# MapReduce架构和原理

1、MapReduce 定义

　　Hadoop 中的 MapReduce是一个使用简单的软件框架，基于它写出来的应用程序能够运行在由上千个机器组成的大型集群上，并以一种可靠容错并行处理TB级别的数据集

2、MapReduce来源

　　Hadoop MapReduce 源于 Google 在2004年12月份发表的 MapReduce 论文。Hadoop MapReduce 其实就是 Google MapReduce 的一个克隆版本

3、MapReduce特点

　　MapReduce 为什么如此受欢迎？尤其现在互联网+时代，互联网+公司都在使用 MapReduce。MapReduce 之所以如此受欢迎，它主要有以下几个特点：

　　1、MapReduce 易于编程

　　它简单的实现一些接口，就可以完成一个分布式程序，这个分布式程序可以分布到大量廉价的 PC 机器运行。也就是说你写一个分布式程序，跟写一个简单的串行程序是一模一样的。 就是因为这个特点使得 MapReduce 编程变得非常流行。

　　2、良好的扩展性

　　当你的计算资源不能得到满足的时候，你可以通过简单的增加机器来扩展它的计算能力。

　　3、高容错性

　　MapReduce 设计的初衷就是使程序能够部署在廉价的 PC 机器上，这就要求它具有很高的容错性。比如其中一台机器挂了，它可以把上面的计算任务转移到另外一个节点上面上运行，不至于这个任务运行失败，而且这个过程不需要人工参与，而完全是由 Hadoop 内部完成的。

　　4、适合 PB 级以上海量数据的离线处理

　　它适合离线处理而不适合在线处理。比如像毫秒级别的返回一个结果，MapReduce 很难做到

　　MapReduce 虽然具有很多的优势，但是它也有不擅长的地方。这里的不擅长不代表它不能做，而是在有些场景下实现的效果差，并不适合 MapReduce 来处理，主要表现在以下几个方面：

　　1、实时计算

　　MapReduce 无法像 Mysql 一样，在毫秒或者秒级内返回结果。

　　2、流式计算

　　流式计算的输入数据时动态的，而 MapReduce 的输入数据集是静态的，不能动态变化。这是因为 MapReduce 自身的设计特点决定了数据源必须是静态的。

　　3、DAG（有向图）计算

　　多个应用程序存在依赖关系，后一个应用程序的输入为前一个的输出。在这种情况下，MapReduce 并不是不能做，而是使用后，每个MapReduce 作业的输出结果都会写入到磁盘，会造成大量的磁盘IO，导致性能非常的低下

4、MapReduce 实例

　　为了分析 MapReduce 的编程模型，这里我们以 WordCount 为实例。就像 Java、C++等编程语言的入门程序 hello word 一样，WordCount 是 MapReduce 最简单的入门程序。

　　下面我们就来逐步分析

　　1、场景：假如有大量的文件，里面存储的都是单词。

　　　　类似应用场景：WordCount 虽然很简单，但它是很多重要应用的模型。

　　　　1) 搜索引擎中，统计最流行的K个搜索词。

　　　　2) 统计搜索词频率，帮助优化搜索词提示。

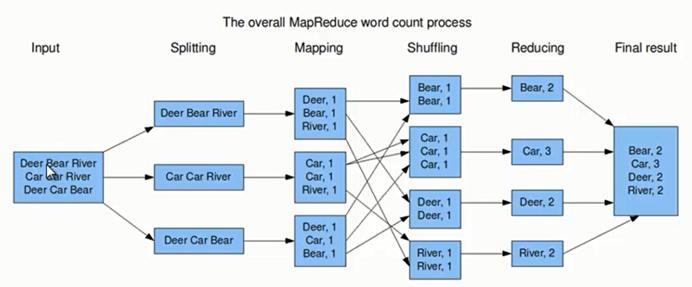
　　2、任务：我们该如何统计每个单词出现的次数？

　　3、将问题规范为：有一批文件（规模为 TB 级或者 PB 级），如何统计这些文件中所有单词出现的次数。

　　4、解决方案：首先，分别统计每个文件中单词出现的次数；然后，累加不同文件中同一个单词出现次数

5、MapReduce编程模型

　　通过上面的分析可知，它其实就是一个典型的 MapReduce 过程。下面我们通过示意图来分析 MapReduce 过程



　　上图的流程大概分为以下几步：

　　1、假设一个文件有三行英文单词作为 MapReduce 的Input（输入），这里经过 Splitting过程把文件分割为3块。分割后的3块数据就可以并行处理，每一块交给一个 map 线程处理。

