# 07 | MapReduce（二）：不怕失败的计算框架

2021-10-04 徐文浩

《大数据经典论文解读》 课程介绍 >

讲述：徐文浩
时长 19:35 大小 17.95M

你好，我是徐文浩。

通过上节课的学习，现在你已经知道 MapReduce 的编程模型是怎么回事儿了。对于开发者来说，你只需要写一个 Map 函数和一个 Reduce 函数，就能完成数据处理过程。具体这些任务用了多少服务器，遇到了失败是怎么解决的，你并不需要关心。

不过，要想学习如何搭建和改进分布式系统，了解 MapReduce 的底层原理必不可少。今天，我们就一起来看看 MapReduce 的框架干了什么。MapReduce 这个“保姆”，为什么可以让你不需要处理复杂的分布式架构的问题。

## MapReduce 框架的三个挑战

要想让写 Map 和 Reduce 函数的人不需要关心“分布式”的存在，那么 MapReduce 框架本身就需要解决好三个很重要的问题：

第一个，自然是如何做好各个服务器节点之间的“**协同**”，以及解决出现各种软硬件问题后的**“容错”**这两部分的设计。

第二个，是上一讲我们没怎么关心的**性能**问题。和我们在 GFS 论文里面讲过的一样，MapReduce 框架一样非常容易遇到网络性能瓶颈。尽量充分利用 MapReduce 集群的计算能力，并让整个集群的性能可以随硬件的增加接近于线性增长，可以说是非常大的一个挑战。

最后一个，还是要回到**易用性**。Map 函数和 Reduce 函数最终还是运行在多个不同的机器上的，并且在 Map 和 Reduce 函数中还会遇到各种千奇百怪的数据。当我们的程序在遭遇到奇怪的数据出错的时候，我们需要有办法来进行 debug。

而谷歌在论文里面，也通过第三部分的“MapReduce 的实现”，以及第四部分的“MapReduce 的完善”，很好地回答了怎么解决这三个问题。下面，我们就来具体看看，论文里是怎么讲的。

## MapReduce 的协同

一个 MapReduce 的集群，通常就是之前的分布式存储系统 GFS 的集群。在这个集群里，本身会有一个**调度系统**（Scheduler）。当我们要运行一个 MapReduce 任务的时候，其实就是把整个 MapReduce 的任务提交给这个调度系统，让这个调度系统来分配和安排 Map 函数和 Reduce 函数，以及后面会提到的 master 在不同的硬件上运行。

在 MapReduce 任务提交了之后，整个 MapReduce 任务就会按照这样的顺序来执行。

第一步，你写好的 MapReduce 程序，已经指定了输入路径。所以 MapReduce 会先找到 GFS 上的对应路径，然后把对应路径下的所有数据进行**分片（Split）**。每个分片的大小通常是 64MB，这个尺寸也是 GFS 里面一个块（Block）的大小。接着，MapReduce 会在整个集群上，启动很多个 MapReduce 程序的复刻（fork）进程。

第二步，在这些进程中，有一个和其他不同的特殊进程，就是一个 master 进程，剩下的都是 worker 进程。然后，我们会有 M 个 map 的任务（Task）以及 R 个 reduce 的任务，分配给这些 worker 进程去进行处理。**这里的 master 进程，是负责找到空闲的**

**（idle）worker 进程**，然后再把 map 任务或者 reduce 任务，分配给 worker 进程去处理。

这里你需要注意一点，并不是每一个 map 和 reduce 任务，都会单独建立一个新的 worker 进程来执行。而是 master 进程会把 map 和 reduce 任务分配给有限的 worker，因为一个 worker 通常可以顺序地执行多个 map 和 reduce 的任务。

第三步，被分配到 map 任务的 worker 会读取某一个分片，分片里的数据就像上一讲所说的，变成一个个 key-value 对喂给了 map 任务，然后等 Map 函数计算完后，会生成的新的 key-value 对缓冲在内存里。

