**MapReduce 工作流程**

**Overview**

Hadoop MapReduce是一个软件框架，用于轻松编写应用程序，以可靠，容错的方式在大型集群（数千个节点）的商用硬件上并行处理大量数据（多TB数据集）。

MapReduce作业通常将输入数据集拆分为独立的块，这些块由map任务以完全并行的方式处理。 框架对地图的输出进行排序，然后输入到reduce任务。 通常，作业的输入和输出都存储在文件系统中。 该框架负责调度任务，监视任务并重新执行失败的任务。

通常，计算节点和存储节点是相同的，即MapReduce框架和Hadoop分布式文件系统（请参阅HDFS体系结构指南）在同一组节点上运行。 此配置允许框架有效地在已存在数据的节点上调度任务，从而在集群中产生非常高的聚合带宽。

MapReduce框架由单个主ResourceManager，每个集群节点一个从NodeManager和每个应用程序的MRAppMaster组成（参见YARN体系结构指南）。

最低限度，应用程序通过适当的接口和/或抽象类的实现来指定输入/输出位置并提供映射和减少功能。 这些和其他作业参数包括作业配置。

然后，Hadoop作业客户端将作业（jar /可执行文件等）和配置提交给ResourceManager，然后ResourceManager负责将软件/配置分发给从站，调度任务并监视它们，为作业提供状态和诊断信息 - 客户。

虽然Hadoop框架是用Java™实现的，但MapReduce应用程序不需要用Java编写。

Hadoop Streaming是一个实用程序，它允许用户使用任何可执行文件（例如shell实用程序）作为映射器和/或reducer创建和运行作业。

Hadoop Pipes是一个SWIG兼容的C ++ API，用于实现MapReduce应用程序（非基于JNI™）。

Inputs and Outputs

MapReduce框架专门在<key，value>对上运行，也就是说，框架将作业的输入视为一组<key，value>对，并生成一组<key，value>对作为输出。 工作，可以想象不同类型。

键和值类必须由框架序列化，因此需要实现Writable接口。 此外，关键类必须实现WritableComparable接口以便于按框架进行排序。

**Mapper**

映射输入的键值对到一组中间键值对。

映射是转换输入记录为中间记录的独立任务。转换的中间记录不需要和输入记录类型一致。输入对可能映射到0或其他输出对。

Hadoop MapReduce框架为job中的每个InputSplit生成一个映射任务。

mapper的实现是通过 [Job.setMapperClass(Class)](https://hadoop.apache.org/docs/current/api/org/apache/hadoop/mapreduce/Job.html)方法传递给job，框架随后在InputSplit中为每个键值对调用 [map(WritableComparable, Writable, Context)](https://hadoop.apache.org/docs/current/api/org/apache/hadoop/mapreduce/Mapper.html)方法。应用可以重载cleanup(Context)方法来执行任意符合要求的清理动作。

Output Pairs，无需与输入对类型一致；可能映射到0或其他输出对；通过context.write(WritableComparable, Writable)方法进行收集。

应用可以使用Counter来报告统计信息。

所有相关中间值都是局部有序的分组，并传递给Reducer来决定最终输出。用户可以通过[Job.setGroupingComparatorClass(Class)](https://hadoop.apache.org/docs/current/api/org/apache/hadoop/mapreduce/Job.html)制定Comparator来控制分组。

Mapper的输出会排序并分割到每个Reducer中，分区数量和job中的reduce任务数量相同。用户可以控制哪个key到哪个Reducer。

用户能通过 [Job.setCombinerClass(Class)](https://hadoop.apache.org/docs/current/api/org/apache/hadoop/mapreduce/Job.html)指定combiner，来执行中间输出的本地聚合，可以减低从mapper到reducer的数据量。

中间排序的输出总是以简单的格式存储（key-len, key, value-len, value），应用可以通过Configuration控制中间输出的压缩方式和压缩编码。

**How Many Maps？**

映射的数量通常由输入的大小决定，即输入文件的块数量。

正确的映射排比等级大概在每个节点10-100映射量，但它已被设置为300个地图，用于cpu-light映射任务。任务的设置需要一定时间，所以最好让映射执行至少一分钟。

因此，如果你期望10TB的数据和128M的块大小，你将得到82000个映射，除非你的设置让其变得更高。

Reducer

Reducer会削减一组中间值，该值会与一组较小的值共享一个key。削减的数量用户可以通过 [Job.setNumReduceTasks(int)](https://hadoop.apache.org/docs/current/api/org/apache/hadoop/mapreduce/Job.html)方法设置。

Reducer的实现通过方法传递，该方法可以重载来初始化自己。随后调用方法。

Reducer有三个主要阶段：shuffle，sort，reduce。

Shuffle

Reducer的输入是mapper排序后的输出。该阶段框架会通过HTTP获取相关的输出分区。

Sort

通过key集合Reducer的输入，shuffle和sort阶段是同时发生的，在获取的同时也在合并。

Secondary Sort

如果要求对中间密钥进行分组的等价规则与在减少之前对密钥进行分组的等效规则不同，则可以通过Job.setSortComparatorClass（Class）指定比较器。 由于Job.setGroupingComparatorClass（Class）可用于控制中间键的分组方式，因此可以结合使用这些键来模拟值的二级排序。

Reduce

在此阶段，为分组输入中的每个<key，（list of values）>对调用reduce（WritableComparable，Iterable <Writable>，Context）方法。

reduce任务的输出通常通过Context.write（WritableComparable，Writable）写入FileSystem。

应用程序可以使用计数器报告其统计信息。

Reducer的输出未排序。

How Many Reduces?

正确的减少次数似乎是0.95或1.75乘以（<节点数> \* <每个节点的最大容器数>）。

使用0.95时，所有缩减都可以立即启动，并在地图完成后开始传输地图输出。 使用1.75，更快的节点将完成第一轮减少并启动第二波减少，从而更好地实现负载平衡。

增加减少的数量会增加框架开销，但会增加负载平衡并降低故障成本。

上面的缩放因子略小于整数，以便在框架中为推测任务和失败任务保留一些减少时隙。