



下载APP



21 | 分而治之：MapReduce如何解决大规模分布式计算问题

2022-02-08 黄清昊

《算法实战高手课》

课程介绍 >



讲述：黄清昊

时长 13:40 大小 12.52M



你好，我是微扰君。

从今天开始，我们就真正开始学习算法在工业界应用了。和前面的章节不同，分布式系统篇的很多算法，一般都是由工程师们提出来的，为了解决一些大规模网络应用中的实际问题，比如为了解决海量网页排名而发明的 pagerank 算法、为了解决分布式系统中共识问题的 Raft 算法、常用的负载均衡算法一致性哈希等等。

领资料

因为都是在实际的工程场景下被发明出来的，这些算法，在现在的互联网架构中也经常看到它们的身影，所以学习这些算法以及其背后解决的问题对我们的实际工作是有很大好处的。



话不多说，我们开始今天的学习——谷歌提出的 MapReduce 算法，知名的开源项目 Hadoop 其实就是对 MapReduce 的工业级实现之一。

为什么发明 MapReduce 算法

想要掌握一个解决实际生产环境中问题的算法或者框架，我们当然应该先来了解一下相关算法的诞生背景。

MapReduce 算法，作为谷歌知名的三驾马车之一，是早期谷歌对大规模分布式计算的最佳实践。他们当时（2004 年）发表了相关的论文，很清晰地描述了提出这个算法的目的。

当时的谷歌已经有很大的业务量了，每天都需要处理海量的数据，也有许多不同的业务场景，所以工程师们实现了数以百计的数据处理程序，用来实现网页抓取、日志汇总分析、计算倒排索引等任务。

这些任务大部分本身倒也不是很复杂，但因为需要面对巨大的数据量，单机的程序显然没有办法应对谷歌的数据规模，这些程序只能是分布式的运行。

我们知道，在分布式环境上运行的程序都会比较复杂，需要考虑各种问题，比如不同环境下可能出现的异常、数据如何分发到各个机器上、计算完成之后又如何汇总等等。

要是每个业务方都针对这些雷同的问题，各自实现一遍处理这些问题的逻辑，显然是非常低效的。所以为了解决这个问题，谷歌的工程师提出了一种新的、通用的、抽象模型 MapReduce，让业务开发人员不再需要关心并行计算、数据冗余、负载均衡等和分布式系统本身相关的细节了。


Map 和 Reduce

那为什么起名叫 MapReduce 呢？其实 map 和 reduce，在一些函数式编程语言中，是十分常用的概念，比如在 Lisp 里 map 和 reduce 都是作为原语存在的，历史非常悠久。谷歌的分布式计算框架也只是借鉴了其中的思想。

我们就以 js 为例子，先来看看在一般编程语言中的 map 和 reduce 都是什么样的操作。

假设一个 number 数组中，我们希望统计出数值大于 5 的那些数向上取整的和。这个问题很简单，常规的写法自然是遍历整个数组，写一个 if-else 判断出大于 5 的数，然后用一个变量做累计求和。但是这个写法引入了状态，在循环里你既需要关心 filter 的逻辑，又要关心累计求和的逻辑，不够清晰。

而比较函数式的写法是这样的：

 复制代码

```
1 function(arr) {  
2   arr  
3   .filter(a => a > 5)  
4   .map(a => Math.ceil(a))  
5   .reduce((acc, i) => acc + i);  
6 }
```

通过 filter \ map \ reduce 等原语，我们把控制逻辑和计算逻辑分离地非常清楚。

这虽然是一个单机且简单的程序，但是大多数的运算和业务逻辑，其实都是可以通过这样简单的 map 和 reduce 函数来实现的。因为**一个很复杂的计算，也无非就是对某些数据据进行一系列规则的变换，转化成另一些数据。**

如果我们把输入数据表示成一个 key\value 对的集合，输出数据也表示成一个 key\value 对的集合。那么我们只需要通过两种操作就可以完成绝大部分转换。

