1、你能简单描述一下 Hbase 吗？能画出它的架构图吗？

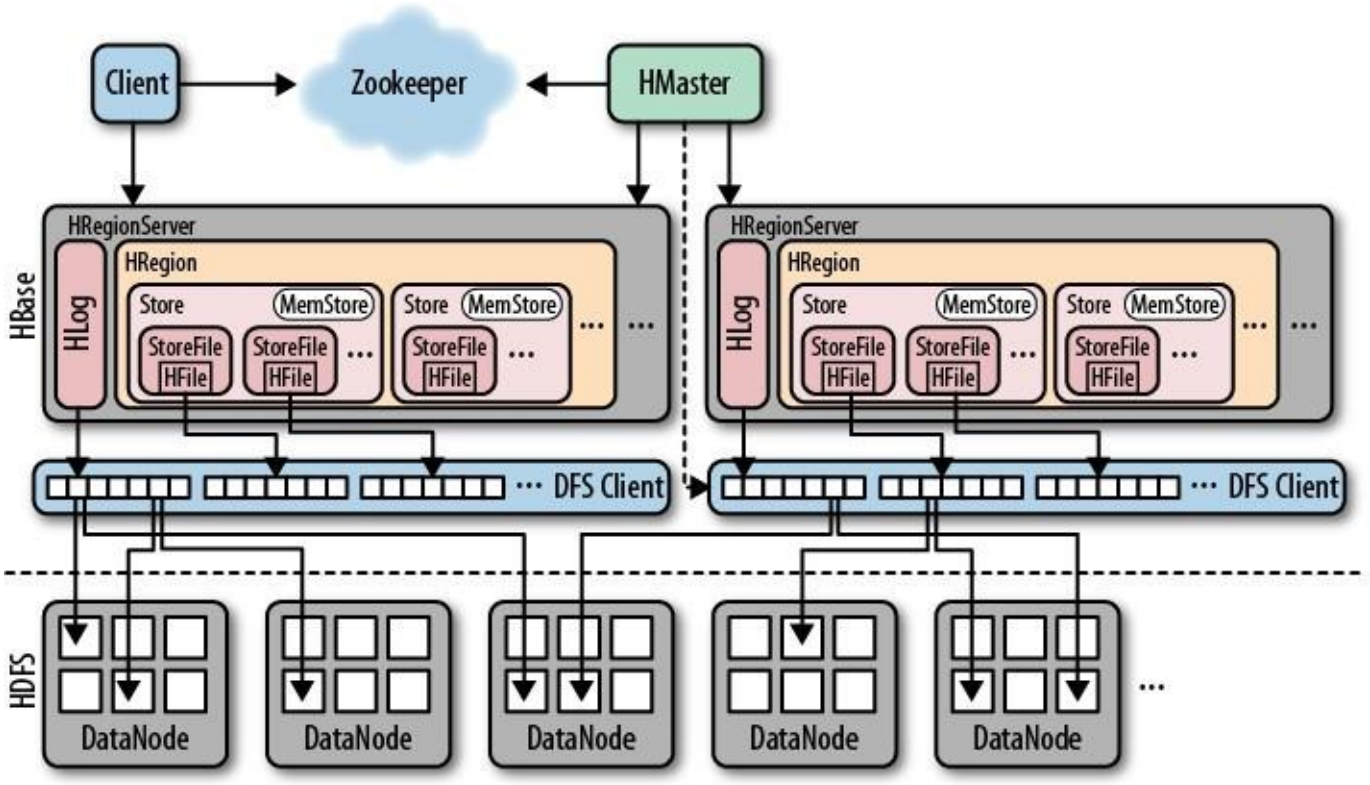
HBase 是一个面向列的 NoSQL 分布式**数据库**，它利用 HDFS 作为底层存储系统。那么，HBase 相对于传统的关系型数据库有什么不同呢？

- HBase 是 schema-free 的，它的列是可以动态增加的（仅仅定义列族），并且为空的列不占物理存储空间。
- HBase 是基于列存储的，每个列族都由几个文件保存，不同的列族的文件是分离的。
- HBase 自动切分数据，使得数据存储自动具有很好的横向扩展性。
- HBase 没有任何事务，提供了高并发读写操作的支持。

HBase 中的 Table 是一个稀疏的、多维度的、排序的映射表，这张表的索引是[RowKey, ColumnFamily, ColumnQualifier, Timestamp]，其中 Timestamp 表示版本，默认获取最新版本。HBase 是通过 RowKey 来检索数据的，RowKey 是 Table 设计的核心，它按照 ASCII 有序排序，因此应尽量避免顺序写入。RowKey 设计应该注意三点：

- 唯一原则：在 HBase 中 rowkey 可以看成是表的主键，必须保证其唯一性。
- 散列原则：由于 rowkey 是按字典有序的，故应避免 rowkey 连续有序而导致在某一台 RegionServer 上堆积的现象。例如可以拼接随机数、将时间戳倒序等。
- 长度原则：设计时 RowKey 要尽量短，这样可以提高有效数据的比例，节省存储空间，也可以提高查询的性能。