　　2、每个map线程中，以每个单词为key，以1作为词频数value，然后输出。

　　3、每个map的输出要经过shuffling（混洗），将相同的单词key放在一个桶里面，然后交给reduce处理。

　　4、reduce接受到shuffle后的数据，会将相同的单词进行合并，得到每个单词的词频数，最后将统计好的每个单词的词频数作为输出结果。

　　上述就是 MapReduce 的大致流程，前两步可以看做map阶段，后两步可以看做reduce 阶段。下面我们来看看MapReduce大致实现

　　1、Input：首先MapReduce输入的是一系列key/value对。key表示每行偏移量，value代表每行输入的单词。

　　2、用户提供了map函数和reduce函数的实现

　　　　map(k,v) ——> list(k1,v1)

　　　　reduce(k1,list(v1)) ——>(k2,v2)

　　map函数将每个单词转化为key/value对输出，这里key为每个单词，value为词频1。(k1,v1)是map输出的中间key/value结果对。reduce将相同单词的所有词频进行合并，比如将单词k1，词频为list(v1)，合并为(k2,v2)。reduce 合并完之后，最终输出一系列(k2,v2)键值对

　　下面我们来看一下 MapReduce 的伪代码

　　　　map(key,value):// map 函数，key代表偏移量，value代表每行单词

　　　　for each word w in value:// 循环每行数据，输出每个单词和词频的键值对(w,1)

　　　　emit(w,1)

　　　　reduce(key,values):// reduce 函数，key代表一个单词，value代表这个单词的所有词频数集合

　　　　result=0

　　　　for each count v in values: // 循环词频集合，求出该单词的总词频数，然后输出(key,result)

　　　　　　result+=v

　　　　emit(key,result)

　　讲到这里，我们可以对 MapReduce 做一个总结：MapReduce将作业的整个运行过程分为两个阶段：Map阶段和Reduce阶段。

　　1、Map阶段

　　Map 阶段是由一定数量的Map Task组成。这些Map Task可以同时运行，每个Map Task又是由以下三个部分组成。

　　　　1) 对输入数据进行解析的组件：InputFormat

　　　　因为不同的数据可能存储的数据格式不一样，这就需要有一个InputFormat组件来解析这些数据。默认情况下，它提供了一个TextInputFormat来解释数据。TextInputFormat 就是我们前面提到的文本文件输入格式，它会将文件的每一行解释成(key,value)，key代表每行偏移量，value代表每行数据内容。通常情况我们不需要自定义 InputFormat，因为 MapReduce提供了很多种InputFormat的实现，我们根据不同的数据格式，选择不同的 InputFormat 来解释就可以了。

　　　　2)输入数据处理：Mapper

　　　　这个 Mapper 是必须要实现的，因为根据不同的业务对数据有不同的处理。

　　　　3)数据分组：Partitioner

　　　　Mapper 数据处理之后输出之前，输出key会经过Partitioner分组或者分桶选择不同的reduce。默认的情况下，Partitioner会对map输出的key进行hash取模，比如有6个Reduce Task，它就是模（mod）6，如果key的hash值为0，就选择第0个 Reduce Task，如果key的hash值为1，就选择第一个 Reduce Task。这样不同的map对相同单词key，它的hash值取模是一样的，所以会交给同一个reduce来处理。

　　2、Reduce 阶段

　　Reduce阶段由一定数量的Reduce Task组成。这些Reduce Task可以同时运行，每个 Reduce Task又是由以下四个部分组成。

　　　　1) 数据运程拷贝

　　　　Reduce Task 要运程拷贝每个map处理的结果，从每个map中读取一部分结果。每个 Reduce Task拷贝哪些数据，是由上面 Partitioner 决定的。