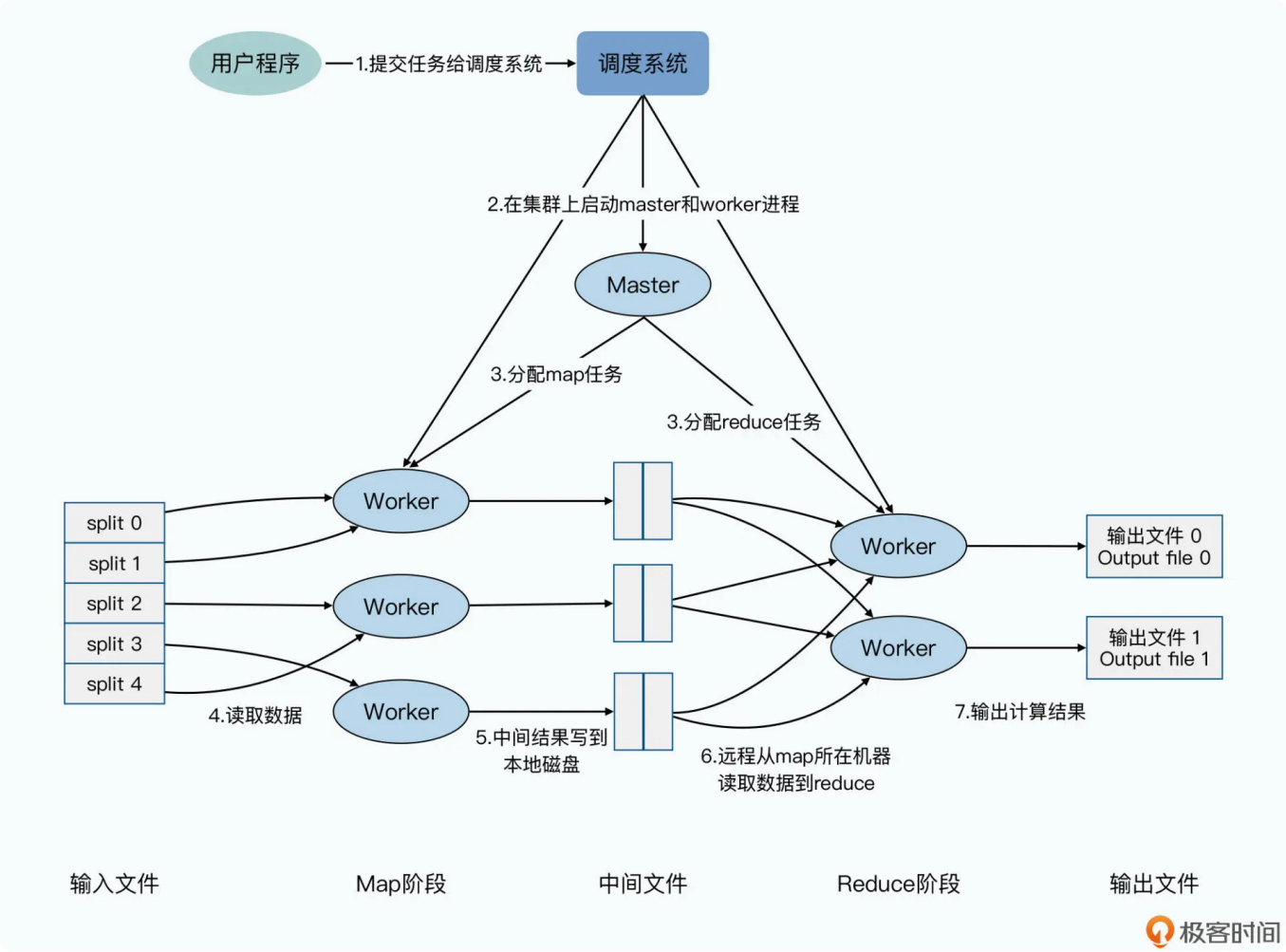
第四步，这些缓冲了的 key-value 对，会定期地写到 map 任务所在机器的本地硬盘上。并且按照一个**分区函数（partitioning function）**，把输出的数据分成 R 个不同的区域。而这些本地文件的位置，会被 worker 传回给到 master 节点，再由 master 节点将这些地址转发给 reduce 任务所在的 worker 那里。

第五步，运行 reduce 任务的 worker，在收到 master 的通知之后，会通过 RPC（远程过程调用）来从 map 任务所在机器的本地磁盘上，抓取数据。当 reduce 任务的 worker 获取到所有的中间文件之后，它就会将中间文件根据 Key 进行排序。这样，所有相同 Key 的 Value 的数据会被放到一起，也就是完成了我们上一讲所说的**混洗（Shuffle）**的过程。

第六步，reduce 会对排序后的数据执行实际的 Reduce 函数，并把 reduce 的结果输出到当前这个 reduce 分片的最终输出文件里。

第七步，当所有的 map 任务和 reduce 任务执行完成之后，master 会唤醒启动 MapReduce 任务的用户程序，然后回到用户程序里，往下执行 MapReduce 任务提交之后的代码逻辑。

其实，以上整个 MapReduce 的执行过程，还是一个典型的 Master-Slave 的分布式系统。map 和 reduce 所在的 worker 之间并不会直接通信，它们都只和 master 通信。另外，像是 map 的输出数据在哪里这样的信息，也是告诉 master，让 master 转达给 reduce 所在的 worker。reduce 从 map 里获取数据，也是直接拿到数据所在的地址去抓取，而不是让 reduce 通过 RPC，调用 map 所在的 worker 去获取数据。