下面是 HBase 的整体架构图：



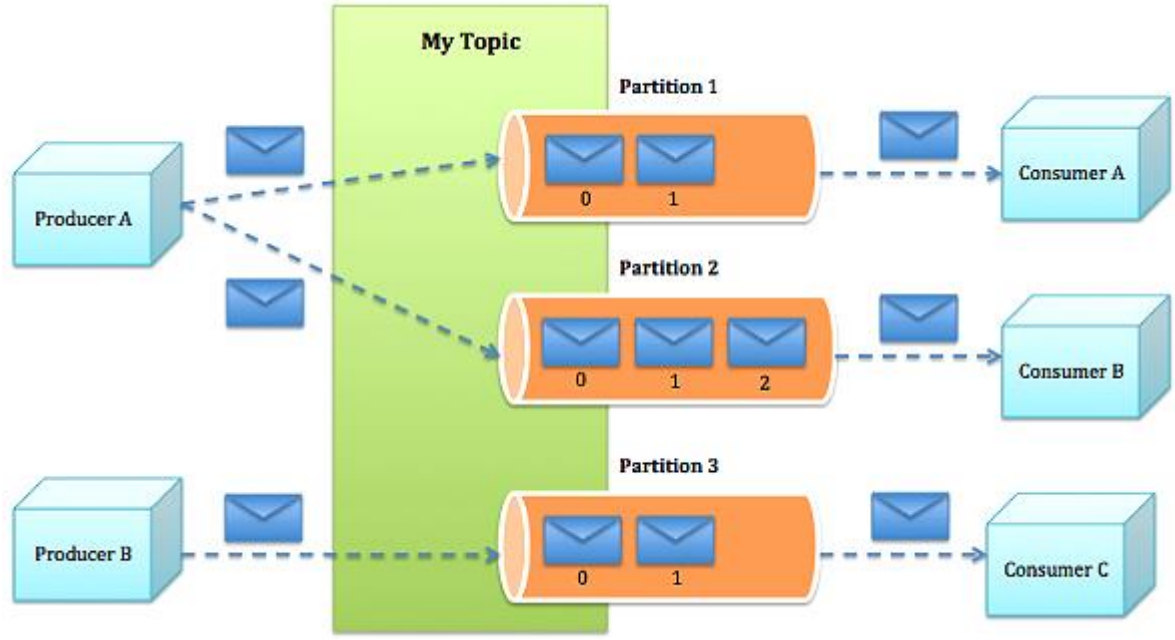
2、你说了解 kafka，能简单描述一下 Kafka 吗？能画出它的架构图吗？

Kafka 是一个高吞吐、易扩展的分布式发布-订阅消息系统，它能够将消息持久化到磁盘，用于批量的消费。Kafka 中有以下几个概念：

- Topic：特指 Kafka 处理的消息源（feeds of messages）的不同分类。

- Partition：Topic 物理上的分组，一个 topic 可以分为多个 partition，每个 partition 是一个有序的队列。partition 中的每条消息都会被分配一个有序的 id（offset）。
- Broker：Kafa 集群中包含一台或多台服务器，这种服务器被称为 broker。
- Producer：生产者，向 Kafka 的一个 topic 发布消息。
- Consumers：消费者，从 kafka 的某个 topic 读取消息。