　　　　2) 数据按照key排序

　　　　Reduce Task读取完数据后，要按照key进行排序。按照key排序后，相同的key被分到一组，交给同一个Reduce Task处理。

　　　　3) 数据处理：Reducer

　　　　以WordCount为例，相同的单词key分到一组，交个同一个Reducer处理，这样就实现了对每个单词的词频统计。

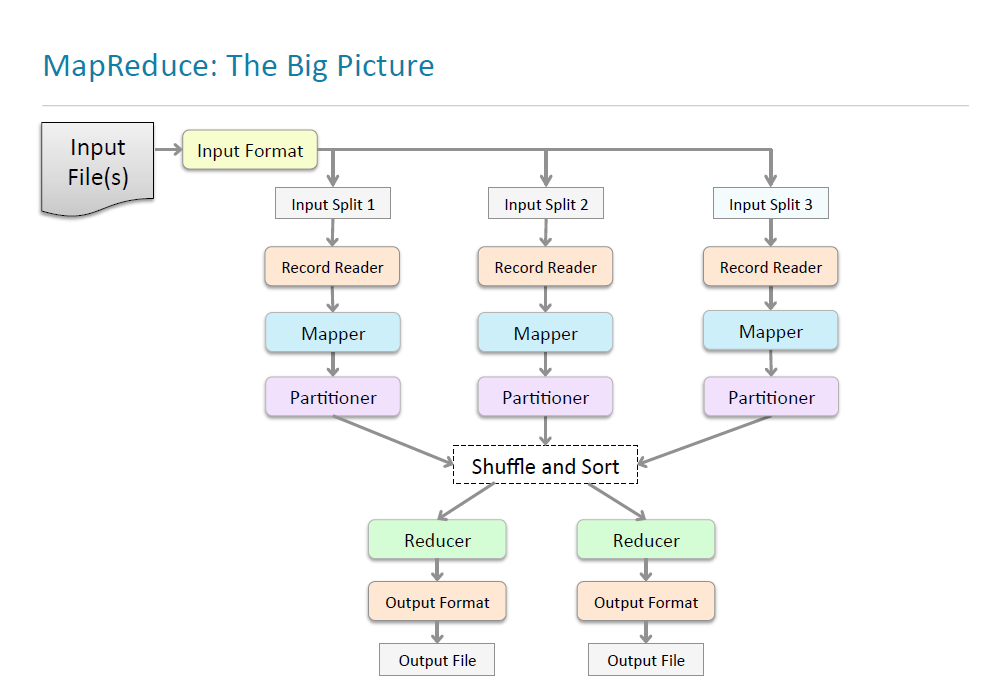
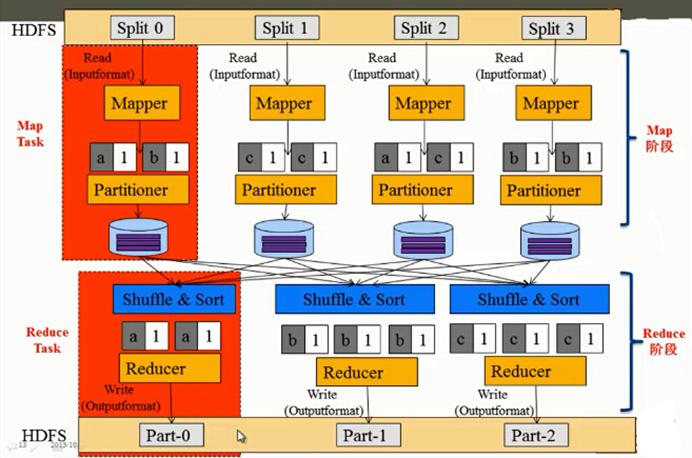
　　　　4) 数据输出格式：OutputFormat

　　　　Reducer统计的结果，将按照OutputFormat格式输出。默认情况下的输出格式为 TextOutputFormat，以WordCount为例，这里的key为单词，value为词频数。

　　　　InputFormat、Mapper、Partitioner、Reducer和OutputFormat 都是用户可以实现的。通常情况下，用户只需要实现 Mapper和Reducer，其他的使用默认实现就可以了。

6、MapReduce内部逻辑

　　下面我们通过MapReduce的内部逻辑，来分析MapReduce的数据处理过程。我们以WordCount为例，来看一下mapreduce 内部逻辑，如下图所示



　　MapReduce 内部逻辑的大致流程主要由以下几步完成

　　1、首先将 HDFS 中的数据以 Split 方式作为 MapReduce 的输入。以前文章提到，HDFS中的数据是以 block存储，这里怎么又变成了以Split 作为输入呢？其实 block 是 HDFS 中的术语，Split 是 MapReduce 中的术语。默认的情况下，一个 Split 可以对应一个 block，当然也可以对应多个block，它们之间的对应关系是由 InputFormat 决定的。默认情况下，使用的是 TextInputFormat，这时一个Split对应一个block。 假设这里有4个block，也就是4个Split，分别为Split0、Split1、Split2和Split3。这时通过 InputFormat 来读每个Split里面的数据，它会把数据解析成一个个的(key,value)，然后交给已经编写好的Mapper 函数来处理。