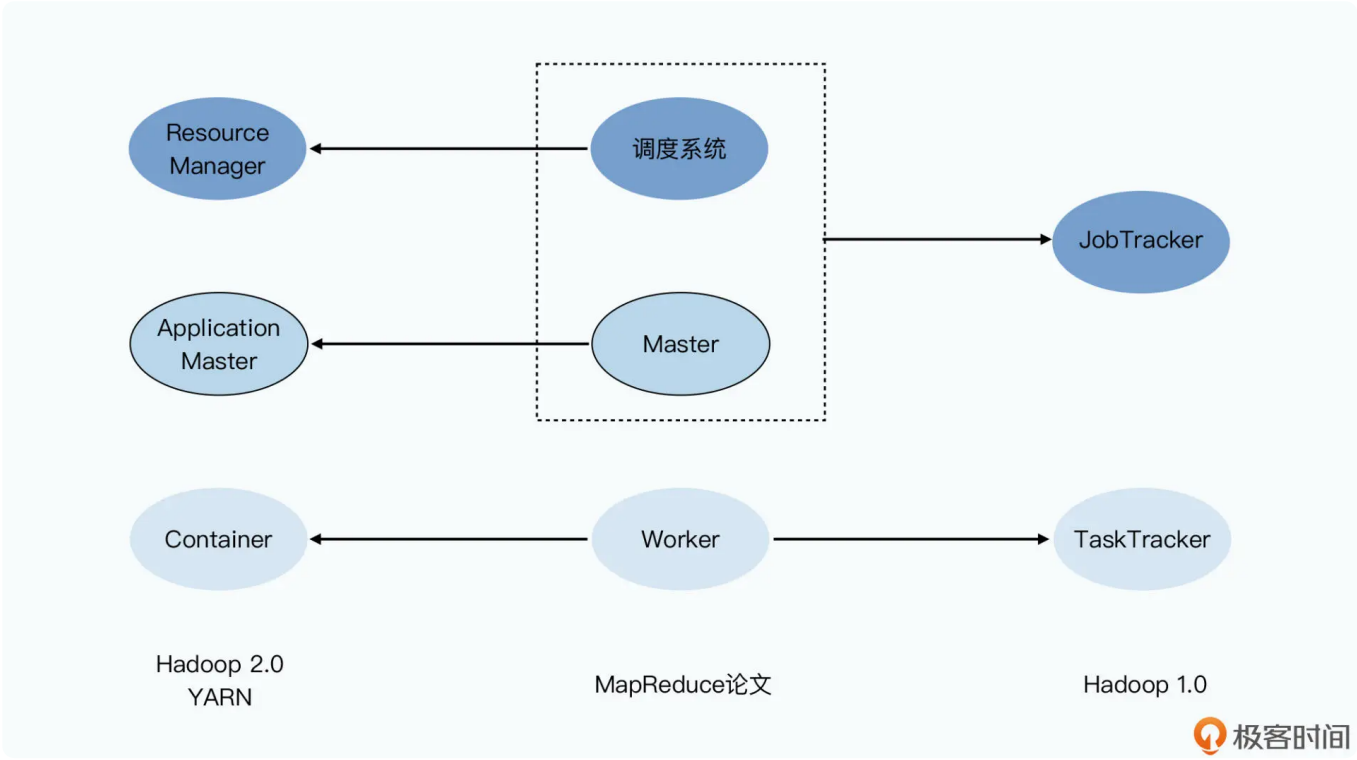



我稍稍修改了一下论文中的原图，加上了容易被我们忽视的调度系统的存在

如果你熟悉 MapReduce 的开源实现 Hadoop 的话，你会发现 Hadoop 1.0 的实现，其实和 MapReduce 的论文不太一样。在 Hadoop 里，每一个 MapReduce 的任务并没有一个独立的 master 进程，而是直接让调度系统承担了所有的 worker 的 master 的角色，这就是 Hadoop 1.0 里的 **JobTracker**。

在 Hadoop 1.0 里，MapReduce 论文里面的 worker 就是 **TaskTracker**，用来执行 map 和 reduce 的任务。而分配任务，以及和 TaskTracker 沟通任务的执行情况，都由单一的 JobTracker 来负责。

这个设计，也导致了只要服务器数量一多，JobTracker 的负载就会很重。所以早年间，单个 Hadoop 集群能够承载的服务器上限，被卡在了 4000 台。而且 JobTracker 也成为了整个 Hadoop 系统很脆弱的“单点”。




在开源的Hadoop环境下理解MapReduce论文，可以先简单按照映射管理来看

所以之后在 Hadoop 2.0，Hadoop 社区把 JobTracker 的角色，拆分成了进行任务调度的 **Resource Mananger**，以及监控单个 MapReduce 任务执行的 **Application Master**，回到了和 MapReduce 论文相同的架构。

而在 2015 年，谷歌发布了 Borg 这个集群管理系统的论文的时候，大家发现谷歌早在 2003~2004 年，就已经有了独立的集群管理系统 Borg，也就是 MapReduce 里面所提到的**调度系统**。在后面的资源调度模块中，我们也会专门解读 Borg 这个调度系统的论文，以及被认为是 Borg 后继者和开源实现 Kubernetes 的论文。

## MapReduce 的容错（Fault Tolerance）

MapReduce 的容错机制非常简单，可以简单地用两个关键词来描述，就是**重新运行**和**写 Checkpoints**。

### worker 节点的失效（Master Failure）

对于 Worker 节点的失效，MapReduce 框架解决问题的方式非常简单。就是换一台服务器重新运行这个 Worker 节点被分配到的所有任务。master 节点会定时地去 ping 每一个 worker 节点，一旦 worker 节点没有响应，我们就会认为这个节点失效了。

