大数据算法

易 珂
香港科技大学

关键词：大数据算法 数据移动

“大数据”（big data）一词最近两年炒得火热。但在我看来，所谓“大数据”和业已存在的“海量数据”（massive data）或者“超大规模数据”（very large data）从技术层面讲并无本质上的区别，都是指大量的用传统方法无法处理的数据。而“大数据”之所以能够改头换面再次掀起热潮，其深层原因是如今大数据已不是科学研究的专属品，随着互联网、物联网、云计算和社交媒体的蓬勃发展，它已经扩散到各个行业乃至个人，以致又重新引起学术界和工业界的广泛关注。

本刊亦在2012年第6期、第9期连续刊登了大数据专题，对大数据环境下诸多方面的问题进行了探讨。然而，算法作为计算机科学的基石，尚未进行过专门的阐述。任何一个计算问题经过分析和建模，几乎都规约为算法问题，在大数据的环境下同样如此。本文尝试在大数据的背景下，浅析一些算法设计所要做出的改变以及面临的挑战。

数据移动成为计算瓶颈

本文认为数据移动（即通信开销）是大数据计算问题的主要瓶颈，从而也是算法设计中的主要优化目标。大数据时代下，计算逐渐从CPU密集型转化为数据密集型。CPU密集型的计算任务数据量不大，但对CPU速度要求高，如各种组合优化问题、线性规划、数值计算等等，对这些问题的算法复杂度往往只要求是多项式级即可，对它们的理论研究关注的也是多项式级和非多项式级的区分。而数据密集型的计算任务面临超大的数据规模，算法的复杂度要求必须为线性或近线性（near-linear），甚至于亚线性（sub-linear）。这意味着数据集里的每一条数据只通过CPU一次或者几次。这样，把数据从它们的存储区域移动到CPU再移走的时间就远远超过了它们在CPU停留的时间，使得CPU不再成为计算的瓶颈。这种计算模式的转化对于硬件发展既是好消息也是坏消息：好消息是我们不必再费尽心力维持CPU速度发展的摩尔定律，坏消息是我们必须提高存储系统、通信系统的性能，解决新的瓶颈问题。同时对于算法设计而言，我们也要将重心从传统的时间复杂度移到通信复杂度上，努力降低算法的数据移动开销。数据移动的代价可以说是根本性的：信息存储需要空间，信息移动速度不会超过光速，计算任务又依赖于数据间的交互，所以不管硬件架构如何发展，数据移动的开销总是无法避免。

算法设计的发展在过去二十多年中也经历了从CPU密集型到数据密集型的转化，诞生了很多面向大数据的新型计算模型。这些计算模型往往都忽略了CPU开销，用一些新瓶颈来刻画算法复杂度。本文在余下的篇幅里将对一些主要的算法模型作简单介绍，从中可以看出，它们用来刻画复杂度的都是某种形式的数据移动的开销。当然，没有一个模型是完美的，正如乔治·布克斯（George Box）的名言：“所有的模型都是错的，但有些还有点儿用”（all models are wrong, but some are useful）。在实际中影响计算性能的因素很多，一个理论模型是否有用取决于它是否抓住了最重要的因素而不失简

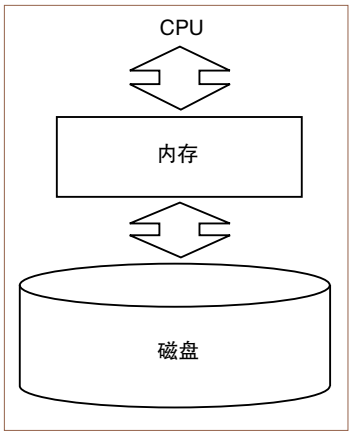

图1 外存模型

洁，能否对现实具有预测能力，为算法设计提供依据。

外存模型 外存（external memory）算法用于高效处理存放在磁盘里的数据。一些经典的外存算法，如外排序（external sorting）和外哈希（external hashing）早在20世纪70年代就已提出。外存模型作为一个计算模型被提出，以及对这些外存算法的理论分析始于20世纪80年代末。

如图1所示，数据一开始都存放在外存（磁盘）中。内存有限，无法容纳所有的数据，而CPU只能访问内存里的数据。这样，为进行计算，数据必须从磁盘移动到内存，不需要的时候还要移回磁盘，释放内存空间。数据在磁盘上以“块”为单位存储，每次读写数据也都要以块为单位。这一点和实际的磁盘系统是相吻合的：磁盘被分为很多同心圆，称为“道”，每条道又被分为很多“扇区”。读写一次磁盘，首先要将读写磁头移至所要读写的道，再等待磁盘旋转到指定的扇区，而实际的读写时间相对来说就短很多了。所以每次读写磁盘时，操作系统都自动地读写整个扇区，用来平均抵消前面磁头移动和磁盘旋转的时间。外存模型下设计的算法往往忽略CPU时间，只计算在磁盘上读写的块的数目，也称作读写（input/output，I/O）数。可以看到，这个读写数正是数据在磁盘和内存间移动的开销！