Kafka 架构图如下：



详见：Apache Kafka：下一代分布式消息系统

http://www.infoq.com/cn/articles/apache-kafka/

4、请介绍一下 MapReduce 的工作原理。

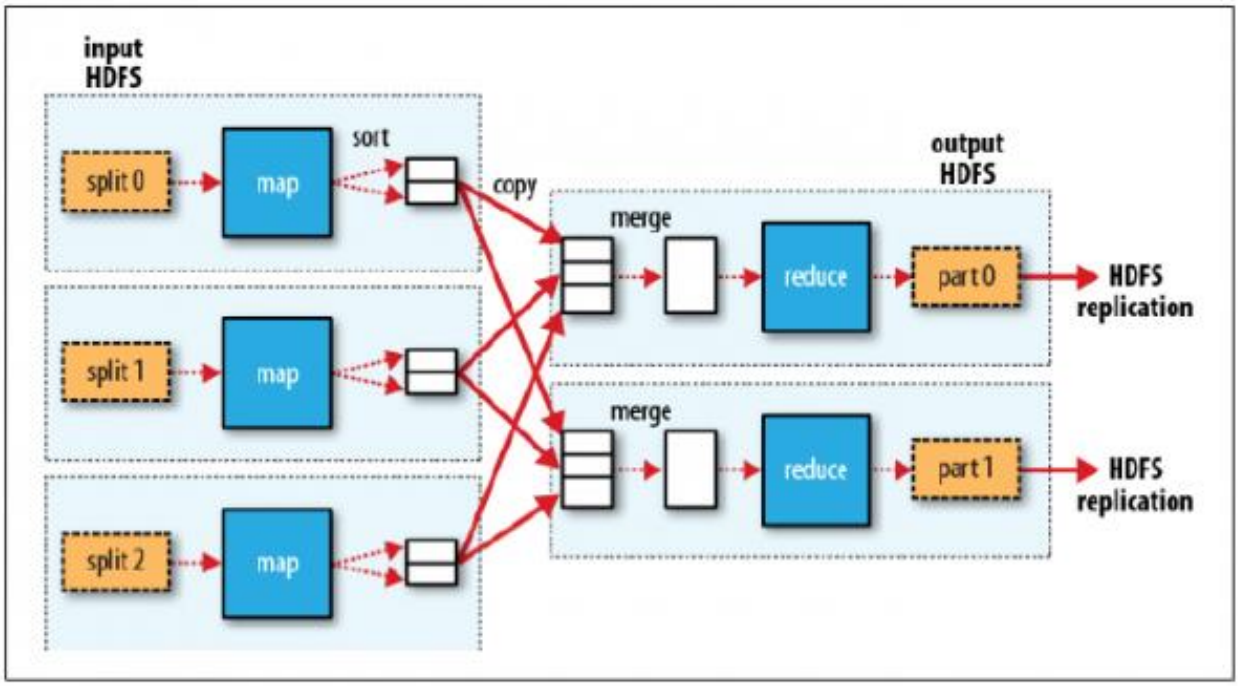
【解】MapReduce 是一个分布式计算框架，用于大规模数据集的并行运算。简单地说，MapReduce 就是"任务的分解与结果的汇总"：将一个大的数据处理任务划分成许多个子任务，并将这些子任务分配给各个节点并行处理，然后通过整合各个节点的中间结果，得到最终结果。

MapReduce 是主从架构，在 master 上跑的是 JobTracker/ResourceManager，负责资源分配与任务调度；而各个 slave 上跑的是 TaskTracker/NodeManager，负责执行任务，并定期向 master 汇报最新状态与执行进度。

对于一个 MR 任务，它的输入、输出以及中间结果都是`<key, value>`键值对：

- Map：`<k1, v1>` ——> `list(<k2, v2>)`
- Reduce：`<k2, list(v2)>` ——> `list(<k3, v3>)`

MR 程序的执行过程主要分为三步：Map 阶段、Shuffle 阶段、Reduce 阶段，如下图：



1. **Map 阶段**

    - 分片（Split）：map 阶段的输入通常是 HDFS 上文件，在运行 Mapper 前，FileInputFormat 会将输入文件分割成多个 split ——1 个 split 至少包含 1 个 HDFS 的 Block（默认为 64M）；然后每一个分片运行一个 map 进行处理。

    - 执行（Map）：对输入分片中的每个键值对调用 `map()` 函数进行运算，然后输出一个结果键值对。

        - Partitioner：对 map 函数的输出进行 partition，即根据 key 或 value 及 reduce 的数量来决定当前的这对键值对最终应该交由哪个 reduce 处理。默认是对 key 哈希后再以 reduce task 数量取模，默认的取模方式只是为了避免数据倾斜。然后该 key/value 对以及 partitionIdx 的结果都会被写入环形缓冲区。
    - 溢写（Spill）：map 输出写在内存中的环形缓冲区，默认当缓冲区满 80%，启动溢写线程，将缓冲的数据写出到磁盘。

        - Sort：在溢写到磁盘之前，使用快排对缓冲区数据按照 partitionIdx, key 排序。（每个 partitionIdx 表示一个分区，一个分区对应一个 reduce）
        - Combiner：如果设置了 Combiner，那么在 Sort 之后，还会对具有相同 key 的键值对进行合并，减少溢写到磁盘的数据量。
    - 合并（Merge）：溢写可能会生成多个文件，这时需要将多个文件合并成一个文件。合并的过程中会不断地进行 sort & combine 操作，最后合并成了一个已分区且已排序的文件。

2. **Shuffle 阶段**：广义上 Shuffle 阶段横跨 Map 端和 Reduce 端，在 Map 端包括 Spill 过程，在 Reduce 端包括 copy 和 merge/sort 过程。通常认为 Shuffle 阶段就是将 map 的输出作为 reduce 的输入的过程

    - Copy 过程：Reduce 端启动一些 copy 线程，通过 HTTP 方式将 map 端输出文件中属于自己的部分拉取到本地。Reduce 会从多个 map 端拉取数据，并且每个 map 的数据都是有序的。

    - Merge 过程：Copy 过来的数据会先放入内存缓冲区中，这里的缓冲区比较大；当缓冲区数据量达到一定阈值时，将数据溢写到磁盘（与 map 端类似，溢写过程会执行 sort & combine）。如果生成了多个溢写文件，它们会被 merge 成一个**有序的最终文件**。这个过程也会不停地执行 sort & combine 操作。