一是对输入的每个 (key, value) 进行**某种变换**，得到和输入集合规模一样的新集合，但每个值都按照指定的规则进行变换。这个新集合，既可以作为最终的输出结果，也可以作为中间的结果供后续转换操作使用。这个就是 map。

二是对 (key, value) 进行一些**合并的计算**。通常来说就是把某个 key 的不同 value，按照某种规则合并起来，从而得到一个比输入规模更小的 (key, value) 集合。这样的操作就是 reduce。

以刚才统计和的程序为例：

1. 向上取整，需要通过 map 来实现（每个值都按规则变换）；

2. 过滤 5 以上的数字是通过 filter 来实现的，这本质上也是一种 reduce 的操作，输入一个集合，合并的时候返回一个列表，但 reduce 计算的时候，只有符合 >5 条件的元素才会被加入累计的值里；
3. 最后的求和，也是通过 reduce 实现。传入的函数就是一个普通的加法运算，对应的 reduce 操作就会自动对整个集合进行求和了。

除了这个例子，还有很多适用于互联网应用的场景，谷歌的同学当时在论文里就给出了一些例子。比如：

分布式 Grep 程序，可以用于对大量数据的模式匹配，比如查询日志中某些模式出现的数量。

统计 URL 访问频数，把所有的访问记录用 map 函数处理成 (url, 1)，再用 reduce 函数对 url 相同的记录，累计计数。

网络连接逆向，把 (source, target) map 为 (target, source)，再用 reduce 把 target 相同的记录中的 source，合并成列表，得到 (target, list(source))。

倒排索引，把文本分词，得到 (word, document) 的集合，并对同样的词进行 reduce 操作，得到 (word, list(document))，这样就可以用来给搜索引擎加速查询。

这些例子，本身都不是特别复杂。如果我们就用单机系统去实现，相信你用传统的循环控制、开全局变量的方式，去实现这些逻辑都是没有问题的。

当然如果用 map\reduce 这样的原语去实现，你的代码逻辑会更清晰，我个人也更推荐你这么写。而且现在各大语言也都开始引入函数式编程的特性，比如 Java 8 的 stream 就是一种对函数式编程能力的部分释放，如果感兴趣你可以系统学习一下相关 API 的使用和背后函数式编程的思想。

分布式实现

但是，谷歌的 MapReduce 当然不只是一种编程思想，而是一个真正意义上的分布式计算的框架和系统。毕竟对于谷歌来说，那些问题虽然简单，但所需要应对的数据量是单机远远存储不下，也计算不了的。

所以我们需要把整个 map 和 reduce 的过程搬运到一个分布式的系统中来实现。所用的机器也都是普通的商用机器，不一定非常稳定，集群规模到达一定程度的时候，机器或者网络出现异常是在所难免的。这些问题自然也都是谷歌需要考虑的。