外存算法在大规模数据处理上取得了巨大成功，特别是在数据库系统中，几乎所有的基本数据库操作用的都是外存算法，如外排序、外哈希和B-树等。当然，随着内存技术不断提高，容量不断增大，很多数据库可以完全安置到内存中（in-memory databases）。尽管如此，外存算法在相当长的一段时间里还是非常重要，原因有二：一是内存是非保持性存储介质，一旦断电数据即丢失，这违背了数据库ACID（atomic（原子性）、consistency（一致性）、isolation（隔离性）和durability（持久性））四大基本原则中的持久性（durability）；二是数据库中建立了大量索引结构（index），用来支持对数据的快速检索和查找。这些结构不可能长时间保持在内存中，所以还需要外存算法对其进行检索和修改等操作。近些年来，闪存（flash）技术突飞猛进，已经出现了以闪存取代磁盘的数据库系统，大大提高了数据读写的效率。然而，尽管闪存不像磁盘那样移动磁头和旋转，但它仍是以“块”为单位来进行数据读写，所以以前的外存算法仍然适用。不过，闪存也有其自身的一些和磁盘不同的读写特性，如读和写开销的不对称性等，需要我们在算法设计时，特别是在进行工程实现时加以注意。

数据流模型 数据流（data streams）模型在外存模型的基础上，完全去掉了磁盘，目标是在内存里解决问题，即便是数据量远远大于内存容量。从某种意义上讲，数据流模型是把数据移动的开销降至0。确切地说，在此模型下，数据以流的形式通过CPU一次，而算法必须用非常有限的内存空间解决问题。这对很多算法是个很大的限制。事实上，有很多问题可以严格证明在此模型下不可解。不过研究人员同时也发现，虽然这些问题在数据流模型下不能精确解决，但是如果允许一定的可控误差，还是存在很有效的算法的。误差对于大数据问题来说是完全可以接受的，一来是数据庞大，结果过度精确没有必要，二是原始数据本身可能就存在误差。

数据流模型由阿隆（Alon）等人于1996年正式提出。不过后来人们发现一些经典算法实际上在更早就已经提出，常举的例子就是米斯拉（Misra）和格里斯（Gries）的频繁项（frequent items）算法。经过十多年的研究，我们现在对很多问题已经有了高效的数据流算法，并且证明了它们的最优性。这些算法广泛应用于各种网络数据流监控系统中。另外，学术界和工业界也研制出很多通用的数据流管理系统（也称复杂事件处理系统（complex

event processing）），如美国的威斯康星大学的NiagaraCQ、斯坦福大学的Stream以及微软的StreamInsight等。

在我看来，数据流模型的成功不仅由于对数据流本身的处理，还有其对大数据研究的意义。这主要体现在它的数据摘要（data summarization）技术，包括数据勾勒（data sketches）、直方图（histograms）、压缩感知（compressed sensing）、核心集（core sets）和采样（sampling）技术等等。大数据虽然量大，但质量低，从浩瀚的数据中摘出最有用、最具概况的信息对于大数据的有效处理意义重大。在数据流模型下，由于内存受限，使得这些技术非常有用并得到广泛研究。但它们的用途不仅仅局限在数据流的处理上，数据摘要对于在并行/分布式环境下减少数据移动的开销（本文的宗旨）也是一个非常有效的工具，后面还会详述。

PRAM模型 上面讲到的两个计算模型考虑的是单机情况下大数据的处理环境。而如果要真正地具有规模，并行或者分布式结构几乎是必须的。各种各样的并行计算模型及算法曾在20世纪80年代风靡一时，一度成为计算机理论科学的主流，产生了一套完整的理论系统和很多优美的算法。然而，进入90年代，对并行算法研究的热潮渐渐退去，甚至被人遗忘。当然，其中原因众多，如模型脱离实际，算法编程实现困难等。在我看来，最根本的一个原因是，它没有将问题的核心放到数据移动上来。我们以最具代表性的并行模型PRAM（parallel random access machine，随机存取并行机器）为例，此模型下有一个单一平坦的可寻址共享内存和n个处理器。算法由一系列“并行步”组成，在每一个并行步里，所有的处理器同步进行1次内存读、1次计算和1次内存写操作。言下之意，就是内存里的任何数据可以在单位时间里移动到任何一个处理器，而与距离无关，这显然和现实大相径庭。事实上，即便在一个多核CPU上，每个核都有自己独享的寄存器和高速缓存，访问共享内存需要通过各种上锁机制解决冲突，费时费力。而松散组织的分布式系统（如大型机群）则根本没有共享内存，要访问远程的存储空间，需通过网络传输来完成，由此会产生瓶颈。

尽管PRAM在实践上没有成功，但其带来的丰富的理论成果和算法设计思想对并行程序设计还是产生了影响。现实中，由于各种并行/分布式系统的体系结构、参数不尽相同，所以科研人员和工程师们往往是从已有的PRAM算法出发，经过适当地修改、调整，以找到适应当前系统的最佳策略。同时，由于并行/分布式程序的编写和调试即便在今天仍是十分困难的，工具缺乏，因此我们往往会牺牲一些算法效率，以追求实现上的便捷。