3. **Reduce 阶段**：Shuffle 阶段最终生成了一个有序的文件作为 Reduce 的输入，对于该文件中的每一个键值对调用 `reduce()` 方法，并将结果写到 HDFS。

---

mapreduce 是 hadoop 的核心之一，mapreduce 经常让我们产生各种困惑，我们只是知道什么是 map，什么是 renduce，甚至我们已经熟悉了 mapreduce 编程，但是内部的原理还是不明白。下面在回帖中，给大家解决部分问题。更多问题有待挖掘。

1.Shuffle 的定义是什么？
2.map task 与 reduce task 的执行是否在不同的节点上？
3.Shuffle 产生的意义是什么？
4.每个 map task 都有一个内存缓冲区，存储着 map 的输出结果，当缓冲区快满的时候需要将缓冲区的数据该如何处理？
5.在 map task 执行时，它是如何读取 HDFS 的？
6.读取的 Split 与 block 的对应关系可能是什么？
7.MapReduce 提供 Partitioner 接口，它的作用是什么？
8.溢写是在什么情况下发生？
9.溢写是为什么不影响往缓冲区写 map 结果的线程？
10.当溢写线程启动后，需要对这 80MB 空间内的 key 做排序(Sort)。排序是 MapReduce 模型默认的行为，这里的排序也是对谁的排序？
11.哪些场景才能使用 Combiner 呢？
12.Merge 的作用是什么？
13.reduce 中 Copy 过程采用是什么协议？
14.reduce 中 merge 过程有几种方式，与 map 有什么相似之处？
15.溢写过程中如果有很多个 key/value 对需要发送到某个 reduce 端去，那么如何处理这些 key/value 值

个人观点仅供参考：

Shuffle 产生的意义是什么？

Shuffle 过程的期望可以有：
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。

每个 map task 都有一个内存缓冲区，存储着 map 的输出结果，当缓冲区快满的时候需要将缓冲区的数据该如何处理？

每个 map task 都有一个内存缓冲区，存储着 map 的输出结果，当缓冲区快满的时候需要将缓冲区的数据以一个临时文件的方式存放到磁盘，当整个 map task 结束后再对磁盘中这个 map task 产生的所有临时文件做合并，生成最终的正式输出文件，然后等待 reduce task 来拉数据。

MapReduce 提供 Partitioner 接口，它的作用是什么？

MapReduce 提供 Partitioner 接口，它的作用就是根据 key 或 value 及 reduce 的数量来决定当前的这对输出数据最终应该交由哪个 reduce task 处理。默认对 key hash 后再以 reduce task 数量取模。默认的取模方式只是为了平均 reduce 的处理能力，如果用户自己对 Partitioner 有需求，可以订制并设置到 job 上。

什么是溢写？

在一定条件下将缓冲区中的数据临时写入磁盘，然后重新利用这块缓冲区。这个从内存往磁盘写数据的过程被称为 Spill，中文可译为溢写。

溢写是为什么不影响往缓冲区写 map 结果的线程？

溢写线程启动时不应该阻止 map 的结果输出，所以整个缓冲区有个溢写的比例 spill.percent。这个比例默认是 0.8，也就是当缓冲区的数据已经达到阈值（buffer size * spill percent = 100MB * 0.8 = 80MB），溢写线程启动，锁定这 80MB 的内存，执行溢写过程。Map task 的输出结果还可以往剩下的 20MB 内存中写，互不影响。

当溢写线程启动后，需要对这 80MB 空间内的 key 做排序(Sort)。排序是 MapReduce 模型默认的行为，这里的排序也是对谁的排序？

当溢写线程启动后，需要对这 80MB 空间内的 key 做排序(Sort)。排序是 MapReduce 模型默认的行为，这里的排序也是对序列化的字节做的排序。

溢写过程中如果有很多个 key/value 对需要发送到某个 reduce 端去，那么如何处理这些 key/value 值？

如果有很多个 key/value 对需要发送到某个 reduce 端去，那么需要将这些 key/value 值拼接到一块，减少与 partition 相关的索引记录。

哪些场景才能使用 Combiner 呢？

Combiner 的输出是 Reducer 的输入，Combiner 绝不能改变最终的计算结果。所以从我的想法来看，Combiner 只应该用于那种 Reduce 的输入 key/value 与输出 key/value 类型完全一致，且不影响最终结果的场景。比如累加，最大值等。Combiner 的使用一定得慎重，如果用好，它对 job 执行效率有帮助，反之会影响 reduce 的最终结果。

Merge 的作用是什么？

最终磁盘中会至少有一个这样的溢写文件存在(如果 map 的输出结果很少，当 map 执行完成时，只会产生一个溢写文件)，因为最终的文件只有一个，所以需要将这些溢写文件归并到一起，这个过程就叫做 Merge