于是，我们会重新在另一台服务器上，启动一个 worker 进程，并且在新的 worker 进程所在的节点上，重新运行所有失效节点上被分配到的任务。而无论失效节点上，之前的 map 和 reduce 任务是否执行成功，这些任务都会重新运行。因为在节点 ping 不通的情况下，我们很难保障它的本地硬盘还能正常访问。

## master 节点的失效（Worker Failure）

对于 master 节点的失效，事实上谷歌已经告诉了我们，他们就任由 master 节点失败了，也就是整个 MapReduce 任务失败了。那么，对于开发者来说，解决这个问题的办法也很简单，就是再次提交一下任务去重试。

因为 master 进程在整个任务中只有一个，它会失效的可能性很小。而 MapReduce 的任务也是一个用户离线数据处理的任务，并不是一个实时在线的服务，失败重来通常也没有什么影响，只是晚一点拿到数据结果罢了。

虽然在论文发表的时候，谷歌并没有实现对于 master 的失效自动恢复机制，但他们也给出了一个很简单的解决方案，那就是让 master 定时把它里面存放的信息，作为一个个的 Checkpoint 写入到硬盘中去。

那么我们动一下脑筋，我们可以把这个 Checkpoint 直接写到 GFS 里，然后让调度系统监控 master。这样一旦 master 失效，我们就可以启动一个新的 master，来读取 Checkpoints 数据，然后就可以恢复任务的继续执行了，而不需要重新运行整个任务。

## 对错误数据视而不见

worker 和 master 的节点失效，以及对应的恢复机制，通常都是来自于硬件问题。但是在海量数据处理的情况下，比如在 TB 乃至 PB 级别的数据下，我们还会经常遇到“**脏数据**”的问题。

这些数据，可能是日志采集的时候就出错了，也可能是一个非常罕见的**边界情况**（edge-case），我们的 Map 和 Reduce 函数正好处理不了。甚至有可能，只是简单的硬盘硬件的问题带来的错误数据。

那么，对于这些异常数据，我们固然可以不断 debug，一一修正。但是这么做，大多数时候都是划不来的，你很可能为了一条数据记录，由于 Map 函数处理不了，你就要重新扫描

几 TB 的数据。

所以，**MapReduce 不仅为节点故障提供了容错机制，对于这些极少数的数据异常带来的问题，也提供了一个容错机制。**MapReduce 会记录 Map 或者 Reduce 函数，运行出错的具体数据的行号，如果同样行号的数据执行重试还是出错，它就会跳过这一行的数据。如果这样的数据行数在总体数据中的比例很小，那么整个 MapReduce 程序会忽视这些错误，仍然执行完成。毕竟，一个 URL 被访问了 1 万次还是 9999 次，对于搜素引擎的排序结果不会有什么影响。

