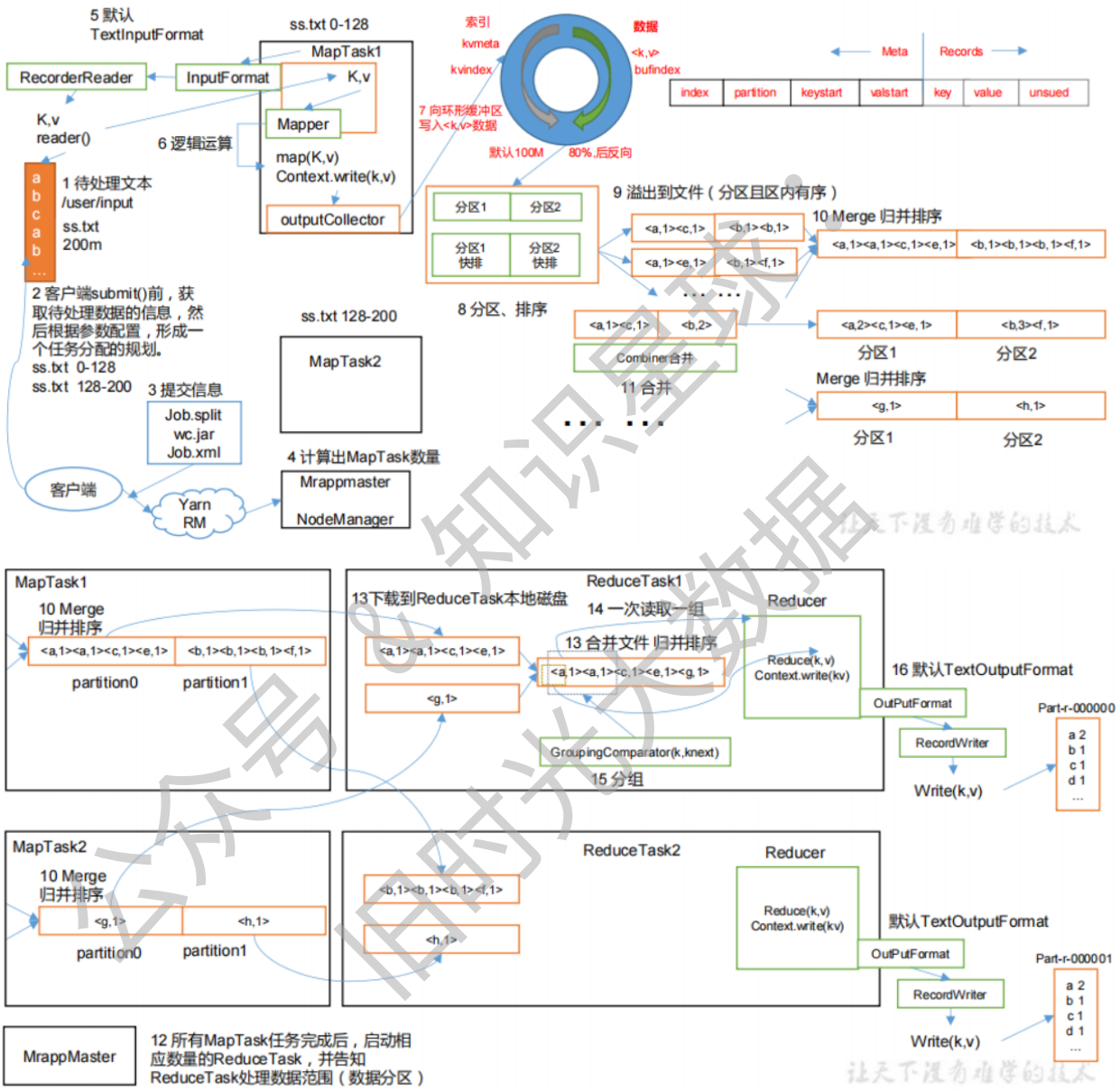
MapReduce工作原理

参考答案：



MapReduce详细流程：

1、准备待处理文件（200M）

2、submit()对原始文件进行切片分析（128M一块，这里0-128M为第一块，128-200位第二块）

3、提交信息

客户端会准备三样东西（Job的切片：Job.split，jar包：wc.jar，xml：Job.xml），对这个集群进行提交。

如果是本地模式，则没有jar包，正常生产模式下，都是用的YARN集群模式，所以jar包是一定有的。

4、计算出MapTask数量

客户端向YARN提交后，YARN会开启一个Mrappmaster（整个任务运行的老大），Mrappmaster会读取客户端对应的信息，最主要的就是读取切片信息（Job.split），Mrappmaster会根据切片个数，开启对应数量的MapTask，这里是两个切片，则对应开启两个MapTask。