　　2、每个Mapper 将输入(key,value)数据解析成一个个的单词和词频，比如(a,1)、(b,1)和(c,1)等等。

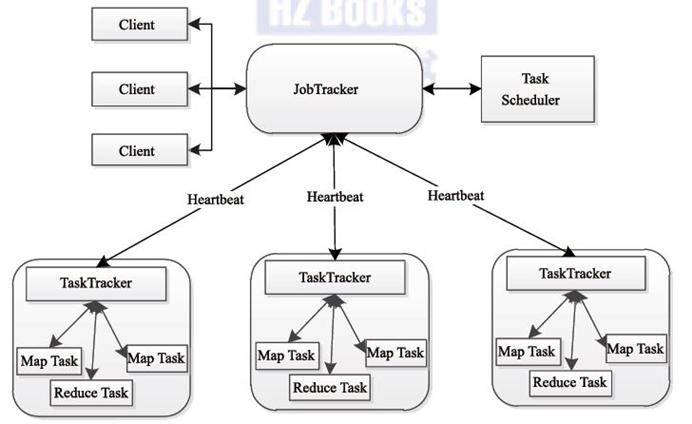
　　3、Mapper解析出的数据，比如(a,1)，经过 Partitioner之后，会知道该选择哪个Reducer来处理。每个 map 阶段后，数据会输出到本地磁盘上。

　　4、在reduce阶段，每个reduce要进行shuffle读取它所对应的数据。当所有数据读取完之后，要经过Sort全排序，排序之后再交给 Reducer 做统计处理。比如，第一个Reducer读取了两个的(a,1)键值对数据，然后进行统计得出结果(a,2)。

　　5、将 Reducer 的处理结果，以OutputFormat数据格式输出到 HDFS 的各个文件路径下。这里的OutputFormat默认为TextOutputFormat，key为单词，value为词频数，key和value之间的分割符为"\tab"。 由上图所示，(a 2)输出到Part-0，(b 3)输出到Part-1，(c 3)输出到Part-2

7、MapReduce架构

　　和HDFS一样，MapReduce也是采用Master/Slave的架构，其架构图如下所示



　　MapReduce包含四个组成部分，分别为Client、JobTracker、TaskTracker和Task，下面我们详细介绍这四个组成部分。

　　1）Client 客户端

　　每一个 Job 都会在用户端通过 Client 类将应用程序以及配置参数 Configuration 打包成 JAR 文件存储在 HDFS，并把路径提交到 JobTracker 的 master 服务，然后由 master 创建每一个 Task（即 MapTask 和 ReduceTask） 将它们分发到各个 TaskTracker 服务中去执行。

　　2）JobTracker

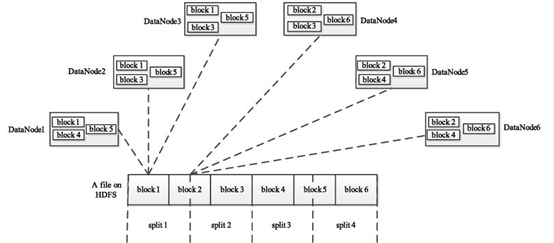
　　JobTracke负责资源监控和作业调度。JobTracker 监控所有TaskTracker 与job的健康状况，一旦发现失败，就将相应的任务转移到其他节点；同时，JobTracker 会跟踪任务的执行进度、资源使用量等信息，并将这些信息告诉任务调度器，而调度器会在资源出现空闲时，选择合适的任务使用这些资源。在Hadoop中，任务调度器是一个可插拔的模块，用户可以根据自己的需要设计相应的调度器。

　　3）TaskTracker

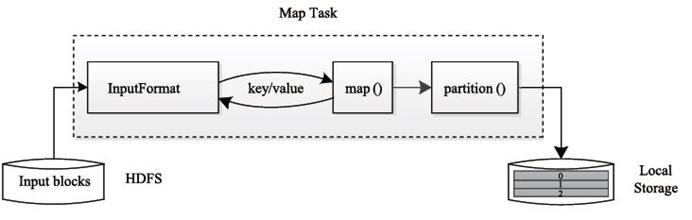
　　TaskTracker 会周期性地通过Heartbeat 将本节点上资源的使用情况和任务的运行进度汇报给JobTracker，同时接收JobTracker 发送过来的命令并执行相应的操作（如启动新任务、杀死任务等）。TaskTracker 使用"slot"等量划分本节点上的资源量。"slot"代表计算资源（CPU、内存等）。一个Task 获取到一个slot 后才有机会运行，而Hadoop 调度器的作用就是将各个TaskTracker 上的空闲slot分配给Task 使用。slot分为Map slot 和Reduce slot 两种，分别供Map Task 和Reduce Task 使用。TaskTracker 通过slot 数目（可配置参数）限定Task 的并发度。