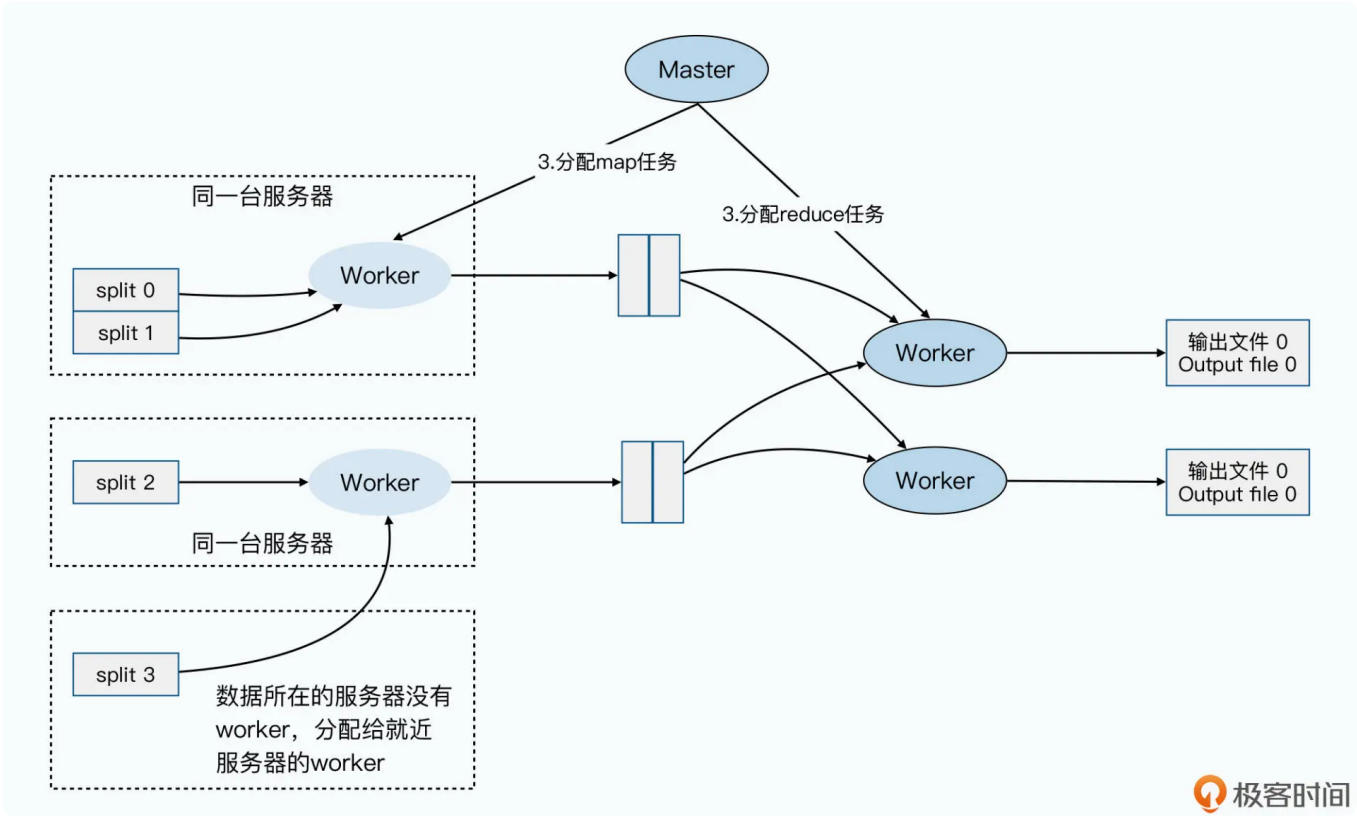
## MapReduce 的性能优化

聊完了 MapReduce 的容错处理，我们接着一起来看看 MapReduce 的性能问题。我们在前面说过，其实 MapReduce 的集群就是 GFS 的集群。所以 MapReduce 集群里的硬件配置，和 GFS 的硬件配置差不多，最容易遇到的性能瓶颈，也是 100MB 或者 1GB 的网络带宽。

### 把程序搬到数据那儿去

既然网络带宽是瓶颈，那么优化的办法自然就是**尽可能减少需要通过网络传输的数据**。

在 MapReduce 这个框架下，就是在分配 map 任务的时候，根据需要读取的数据在哪里进行分配。通过前面 GFS 论文的学习，我们可以知道，GFS 是知道每一个 Block 的数据是在哪台服务器上的。而 MapReduce，会找到同样服务器上的 worker，来分配对应的 map 任务。如果那台服务器上没有，那么它就会找离这台服务器最近的、有 worker 的服务器，来分配对应的任务。你可以参考下面给出的示意图：




除此之外，由于 MapReduce 程序的代码往往很小，可能只有几百 KB 或者几 MB，但是每个 map 需要读取的一个分片的数据是 64MB 大小。这样，我们通过把要执行的 MapReduce 程序，复制到数据所在的服务器上，就不用多花那 10 倍乃至 100 倍的网络传输量了。

这就好像你想要研究金字塔，最好的办法不是把金字塔搬到你家来，而是你买张机票飞过去。这里的金字塔就是要处理的数据，而你，就是那个分配过去的 MapReduce 程序。

## 通过 Combiner 减少网络数据传输

除了 Map 函数需要读取输入的分片数据之外，Reduce 所在的 worker 去抓取中间数据，一样也需要通过网络。那么要在这里减少网络传输，最简单的办法，就是**尽可能让中间数据的数据量小一些**。

自然，在 MapReduce 的框架里，也不会放过这一点。MapReduce 允许开发者自己定义一个 **Combiner 函数**。这个 Combiner 函数，会对在同一个服务器上所有 map 输出的结果运行一次，然后进行数据合并。

比如，在上一讲我留下的思考题里，如果你想要统计每个域名的访问次数，那么 Map 函数的输出结果，就会是一个域名 + 一次访问计数的 1。

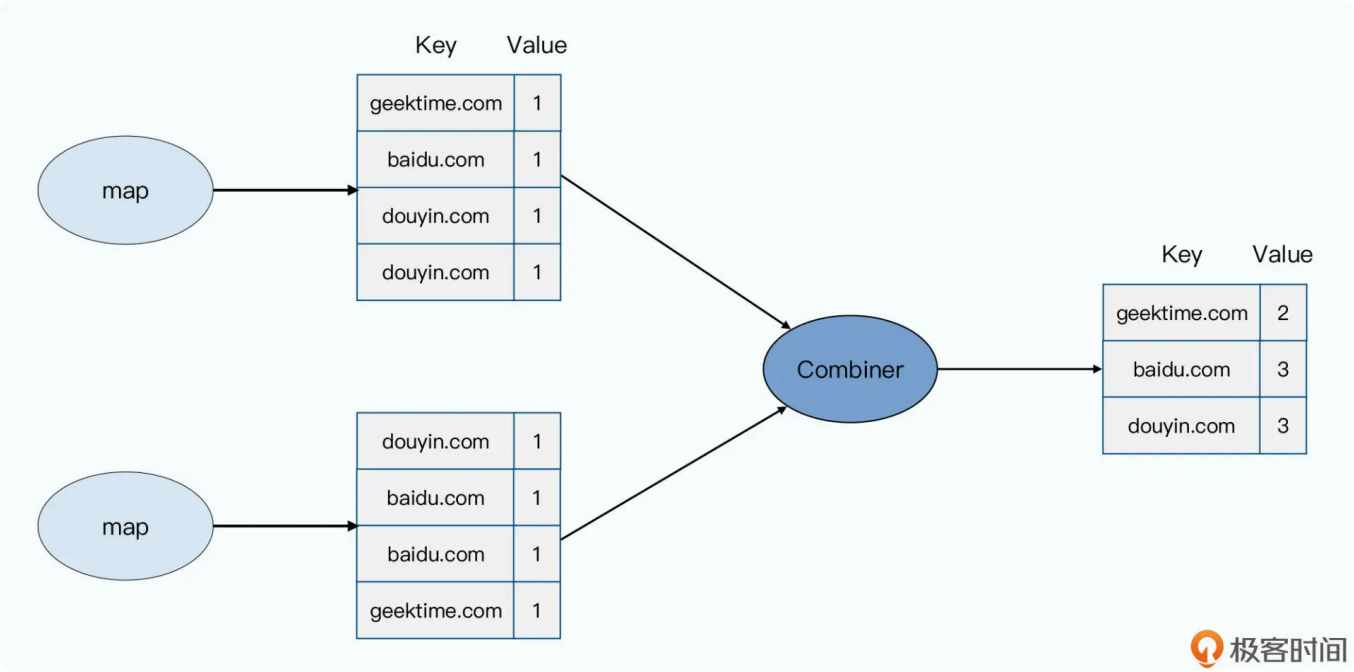
而对于用户会高频访问的网站，在 map 输出的中间结果里就会有很多条记录，比如用户访问了 baidu.com、douyin.com 这样的域名就会有大量的记录。这些记录的 Key 就是对应的 baidu.com、douyin.com 的域名，而 value 都是 1。