每个 reduce task 不断的通过什么协议从 JobTracker 那里获取 map task 是否完成的信息？

每个 reduce task 不断地通过 RPC 从 JobTracker 那里获取 map task 是否完成的信息

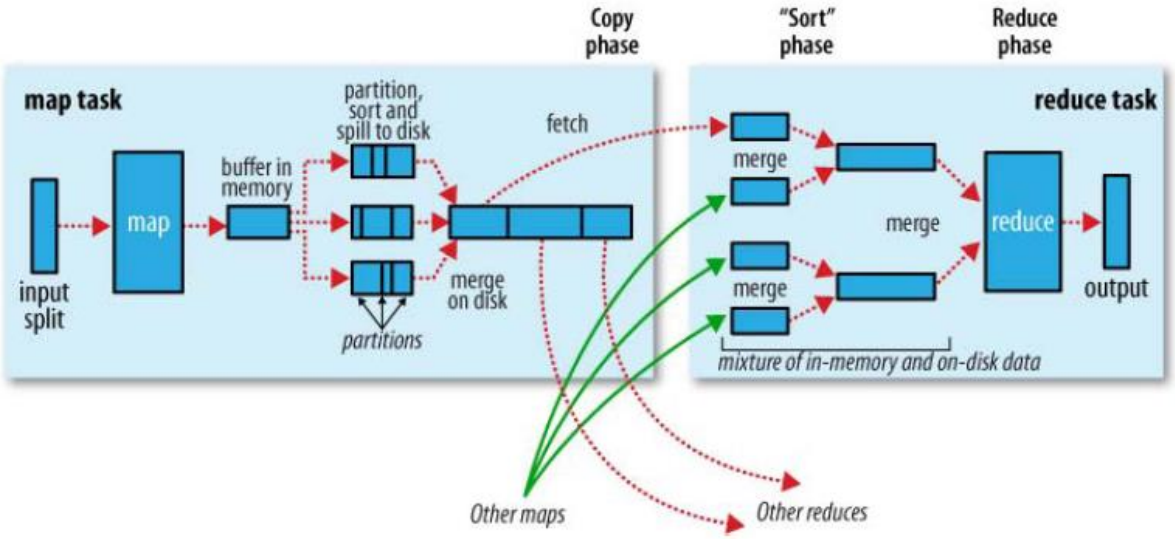
reduce 中 Copy 过程采用是什么协议？

Copy 过程，简单地拉取数据。Reduce 进程启动一些数据 copy 线程(Fetcher)，通过 HTTP 方式请求 map task 所在的 TaskTracker 获取 map task 的输出文件。

reduce 中 merge 过程有几种方式？

merge 有三种形式:1)内存到内存 2)内存到磁盘 3)磁盘到磁盘。默认情况下第一种形式不启用,让人比较困惑，是吧。当内存中的数据量到达一定阈值，就启动内存到磁盘的 merge。与 map 端类似，这也是溢写的过程，这个过程中如果你设置有 Combiner，也是会启用的，然后在磁盘中生成了众多的溢写文件。第二种 merge 方式一直在运行，直到没有 map 端的数据时才结束，然后启动第三种磁盘到磁盘的 merge 方式生成最终的那个文件。

Shuffle 的正常意思是洗牌或弄乱，可能大家更熟悉的是 Java API 里的 Collections.shuffle(List)方法，它会随机地打乱参数 list 里的元素顺序。如果你不知道 MapReduce 里 Shuffle 是什么，那么请看这张图：



这张是官方对 Shuffle 过程的描述。但我可以肯定的是，单从这张图你基本不可能明白 Shuffle 的过程，因为它与事实相差挺多，细节也是错乱的。后面我会具体描述 Shuffle 的事实情况，所以这里你只要清楚 Shuffle 的大致范围就成－怎样把 map task 的输出结果有效地传送到 reduce 端。也可以这样理解， Shuffle 描述着数据从 map task 输出到 reduce task 输入的这段过程。

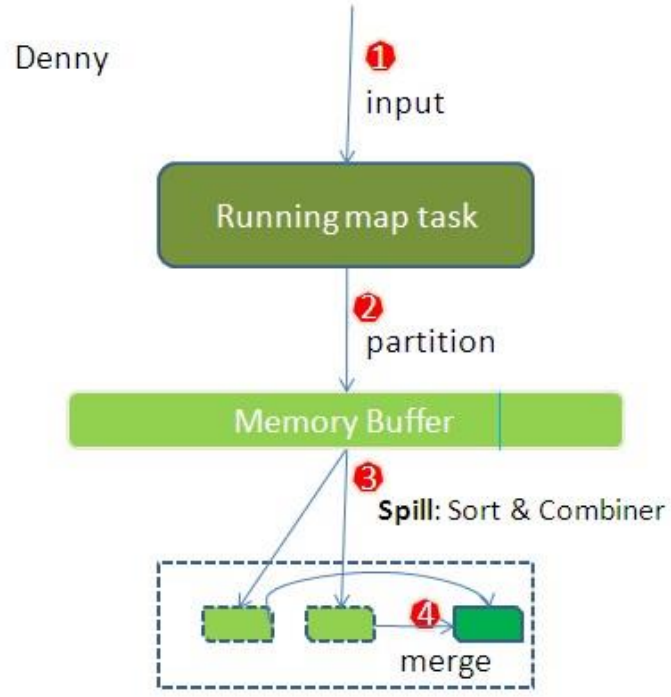
在 Hadoop 这样的集群环境中，大部分 map task 与 reduce task 的执行是在不同的节点上。当然很多情况下 Reduce 执行时需要跨节点去拉取其它节点上的 map task 结果。如果集群正在运行的 job 有很多，那么 task 的正常执行对集群内部的网络资源消耗会很严重。这种网络消耗是正常的，我们不能限制，能做的就是最大化地减少不必要的消耗。还有在节点内，相比于内存，磁盘 IO 对 job 完成时间的影响也是可观的。从最基本的要求来说，我们对 Shuffle 过程的期望可以有：