那么，MapReduce 到底是如何运作的呢？

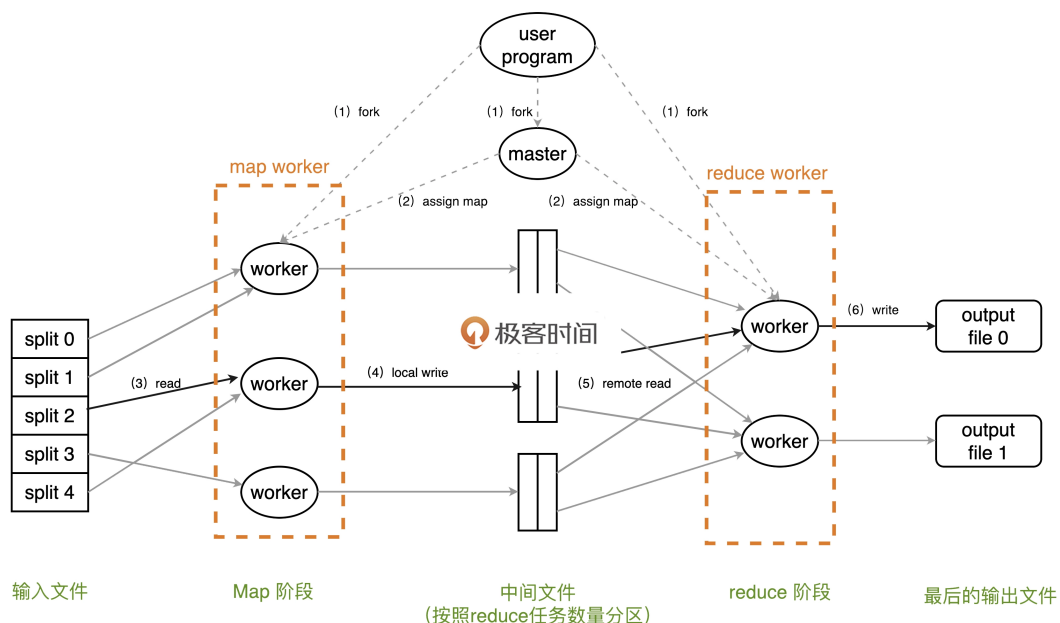
想要任务在 MapReduce 机器上顺利执行，大体上来说就是要做到**把数据分区，交给不同的机器执行，在要汇总的时候也需要有办法进行数据的汇总，并且要有一定应对故障的能力。**

当然了，整个 MapReduce 系统是要有一套全局共享的存储系统的，这就是谷歌鼎鼎大名的三驾马车中的另一架 GFS (Google File System) 的作用，MapReduce 也是建立在这个存储系统之上的，感兴趣的同学可以自行查阅相关资料了解。

我们一起来看当用户发起 MapReduce 任务时，会发生什么。

执行过程

任务发起后，程序首先会把输入文件分成 M 个数据段，每个数据段大小可控，通常为 16MB；然后，整个集群就会快速地分发需要执行的计算任务给到各个节点，让这些节点可以进行计算任务。



在这里有一个很常见的设计，我们会让众多进程中的一个成为 **master 进程**，由它来进行任务的调度，其他进程都是 **worker 进程**，进行实际的计算任务。master 会把 M 个 map 任务和 R 个 reduce 任务分配给空闲的进程。

对于 map 任务，worker 在读取输入数据之后，根据任务内容进行相应的 map 计算，由 map 函数输出中间的结果缓存在内存中；然后 worker 会通过分区函数，把中间结果定期落盘，分离在 r 个区域；这些区域的信息会传递给 master，以待后续 reduce 使用。

对于 reduce 任务，worker 会收到 master 传来的中间结果的位置信息，通过 RPC 读取相应节点中间结果。由于 reduce 是按照不同的 key 进行聚合操作，所以读取完数据之后会做排序，再按照 reduce 函数进行结果的计算。这里计算出的结果同样会被分区追加到对应的分区文件里。

当 map 和 reduce 程序全部执行完成，用户程序会收到通知，读取最终的计算结果。

容错

整个执行过程是比较复杂的，在分布式系统下，一个工业级的应用，必须要考虑容错问题。

一是因为和超级计算机不同，互联网应用通常使用比较廉价的机器，然后通过大规模的部署来提高计算能力和稳定性；二是我们的服务也需要长时间在线，不能每次出现故障都由管理员手动恢复。所以容错灾备是互联网应用的基础能力，MapReduce 需要很好地处理机器的故障。

在 MapReduce 场景下，故障主要包括 worker 故障和 master 故障。

worker 故障

worker 是实际负责计算的节点。

在计算场景下，如果任意一个 worker 正在处理的任务失败，不进行任何处理，整个计算任务就会因为部分失败而全部失败，所有 worker 全部完成任务才能得到完整的数据处理结果。