既然只是对访问次数计数，我们自然就可以**通过一个 Combiner，把 1 万条相同域名的访问记录做个化简**。把它们变成 Key 还是域名，Value 就是有多少次访问的数值这样的记录就好了。而这样一化简，reduce 所在的 worker 需要抓取的数据，就从 1 万条变成了 1 条。

实际上，不仅是同一个 Map 函数的输出可以合并，同一台服务器上多个 Map 的输出，我们都可以合并。反正它们都在一台机器上，合并只需要本地的硬盘读写和 CPU，并不需要我们最紧缺的网络资源。

我就以域名的访问次数为例，它的数据分布一定有很强的头部效应，少量 20% 的域名可能占了 80% 的访问记录。这样一合并，我们要传输的数据至少可以减少 60%。如果考虑一台 16 核的服务器，有 16 个 map 的 worker 运行，应该还能再减少 80% 以上。这样，通过一个中间的 Combiner，我们要传输的数据一下子就下降了两个数量级，大大缓解了网络传输的压力。

你可以参考下面给出的示意图：




## MapReduce 的 debug 信息

好，性能优化完了，最后我们来看一下 MapReduce 对于开发者的易用性。

虽然我们一直说，我们希望 MapReduce 让开发者意识不到分布式的存在。但是归根到底，**map 和 reduce 的任务都是在分布式集群上运行的，这个就给我们对程序 debug 带来了很大的挑战。**无论是通过 debugger 做单步调试，还是打印出日志来看程序执行的情况，都不太可行。所以，MapReduce 也为开发者贴心地提供了三个办法来解决这一点。