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。

OK，看到这里时，大家可以先停下来想想，如果是自己来设计这段 Shuffle 过程，那么你的设计目标是什么。我想能优化的地方主要在于减少拉取数据的量及尽量使用内存而不是磁盘。

我的分析是基于 Hadoop0.21.0 的源码，如果与你所认识的 Shuffle 过程有差别，不吝指出。我会以 WordCount 为例，并假设它有 8 个 map task 和 3 个 reduce task。从上图看出，Shuffle 过程横跨 map 与 reduce 两端，所以下面我也会分两部分来展开。



先看看 map 端的情况，如下图：

上图可能是某个 map task 的运行情况。拿它与官方图的左半边比较，会发现很多不一致。官方图没有清楚地说明 partition，sort 与 combiner 到底作用在哪个阶段。我画了这张图，希望让大家清晰地了解从 map 数据输入到 map 端所有数据准备好的全过程。

整个流程我分了四步。简单些可以这样说，每个 map task 都有一个内存缓冲区，存储着 map 的输出结果，当缓冲区快满的时候需要将缓冲区的数据以一个临时文件的方式存放到磁盘，当整个 map task 结束后再对磁盘中这个 map task 产生的所有临时文件做

合并，生成最终的正式输出文件，然后等待 reduce task 来拉数据。

当然这里的每一步都可能包含着多个步骤与细节，下面我对细节来一一说明：

1. 在 map task 执行时，它的输入数据来源于 HDFS 的 block，当然在 MapReduce 概念中，map task 只读取 split。Split 与 block 的对应关系可能是多对一，默认是一对一。在 WordCount 例子里，假设 map 的输入数据都是像“aaa”这样的字符串。

2. 在经过 mapper 的运行后，我们得知 mapper 的输出是这样一个 key/value 对： key 是“aaa”， value 是数值 1。因为当前 map 端只做加 1 的操作，在 reduce task 里才去合并结果集。前面我们知道这个 job 有 3 个 reduce task，到底当前的“aaa”应该交由哪个 reduce 去做呢，是需要现在决定的。

MapReduce 提供 Partitioner 接口，它的作用就是根据 key 或 value 及 reduce 的数量来决定当前的这对输出数据最终应该交由哪个 reduce task 处理。默认对 key hash 后再以 reduce task 数量取模。默认的取模方式只是为了平均 reduce 的处理能力，如果用户自己对 Partitioner 有需求，可以订制并设置到 job 上。

在我们的例子中，“aaa”经过 Partitioner 后返回 0，也就是这对值应当交由第一个 reducer 来处理。接下来，需要将数据写入内存缓冲区中，缓冲区的作用是批量收集 map 结果，减少磁盘 IO 的影响。我们的 key/value 对以及 Partition 的结果都会被写入缓冲区。当然写入之前，key 与 value 值都会被序列化成字节数组。

整个内存缓冲区就是一个字节数组，它的字节索引及 key/value 存储结构我没有研究过。如果有朋友对它有研究，那么请大致描述下它的细节吧。

3. 这个内存缓冲区是有大小限制的，默认是 100MB。当 map task 的输出结果很多时，就可能会撑爆内存，所以需要在一定条件下将缓冲区中的数据临时写入磁盘，然后重新利用这块缓冲区。这个从内存往磁盘写数据的过程被称为 Spill，中文可译为溢写，字面意思很直观。这个溢写是由单独线程来完成，不影响往缓冲区写 map 结果的线程。溢写线程启动时不应该阻止 map 的结果输出，所以整个缓冲区有个溢写的比例 spill.percent。这个比例默认是 0.8，也就是当缓冲区的数据已经达到阈值（buffer size * spill percent = 100MB * 0.8 = 80MB），溢写线程启动，锁定这 80MB 的内存，执行溢写过程。Map task 的输出结果还可以往剩下的 20MB 内存中写，互不影响。