所幸，我们有一个全局的控制节点 master，它能很好发现 worker 的故障，并适时地进行任务的重新调度。具体做法也是在网络中很常见的**定时 ping 操作**。master 会周期性地给 worker 发送 ping 请求，如果 worker 正常就会回复，所以如果 master 一段时间没有收到回复，会把这个 worker 标记为失效 worker，相关的任务也会被设置为空闲状态，进而分配给其他空闲的 worker。

如果 Map 的任务异常，由于 Map 任务的中间结果都存储在本地节点中，当节点异常时，我们就需要重新执行该节点上已经完成的 Map 任务；而 reduce worker，因为完成任务之后会直接输出到全局文件系统中，不需要重新执行已经完成的 reduce 的任务，只需要重新执行这次失败的任务即可。

引入 master 节点，对 worker 节点进行监控和重新调度的机制，在分布式系统中是非常常见的，你可以好好掌握。

master 故障

现在 master 能很好地帮助我们解决 worker 的故障了，那如果 master 出现了故障又该怎么办呢？

一种最简单的方式是**周期性的把 master 相关的状态信息保存到磁盘中，形成一个个检查点**。如果 master 任务失败了，我们就从最近的一个检查点恢复当时的执行状态，全部重新执行。

另外一些比较常用的手段，比如可以对 master 引入 backup 节点，如果一个节点挂了，我们马上把备份的节点拿来当新的主节点使用，这样恢复的速度就会快很多。但谷歌的工程师当时的选择更简单一些，就是直接终止这个程序，让用户感知到并重新提交任务，其实不能说是最好的解决方案。

总结

MapReduce 在整个互联网世界取得了巨大的成功，第一次将大规模的分布式计算用简洁易用的 API 抽象出来，封装了并行处理、数据分发、容错、负载均衡等繁琐的技术细节，把面对海量数据的应用开发人员解放出来，解决了许多不同类型的问题。

可以说后面的所有分布式计算框架比如 Hadoop、Spark、Flink 等等都是建立在 Google MapReduce 工作的基础上的。

系统设计，非常重要的一点就是对容灾能力的支持，主要就要分为故障检测和故障恢复两个步骤。

对于检测来说，引入一个控制节点对其他节点进行监控是一个非常有效的手段，通过定时的心跳，控制节点就很容易发现其他节点的异常；而故障恢复的一个有效手段就是定时设置 checkpoint，定期记录下运行正确时系统的状态，这样异常发生的时候就可以快速恢复，重新执行需要的计算。

课后作业

MapReduce 毕竟是一个非常古老的系统了，学习它能带给你很多启发，但你也要了解其中设计不足的地方。说一说你觉得 MapReduce 哪些机制设计的不好，为什么后面又产生了许多新的分布式计算框架呢？

欢迎在留言区写下你的思考，如果觉得这篇文章对你有帮助的话，也欢迎转发给你的朋友。我们下节课见～

分享给需要的人，Ta购买本课程，你将得 20 元

 生成海报并分享

 赞 1  提建议

© 版权归极客邦科技所有，未经许可不得传播售卖。页面已增加防盗追踪，如有侵权极客邦将依法追究其法律责任。

上一篇 即学即练 | 计算机网络篇：复习卡 & 算法题特训

下一篇 22 | PageRank：谷歌是如何计算网页排名的

学习推荐

JVM + NIO + Spring

各大厂面试题及知识点详解

限时免费



精选留言 (2)

写留言



peter

2022-02-08

请教老师三个问题：

Q1：数组求和中的“状态”是指什么？

“number 数组中，我们希望统计出数值大于 5 的那些数向上取整的和”，循环遍历的写法，“但是这个写法引入了状态”。

这里的“状态”是指什么？是指“求和”吗？...

展开



Paul Shan

2022-02-08

个人觉得MapReduce框架中master节点任务过重，成为关键节点，一旦出错，恢复比较麻烦，备份的话，要同步的信息也很多，容易成为瓶颈。