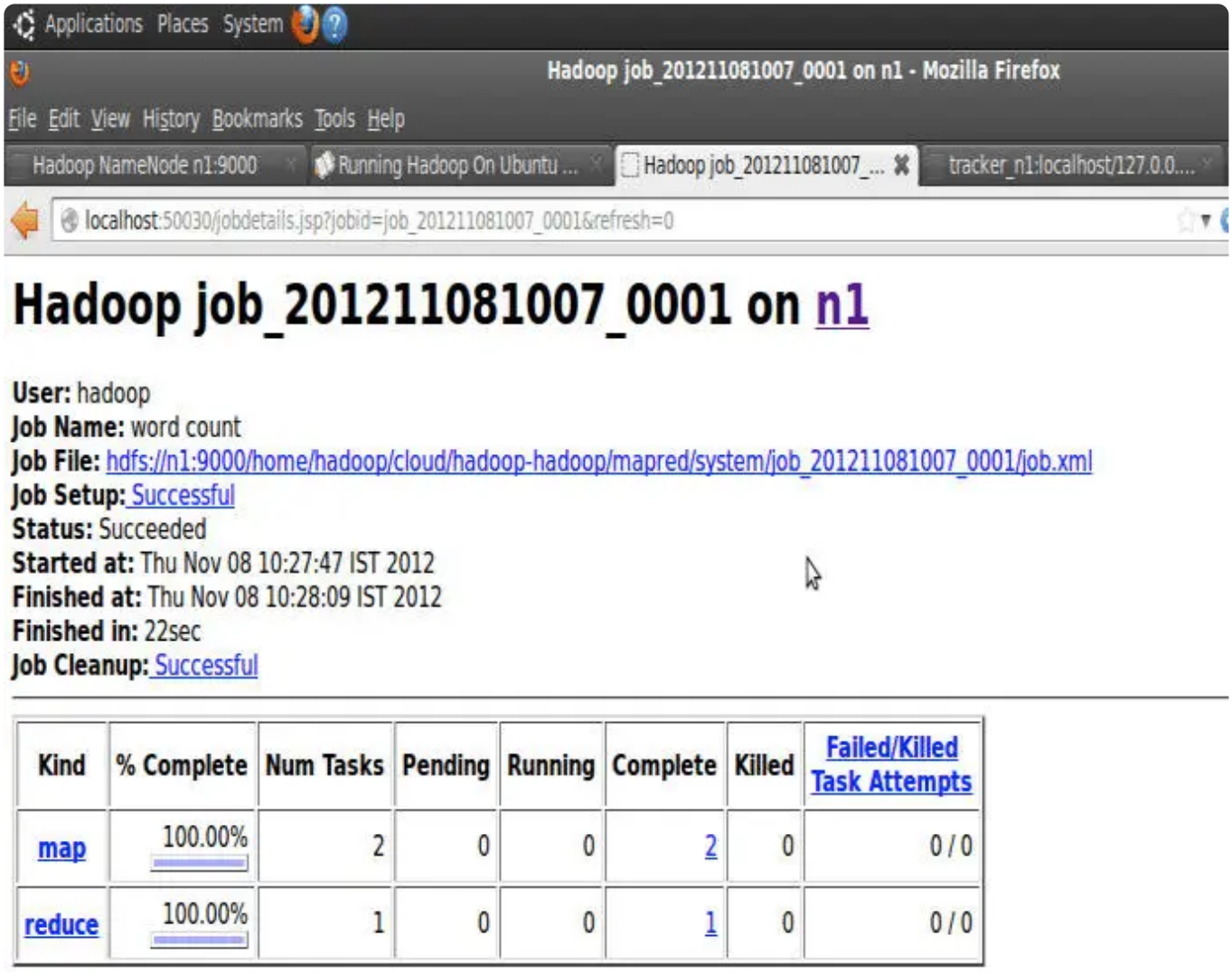
第一个，是**提供一个单机运行的 MapReduce 的库**，这个库在接收到 MapReduce 任务之后，会在本地执行完成 map 和 reduce 的任务。这样，你就可以通过拿一点小数据，在本地调试你的 MapReduce 任务了，无论是 debugger 还是打日志，都行得通。

第二个，是在 **master 里面内嵌了一个 HTTP 服务器**，然后把 master 的各种状态展示出来给开发者看到。这样一来，你就可以看到有多少个任务执行完了，有多少任务还在执行过程中，它处理了多少输入数据，有多少中间数据，有多少输出的结果数据，以及任务完成的百分比等等。同样的，里面还有每一个任务的日志信息。

另外通过这个 HTTP 服务器，你还可以看到具体是哪一个 worker 里的任务失败了，对应的错误日志是什么。这样，你就可以快速在线上定位你的程序出了什么错，是在哪台服务器上。我在《深入浅出计算机组成原理》的课程中，就举过一个我自己遇到的“单比特翻转”导致的问题。而对于产生这个问题根本原因的推断，就是通过查看 MapReduce 的任务日志，发现数据出错的 worker 总是在固定的一台服务器上。

最后一个，是 **MapReduce 框架里提供了一个计数器（counter）的机制**。作为开发者，你可以自己定义几个计数器，然后在 Map 和 Reduce 的函数里去调用这个计数器进行自增。所有 map 和 reduce 的计数器都会汇总到 master 节点上，通过上面的 HTTP 服务器里展现出来。

比如，你就可以利用这个计数器，去统计有多少输入日志的格式和预期的不一样。如果比例太高，那么多半你的程序就有 Bug，没有兼容所有合法的日志。下图展示的就是在 Hadoop 里，通过 JobTracker 查看 Task 的执行情况，以及对应每个 Task 的日志：




Hadoop里，你可以通过JobTracker看到任务的执行情况和对应每个Task的日志

> 图片来源

这些机制看起来好像并不起眼，似乎和分布式计算框架的名头关系不大。但也正是这些易用的小功能，就让开发者在开发分布式数据处理任务的时候效率大增，不需要吭哧吭哧一台台服务器去翻日志来排查问题，可谓是功莫大焉。

## 小结

MapReduce 的论文到这里就讲解完了。和 GFS 一样，MapReduce 的实现是比较简单的，就是**一个典型的单 master 多 worker 组成的主从架构**。在分布式系统容错上，MapReduce 也采取了简单的**重新运行、再来一次**的方案。对于 master 这个单点可能出现的故障，谷歌在最早的实现里，根本就没有考虑失效恢复，而是选择了任由 master 失败，让开发人员重新提交任务重试的办法。

还有一点也和 GFS 一样，MapReduce 论文发表时的硬件，用的往往是 100MB 或者 1GB 的网络带宽。所以 MapReduce 框架对于这一点，就做了不少性能优化动作。通过尽量让各个 worker 从本地硬盘读取数据，以及通过 Combiner 合并本地 Map 输出的数据，来尽可能减少数据在网络上的传输。

而为了方便开发人员去 debug 程序，以及监控程序的执行，MapReduce 框架通过 master 内嵌的 Web 服务器，展示了所有 worker 的运行情况和日志。你还可以通过自定义的计数器，统计更多你觉得有价值的信息。

