**MapReduce七个步骤 :**

1. 用户程序中的MapReduce库首先**将输入文件分割成M个文件块，每块通常有16到64兆字节**(用户可以通过一个可选参数来控制)。然后它将启动集群机器上的程序的多个副本。
2. 这个程序的副本中有一份是特别的——the master.其他休眠的worker接收work通过master。存在着M个Map任务和R个reduce任务等待接收。The master 挑选闲置的线程并分配给它们每个一个map或者reduce任务。
3. 被分配map任务的工作者线程**读取相应的输入分割的内容**。**它从输入数据中解析出键/值对，并将每个键/值对传递给用户定义的Map函数。Map函数产生的中间键/值对被缓冲在内存中。**
4. 缓冲对 定期写入本地磁盘，**通过分区函数将其划分为R区域**。本地磁盘上这些缓冲对的locations被传递回master，master负责将这些locations转发给reduce workers。
5. 当 master 通知reduce worker这些locations时，**它使用远程过程调用(RPC)从map worker的本地磁盘读取缓冲数据。**当reduce worker读取了所有的中间数据时，它将按中间键对其进行排序，**以便将所有出现的相同键分组在一起**。**之所以需要排序，是因为通常有许多不同的键映射到同一个reduce任务**。如果中间数据量太大而无法装入内存，则使用**外部排序。**
6. reduce worker迭代已排序的中间数据，**对于遇到的每个唯一中间键，它将该键和相应的中间值集传递给用户的reduce函数。**Reduce函数的输出被附加到这个Reduce分区的最终输出文件中。
7. 当所有map任务和reduce任务完成时，the master 唤醒用户程序。此时，用户程序中的MapReduce调用返回到用户代码。

**容错 :**

**Worker Failure :**

The master 定期 pings 每个worker. 如果在一定的时间内没有收到worker的响应，则master将该worker标记为failed。worker完成的任何映射任务都被重置为初始空闲状态，因此可以对其他worker进行调度。类似地，在一个失败的worker上进行中的任何map任务或reduce任务也会被重置为空闲(idle)，并有资格重新调度。 完成的map任务在故障后将重新执行，因为它们的输出存储在故障机器的本地磁盘上，因此无法访问。已完成的reduce任务不需要重新执行，因为它们的输出存储在全局文件系统中。当map任务首先由worker A执行，然后再由worker B执行时(因为a失败)，所有执行reduce任务的worker都会收到重新执行的通知, 任何还没有从worker A读取数据的reduce任务都将从worker B读取数据。MapReduce对大规模worker失败具有弹性。例如，在一个MapReduce操作期间，运行集群上的网络维护导致80台机器的组在数分钟内无法访问。 MapReduce主程序简单地重新执行那些不可到达的worker所完成的工作，然后继续进行ward进程，最终完成MapReduce操作。

Master Failure :

让master定期写上面描述的这些结构的检查点是很容易的。如果the master任务终止，则可以从最后一个检查点状态启动一个新的副本。然而，考虑到只有一个master，它失败的可能性不大;因此，如果master失败，我们当前的实现将中止MapReduce计算。客户端可以检查这个条件，并重试MapReduce操作。

**3.4 Locality**

网络带宽在我们的计算环境中是一个相对稀缺的资源。我们利用这样一个事实，即输入数据(由GFS[8]管理)存储在组成集群的机器的本地磁盘上，从而节省网络带宽。GFS将每个文件划分为64 MB的块，并在不同的机器上存储每个块的几个副本(通常为3个副本)。MapReduce主程序考虑输入文件的位置信息，并尝试在一台包含相应输入数据副本的机器上调度map任务。如果失败，它会尝试在任务的输入数据附近安排一个map任务。例如:在与包含数据的机器在同一网络交换机上的工作机器上)。当在集群中相当一部分工作上运行大型MapReduce操作时，大多数输入数据都是在本地读取的，不消耗网络带宽。

**3.5 TaskGranularity**

We subdivide the map phase into M pieces and the reduce phase into R pieces,as described above. Ideally,M and R should be much larger than the number of worker machines. Having each worker perform many different tasks improves dynamic load balancing, and also speeds up recovery when a worker fails: the many map tasks it has completed can be spread out across all the other worker machines. There are practical bounds on how large M and R can be in our implementation, since the master must make O(M + R) scheduling decisions and keeps O(M ∗R) state in memory as described above. (The constant factors form emory usage are smal how ever: the O(M∗R) piece of the state consists of approximately one byte of data per map task/reducetask pair.)

我们将map阶段细分为M个部分，reduce阶段细分为R个部分，如上所述。理想情况下，M和R应该远远大于工作机器的数量。让每个worker执行许多不同的任务可以提高动态负载平衡，还可以加速worker失败时的恢复:它完成的许多映射任务可以分散到所有其他worker机器上。有多大的M和R可以在我们的实现，因为主人必须作出O(M + R)调度决策，并保持O(M∗R)的状态在内存如上所述。(艾莫利使用的常量因素很小:状态的O(M∗R)部分大约由每个map任务/reduce task对一个字节的数据组成。)

Furthermore, R is often constrained by users because the output of each reduce task ends up in a separate output ﬁle. In practice, we tend to choose M so that each individualtask is roughly 16MB to 64MB of input data (so that the locality optimization described above is most effective), and we make R a small multiple of the number of worker machines we expect to use. We often perform MapReduce computations with M = 200,000and R =5 ,000,using2,000 worker machines

此外，**R常常受到用户的限制，因为每个reduce任务的输出都在一个单独的输出文件中结束。在实践中，我们倾向于选择M**，这样每个单独的任务大约是16MB到64MB的输入数据(这样上面描述的局部优化是最有效的)，而我们使R是我们期望使用的工作机器数量的一个小倍数。我们经常使用M = 200000和R = 5,000进行MapReduce计算，使用2000台工人机器

**4.1 Partitioning Function**

The users of MapReduce specify the number of reduce tasks/output ﬁles that they desire (R). Data gets partitionedacross thesetasksusinga partitioningfunctionon  
the intermediate key. A default partitioning function is provided that uses hashing (e.g. “hash(key) mod R”). This tends to result in fairly well-balanced partitions. In some cases, however, it is useful to partition data by some other function of the key. For example, sometimes the output keys are URLs, and we want all entries for a single host to end up in the same output ﬁle. To support situations like this, the user of the MapReduce library can provideas pecial part itioning function. For example, using “hash(Hostname(urlkey)) mod R” as the partitioning function causesall URLs from the same host to end up in the same output ﬁle.

MapReduce的用户指定reduce任务/输出文件的数量，他们希望(R)

中间键。提供了一个使用哈希的默认分区函数(例如“hash(key) mod R”)。这往往会产生相当平衡的分区。但是，在某些情况下，根据键的其他函数对数据进行分区是有用的。例如，有时候输出键是url，我们希望单个主机的所有条目都出现在同一个输出文件中。为了支持这样的情况，MapReduce库的用户可以提供特殊的部分分配功能。例如，使用“hash(Hostname(urlkey)) mod R”作为分区函数会导致来自同一主机的所有url出现在相同的输出文件中。