当溢写线程启动后，需要对这 80MB 空间内的 key 做排序(Sort)。排序是 MapReduce 模型默认的行为，这里的排序也是对序列化的字节做的排序。

在这里我们可以想想，因为 map task 的输出是需要发送到不同的 reduce 端去，而内存缓冲区没有对将发送到相同 reduce 端的数据做合并，那么这种合并应该是体现是磁盘文件中的。从官方图上也可以看到写到磁盘中的溢写文件是对不同的 reduce 端的数值做过合并。所以溢写过程一个很重要的细节在于，如果有很多个 key/value 对需要发送到某个 reduce 端去，那么需要将这些 key/value 值拼接到一块，减少与 partition 相关的索引记录。

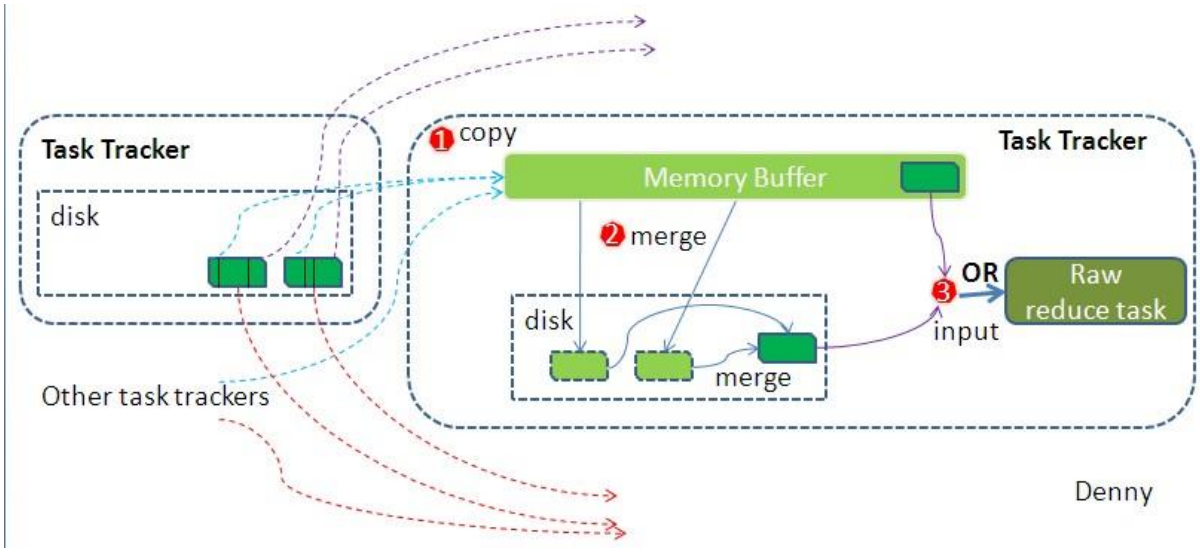
在针对每个 reduce 端而合并数据时，有些数据可能像这样：“aaa”/1， “aaa”/1。对于 WordCount 例子，就是简单地统计单词出现的次数，如果在同一个 map task 的结果中有很多个像“aaa”一样出现多次的 key，我们就应该把它们的值合并到一块，这个过程叫 reduce 也叫 combine。但 MapReduce 的术语中，reduce 只指 reduce 端执行从多个 map task 取数据做计算的过程。除 reduce 外，非正式地合并数据只能算做 combine 了。其实大家知道的，MapReduce 中将 Combiner 等同于 Reducer。

如果 client 设置过 Combiner，那么现在就是使用 Combiner 的时候了。将有相同 key 的 key/value 对的 value 加起来，减少溢写到磁盘的数据量。Combiner 会优化 MapReduce 的中间结果，所以它在整个模型中会多次使用。那哪些场景才能使用 Combiner 呢？从这里分析，Combiner 的输出是 Reducer 的输入，Combiner 绝不能改变最终的计算结果。所以从我的想法来看，Combiner 只应该用于那种 Reduce 的输入 key/value 与输出 key/value 类型完全一致，且不影响最终结果的场景。比如累加，最大值等。Combiner 的使用一定得慎重，如果用好，它对 job 执行效率有帮助，反之会影响 reduce 的最终结果。

4. 每次溢写会在磁盘上生成一个溢写文件，如果 map 的输出结果真的很大，有多次这样的溢写发生，磁盘上相应的就会有多个溢写文件存在。当 map task 真正完成时，内存缓冲区中的数据也全部溢写到磁盘中形成一个溢写文件。最终磁盘中会至少有一个这样的溢写文件存在(如果 map 的输出结果很少，当 map 执行完成时，只会产生一个溢写文件)，因为最终的文件只有一个，所以需要将这些溢写文件归并到一起，这个过程就叫做 Merge。Merge 是怎样的？如前面的例子，“aaa”从某个 map task 读取过来时值是 5,从另外一个 map 读取时值是 8,因为它们有相同的 key,所以得 merge 成 group。什么是 group。对于“aaa”就是像这样的:{“aaa”, [5, 8, 2, …]}，数组中的值就是从不同溢写文件中读取出来的，然后再把这些值加起来。请注意，因为 merge 是将多个溢写文件合并到一个文件，所以可能也有相同的 key 存在，在这个过程中如果 client 设置过 Combiner，也会使用 Combiner 来合并相同的 key。