5、MapTask工作

MapTask一启动后，就会开始工作，MapTask通过InputFormat来读取输入的文件，默认使用TextInputFormat，TextInputFormat源码里面有两个方法：1）RecorderReader（按行进行读取）和 2）isSplitable（判断这个文件是否可以切割），通过RecorderReader对文件数据按行进行读取，读完后，返回K（整个文件中的起始字节偏移量）和V（这行的内容）。

6、逻辑运算

数据读取完成后，将数据给到Mapper，Mapper里面是用户根据实际需求自己写的业务逻辑代码。数据处理完成后，通过outputCollector向环形缓冲器写入<k，v>数据。

7、向环形缓冲器写入<k，v>数据

环形缓存区默认大小为100M，左侧存索引，右侧存数据。当环形缓冲区数据写到80%时，进行反向写，将数据写入到磁盘。

这里可能会有一个小问题：为什么不存到100%才开始反向写？

主要是为了不影响Mapper处理完的数据写入环形缓冲区，如果写到100%环形缓冲区再反向写，就会导致Mapper处理后的数据无法再接着写入环形缓冲区，需要等环形缓冲区的数据反写到磁盘，才能继续写入环形缓冲区，影响处理进度，所以当环形缓冲区数据写到80%时，则开始进行反写，留下20%空间可以进行写入Mapper处理完的数据。

这里可能又有一个问题：如果当环形缓冲区数据到80%后反写，但是Mapper处理完的数据写入速度大于反写到磁盘的速度，导致环形缓冲区存储100%怎么办？

这个问题底层已经做过处理，如果Mapper处理完的数据写入速度大于反写到磁盘的速度，导致环形缓冲区存储100%，此时数据写入会暂定，直到有一定空间后，才会继续往环形缓冲区中写入数据，避免处理误差。

8、分区、排序

当数据写入到环形缓冲区时，也就是写入之前已经进行了分区，后续会根据分区，数据分别进入到对应的Reduce中独立进行处理。当数据溢写到磁盘时，会在溢写前进行排序，对数据索引进行排序，**此时使用快排方式进行排序。**

9、溢出文件到磁盘（分区且区内有序）

当环形缓冲区中数据达到80%时，数据就会溢写到磁盘上，溢写文件可以是多个。

10、Merge归并排序

通过归并排序对所有溢写文件根据分区进行排序，保证每一个分区内数据有序。归并排序后数据就会存储到磁盘上。

11、Combiner合并

这个环节是发生在归并排序前面，Combiner合并会将每个分区中k相同的数据进行预聚合，比如一个分区存在<a,1>和<a,1>两个数据，则会合并成<a,2>，提高传输效率。

12、启动ReduceTask

当所有的MapTask任务完成后，则启动相对应数量的ReduceTask，并告知ReduceTask处理数据范围（数据分区）。注意，并不一定是当所有的MapTask任务完成后才会开启ReduceTask，当我们MapTask数量比较多的时候，可以设置当完成多少个MapTask任务后就开启ReduceTask任务，处理完的数据再跟后面MapTask处理好的数据一起处理，提升处理效率。

13、下载数据到ReduceTask

ReduceTask主动从MapTask对应指定的分区拉取数据，再对来自不同分区的数据进行合并、**归并排序。**

14、分组（可选流程）

此时可根据实际情况，是否选择进行一次分组。

15、Reduce读取数据

Reduce方法中每次对一组数据（相同key）进行处理，数据处理完成后，则往外写数据。

16、往外写数据

通过OutputFormat往外写数据，OutPutFormat中包含RecordWriter方法，通过RecordWriter方法往外写数据，从而形成输出文件。其它的Reduce也是一样的流程。

上面的流程是整个MapReduce最全工作流程，Shuffle过程是从第7步开始到第16步结束，Shuffle大概过程如下：

1）MapTask收集我们的map()方法输出的kv对，放到内存缓冲区中

2）从内存缓冲区不断溢出本地磁盘文件，可能会溢出多个文件

3）多个溢出文件会被合并成大的溢出文件

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

5）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

6）ReduceTask会抓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）

7）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）