MapReduce模型 编写正确高效的大规模并行/分布式程序是计算机工程领域的一大难题，所以谷歌于2004年公布的MapReduce编程模型在工业界乃至学术界产生了极大的影响，以至于“谈大数据必谈MapReduce”。

前面提到，MapReduce处理的数据是键-值对（key-value pairs）。这些键-值数据一开始就分布式地存放在一个由成千上万个节点组成的大型机群中，每个节点只存放一部分数据。如图2所示，一个MapReduce任务分为三个步骤：在Map阶段，每个节点调用一个程序员编写的Map函数，作用于每一个在此节点存放的键-值对[1]。Map函数的输出同样是一些键-值对，这些中间结果进入Shuffle阶段。这个阶段是由系统自动完成的，程序员无须也无法

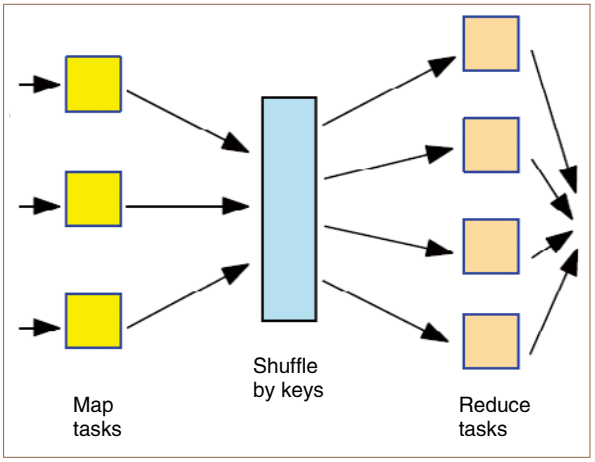

图2 一个MapReduce任务分解过程

CCF CNCC 2012科技成果展获奖名单

最佳组织展示奖 大连大学

最佳展示奖 阿里云计算有限公司 中国传媒大学计算机学院

控制[2]。Shuffle阶段会把所有中间结果里键相同的所有键-值对通过网络传递给同一个目标节点。在最后的Reduce阶段，每个节点会对所有键相同的键-值对调用另一个程序员编写的Reduce函数，输出最终结果。当然，Reduce函数也可以选择再次输出一些键-值对，从而可以启动新一轮的MapReduce过程，如此往复。

实验人员发现，一个MapReduce任务的瓶颈往往是中间的Shuffle阶段，特别是当系统中节点数量多、并发任务数多的时候。其原因在于：Map和Reduce阶段各个节点都是独立工作，有很高的并行性；Shuffle阶段各节点则需要交互，共享网络带宽，而网络带宽恰恰是大型机群和数据中心中最宝贵的资源。这再次印证了本文最初的观点：大数据算法的瓶颈是数据移动！为此，在设计MapReduce算法时，就要尽可能减少中间结果，哪怕在Map和Reduce阶段每个节点多做一些工作。这里就正好用到前面提到的数据流模型下发展起来的数据摘要技术，它们对于减少MapReduce任务里的中间结果十分有效。

MapReduce成功的最大因素是它简单的编程模型。程序员只要设计Map和Reduce两个函数，剩下的工作，如节点调度、负载均衡、容错处理和故障恢复都由系统自动完成，设计出的程序也有很高的可扩展性。这对需要编写大规模并行/分布式程序的程序员来说是一大福音。可是，编程模型的简单也大大限制了程序员的自由度，很多较复杂的任务难以完成，成为MapReduce的最大弱点之一。此外，MapReduce还存在如下问题：（1）启动开销大，对简单任务也要经历Map-Shuffle-Reduce三个过程，无法做到实时响应；（2）只能处理静态数据，对变化快的数据（如数据流）无能为力；（3）MapReduce的系统实现至今仍为谷歌机密，而开源的版本Hadoop效率低下（据谷歌内部人士透露，5年前的谷歌MapReduce版本也比当前的Hadoop快一个数量级；现在谷歌的版本到底有多快，是否有新的功能，不得而知）。突破上述限制已成为当前学术界和工业界的研究热点：例如，很多人尝试结合关系型数据库和MapReduce（如HadoopDB等），突破MapReduce编程模型简单的局限；谷歌于2010年公布的Dremel系统可用于大规模数据分析和查询的实时化，弥补了MapReduce启动开销大的问题，但其实现细节仍未发布。另外，很多数据流管理系统开始尝试进行分布式扩展，我本人也在从事分布式数据流上的算法研究，希望能够突破MapReduce只能处理静态数据的限制。

大数据是个大题目，除了算法外，还包括数据挖掘、可视化、异构数据的模型和处理、数据存储、数据质量、数据安全和隐私等诸多方面。有兴趣的读者可参阅本刊今年第6、9期有关“大数据”的专题文章。■

易 珂

香港科技大学副教授。主要研究方向为海量数据算法、数据库技术等。

yike@ust.hk

[1] 在严格的MapReduce模型中，作用于每个键-值对的Map函数是独立调用的；但在扩展的模型中，同一节点的所有Map函数调用是可以通过本节点的内存进行通信的。

[2] 程序员可以通过定制的哈希函数使Shuffle做得快一些，但仍无法操纵结果。