至此，map 端的所有工作都已结束，最终生成的这个文件也存放在 TaskTracker 够得着的某个本地目录内。每个 reduce task 不断地通过 RPC 从 JobTracker 那里获取 map task 是否完成的信息，如果 reduce task 得到通知，获知某台 TaskTracker 上的 map task 执行完成，Shuffle 的后半段过程开始启动。

简单地说，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，也最终形成一个文件作为 reduce task 的输入文件。见下图：



如map 端的细节图,Shuffle在reduce端的过程也能用图上标明的三点来概括。当前reduce copy数据的前提是它要从JobTracker获得有哪些map task已执行结束，这段过程不表，有兴趣的朋友可以关注下。Reducer真正运行之前，所有的时间都是在拉取数据，做merge，且不断重复地在做。如前面的方式一样，下面我也分段地描述reduce 端的Shuffle细节：

1. Copy过程,简单地拉取数据。Reduce进程启动一些数据copy线程(Fetcher),通过HTTP方式请求map task所在的TaskTracker获取map task的输出文件。因为map task早已结束，这些文件就归TaskTracker管理在本地磁盘中。

2. Merge阶段。这里的merge如map端的merge动作，只是数组中存放的是不同map端copy来的数值。Copy过来的数据会先放入内存缓冲区中，这里的缓冲区大小要比map端的更为灵活，它基于JVM的heap size设置，因为Shuffle阶段Reducer不运行，所以应该把绝大部分的内存都给Shuffle用。这里需要强调的是，merge有三种形式：1)内存到内存 2)内存到磁盘 3)磁盘到磁盘。默认情况下第一种形式不启用，让人比较困惑，是吧。当内存中的数据量到达一定阈值，就启动内存到磁盘的merge。与map 端类似，这也是溢写的过程，这个过程中如果你设置有Combiner，也是会启用的，然后在磁盘中生成了众多的溢写文件。第二种merge方式一直在运行，直到没有map端的数据时才结束，然后启动第三种磁盘到磁盘的merge方式生成最终的那个文件。

3. Reducer的输入文件。不断地merge后，最后会生成一个“最终文件”。为什么加引号？因为这个文件可能存在于磁盘上，也可能存在于内存中。对我们来说，当然希望它存放于内存中，直接作为Reducer的输入，但默认情况下，这个文件是存放于磁盘中的。当Reducer的输入文件已定，整个Shuffle才最终结束。然后就是Reducer执行，把结果放到HDFS上

Kafka 原理剖析及实战演练

Kafka是流式处理系统如Spark streaming，Strom及Flink事实上的标准数据入口。本课程将分析Kafka的架构，Topic与Partition的关系，Kafka如何使用Consumer group实现group内的消息单播和group间的消息广播，Kafka如何利用Partition实现水平扩展以及Kafka如何实现高吞吐率。并结合源码分析Kafka实现数据复制，Leader election及Consumer rebalance的原理。同时介绍实用的Kafka监控工具kafka manager。并结合实例介绍Kafka如何与其它流行的开源系统(如Flume，Storm，Spark streaming）集成。

课程大纲：

第一课. Kafka简介

1.1 为什么需要消息系统

1.2 Kafka设计目标

1.3 如何安装和使用Kafka集群

第二课. Kafka架构

2.1 Kafka整体架构

2.2 Topic & Partition

2.3 Partitioner

2.4 Sync Producer vs. Async Producer

2.5 Producer重试机制

第三课. Kafka HA

3.1 Kafka一致性重要机制之ISR

3.2 Kafka 数据复制机制

3.3 Fail over

第四课. Zookeeper 与 Kafka

4.1 Zookeeper 典型用法

4.2 Zookeeper 使用注意事项

4.3 Kafka 如何使用 Zookeeper

第五课. Kafka 领导选举

5.1 领导选举算法

5.2 Kafka“各自为政”领导选举算法

5.3 Kafka 基于 Controller 的领导选举

第六课. Consumer

6.1 Pull vs Push

6.2 Low level API vs. High level API

6.3 单播 vs. 多播

6.4 Consumer rebalance

第七课. Consumer offset 管理

7.1 基于 Zookeeper 的 offset 管理

7.2 基于 broker 的 offset 管理

第八课. Consumer 的 stream 接口

8.1 Blocking 接口

8.2 Stream 接口

第九课. Kafka 高性能之道

9.1 顺序写磁盘

9.2 零拷贝

9.3 批处理

9.4 基于 ISR 的动态平衡一致性算法

第十课. kafka 监控工具

9.1 Zookeeper viewer

9.2 Kafka manager

第十一课. Kafka 运维

第十二课. Kafka 性能测试

授课时间：

课程预计 2016 年 11 月 19 日开课，预计课程持续时间为 14 周