当然，MapReduce 里还有备用任务（Backup Tasks）、自定义的 Partitioner 等更多的细节值得你去探索。这些就留给你去仔细研读论文，好好琢磨了。

## 遗憾与缺陷

尽管 MapReduce 框架已经作出了很多努力，但是今天来看，整个计算框架的缺陷还是不少的。在我看来，主要的缺陷有两个：

第一个是**还没有 100% 做到让用户意识不到“分布式”的存在**，无论是 Combiner 还是 Partitioner，都是让开发者意识到，它面对的还是分布式的数据和分布式的程序。

第二个是**性能仍然不太理想**，这体现在两个方面，一个是每个任务都有比较大的 overhead，都需要预先把程序复制到各个 worker 节点，然后启动进程；另一个是所有的中间数据都要读写多次硬盘。map 的输出结果要写到硬盘上，reduce 抓取数据排序合并之后，也要先写到本地硬盘上再进行读取，所以快不起来。

不过，随着时间的变迁，会有更多新一代的系统，像是 Dremel 和 Spark 逐步取代 MapReduce，让我们能更容易地写出分布式数据处理程序，处理起数据也比原始的 MapReduce 快上不少。

## 推荐阅读

对于分布式系统，我们总是希望增加机器就能够带来同比例的性能提升，但是这一点其实很难做到。

Storm 的作者南森·马茨（Nathan Marz）在 2010 年就发表过一个很有意思的博文，告诉大家为什么优化 MapReduce 任务 30% 的运行时间，就会减少 80% 任务实际消耗的时

间。这篇文章应该更加有助于你理解为什么我们说 MapReduce 的遗憾与缺陷中提到的额外开销（overhead）问题。我把这篇文章的链接放在这里，推荐你去阅读学习一下。

## 思考题

在用 MapReduce 处理数据的时候，因为数据不平衡，可能会使得 MapReduce 的任务运行得很慢。MapReduce 论文里面给出的解决方法，是通过开发人员自己去实现一个分区函数。

不过，一旦需要开发人员自己思考如何分片，其实已经让我们要意识到这个“分布式”本身的存在了。那么，如果让你在 MapReduce 框架层面解决好这一个问题，你觉得有什么好办法吗？



上一篇 06 | MapReduce（一）：源起Unix的设计思想

下一篇 08 | Bigtable（一）：错失百亿的Friendster

## 精选留言 (8)



**qinsi**

2021-10-04

坐时光机快进的话，现代的大数据框架大都提供了类似SQL的接口，于是任务的实现和优化就类似数据库中查询计划的生成和优化，对于使用者透明了

展开

1

**在路上**

2021-10-06

徐老师好，MapReduce的第一个问题后来通过SQL得到解决，编程界面更友好，第二个问题通过内存+硬盘混合存储得到了解决，内存保存中间数据更快，硬盘保存中间数据更稳定，中间数据丢失可以根据依赖的数据和逻辑重新生成。

...

展开

**峰**

2021-10-06

mapper程序完成后，master搜集了所有mapper输出的信息，简单性的角度，mapper阶段可统计中间key出现的次数，对倾斜的key，由mapper搜集后，做一次再分区。

**Monroe He**

2021-10-05

思考题：
步骤1 论文里说 combiner 与 reduce 的不同只在于 reduce 将结果写入到文件系统中，而 combiner 将结果传递给 reduce, 那是不是可以不用写 combiner 函数而直接在 map 之后使用 reduce 函数做一次局部 reduce, 然后再将结果collect到 reduce中做全局 reduce，以减少数据量，防止数据倾斜。...

展开

**泊浮目**

2021-10-05

如果让你在 MapReduce 框架层面解决好这一个问题，你觉得有什么好办法吗？——现在的做法都是SQL做声明，底层用CBO做优化。

展开

**Lebron**

2021-10-04

思考题个人解答：针对数据倾斜问题，我个人的想法是可以让MapReduce程序在Map之后，程序再将Reduce的work进程分配到倾斜数据量大的服务器或者其相近的服务器，这样在Reduce的时候就可以尽量不占用网络传输。从而不仅提高效率，也可以让“开发者”意识不到“分布式”本身的存在。

展开

**Lebron**

2021-10-04

徐老师，您在文章中提到："除此之外，由于 MapReduce 程序的代码往往很小，通过把要执行的 MapReduce 程序，复制到数据所在的服务器上，就不用多花那 10 倍乃至 100 倍的网络传输量了。"这句话是在什么场景下会复制Map？如果数据所在的服务器上没有worker，是不是就不会复制过去了？以及您在worker节点的失效（Master Failure）和mster节点的失效（Worker Failure）这里写反了，应该是worker节点的失效（Worker Failure）...

展开

1

**那时刻**

2021-10-04

我的想法是通过增加一层Partition层处理，来解决用 MapReduce 处理数据时候数据不平衡问题。Partition层把输出的数据分成 R 个不同的区域，R的值可以依据数据量的大小来确定，默认是1，也就是本地。如果数据大于一个block，按照 数据量/block 来确定R的值。

展开