　　4）Task

　　Task 分为Map Task 和Reduce Task 两种，均由TaskTracker 启动。HDFS 以固定大小的block 为基本单位存储数据，而对于MapReduce 而言，其处理单位是split。split 是一个逻辑概念，它只包含一些元数据信息，比如数据起始位置、数据长度、数据所在节点等。它的划分方法完全由用户自己决定。但需要注意的是，split 的多少决定了Map Task 的数目，因为每个split 只会交给一个Map Task 处理。Split 和 Block的关系如下图所示

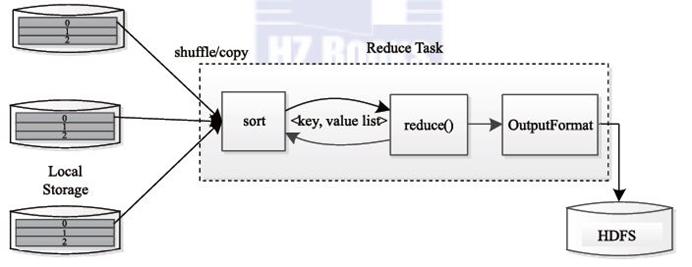


　　Map Task 执行过程如下图 所示：由该图可知，Map Task 先将对应的split 迭代解析成一个个key/value 对，依次调用用户 自定义的map() 函数进行处理，最终将临时结果存放到本地磁盘上, 其中临时数据被分成若干个partition，每个partition 将被一个Reduce Task 处理



　　Reduce Task 执行过程下图所示。该过程分为三个阶段：

* 从远程节点上读取Map Task 中间结果（称为"Shuffle 阶段"）；
* 按照key 对key/value 对进行排序（称为"Sort 阶段"）；
* 依次读取< key, value list>，调用用户自定义的reduce() 函数处理，并将最终结果存到HDFS 上（称为"Reduce 阶段"）



8、MapReduce框架的容错性

　　MapReduce 最大的特点之一就是有很好的容错性，即使你的节点挂掉了1个、2个、3个，都是没有问题的， 它都可以照常来运行，把你的作业或者应用程序运行完成。不会出现某个节点挂了，你的作业就运行失败这种情况。 那么 MapReduce 到底是通过什么样的机制，使它具有这么好的容错性呢？下面我们依次来介绍一下。

　　1、JobTracker

　　很不幸，JobTracker 存在单点故障，一旦出现故障，整个集群就不可用。这个是1.0里面出现的问题，在2.0里面这个问题已经得到了解决。不过大家放心，即使在1.0中，MapReduce也不会经常出现故障。它可能一年也就是出现几次故障，出现故障之后，你重启一下，再把作业重新提交就可以了，它不会像 HDFS 那样出现数据的丢失。 因为 MapReduce 是一个计算框架，计算过程是可以重现的，即使某个服务挂掉了，你重启一下服务，然后把作业重新提交，也是不会影响你的业务的。

　　2、TaskTracker

　　TaskTracker 周期性的向 JobTracker 汇报心跳，如果一定的时间内没有汇报这个心跳，JobTracker 就认为该TaskTracker 挂掉了，它就会把上面所有任务调度到其它TaskTracker（节点）上运行。这样即使某个节点挂了，也不会影响整个集群的运行。

　　3、MapTask和ReduceTask

　　MapTask和ReduceTask 也可能运行挂掉。比如内存超出了或者磁盘挂掉了，这个任务也就挂掉了。这个时候 TaskTracker 就会把每个MapTask和ReduceTask的运行状态回报给 JobTracker，JobTracker 一旦发现某个Task挂掉了，它就会通过调度器把该Task调度到其它节点上。这样的话，即使任务挂掉了，也不会影响应用程序的运行

9、MapReduce资源组织方式

　　MapReduce 计算框架并没有直接调用 CPU和内存等多维度资源，它把多维度资源抽象为 "slot"，用 "slot" 来描述资源的数量。管理员可以在每个节点上单独配置slot个数。slot可以分为map slot和reduce slot。从一定程度上，slot可以看做"任务运行并行度"。如果某个节点配置了5个map slot，那么这个节点最多运行5个Map Task；如果某个节点配置了3个reduce slot，那么该节点最多运行3个Reduce Task。下面我们分别介绍 Map slot和Reduce slot。

　　1、Map slot

* Map slot可用于运行 Map Task的资源，而且只能运行Map Task。
* 每个Map Task通常使用一个map slot。而比如像容量调度器，它可以有比较大的 MapTask。这样的MapTask使用内存比较多，那么它可能使用多个map slot。

　　2、Reduce slot

* Reduce slot 可用于运行ReduceTask，而且只能运行ReduceTask。
* 每个ReduceTask通常使用一个reduce slot。而比如像容量调度器，它可以有比较大的 ReduceTask。这样的ReduceTask使用内存比较多，那么它可能使用多个reduce slot