分布式系统论文读后感

高浩雨

19301113

论文的核心在讨论分布式系统，与我们上课讲到的很多内容都有所呼应，比如CAP原则。热力学第二定律说明了永动机是不可能存在的，不要去妄图设计永动机。与之类似，CAP 理论的意义就在于明确提出了不要去妄图设计一种对CAP 三大属性都完全拥有的完美系统，因为这种系统在理论上就已经被证明不存在。分布式从根本出发其实就是并发，并发，其实就是通过设计可以同时处理多个请求。我们来看一张图了解一下，通过金额来看看数据。也就是说，在同一时刻不是一个人在下单，可能时10个，100个，1000个，10000个人同时下单。而在这种情况下，服务器正常运行，能够在将处理结果毫无错误的正常执行，这就时处理并发能力产生的果效。

对于mapreduce来说，本质是一个软件框架，基于该框架能够容易地编写应用程序，这些应用程序能够运行在由上千个商用机器组成的大集群上，并以一种可靠的，具有容错能力的方式并行地处理上TB级别的海量数据集。它主要用来处理海量的数据，在此之前也有许多程序员实现过成百上千的专用的计算方法，这些计算方法来处理大量原始数据，但由于输入的数据量巨大，因此要想在可接受的时间内完成运算，只有将这些计算分布在成百上千的主机上。如何处理并行计算、如何分发数据、如何处理错误？所有这些问题综合在一起，需要大量的代码处理，因此也使得原本简单的运算变得难以处理。MapReduce既是一种编程模型，也是一种与之关联的、用于处理和产生大数据集的实现。用户要特化一个map程序去处理key/value对，并产生中间key/value对的集合，以及一个reduce程序去合并有着相同key的所有中间key/value对。本文指出，许多实际的任务都可以用这种模型来表示。MapReduce的思想就是“分而治之”。Mapper负责分”，即把复杂的任务分解为若干个“简单的任务”来处理。“简单的任务”包含三层含义：一是数据或计算的规模相对原任务要大大缩小；二是就近计算原则，即任务会分配到存放着所需数据的节点上进行计算；三是这些小任务可以并行计算，彼此间几乎没有依赖关系。

接着它开始讲MapReduce的实现，虽然我是软件专业的，但我还是看不懂它到底是怎样实现的。我大致的理解是MapReduce先把输进来的数据分为若干个片段，然后创建大量副本，然后将任务分配给各个设备去完成，完成任务后再唤醒用户程序，来输出。这也非常符合我们对大数据操作的第一反应，大数据既然操作不了，那就切分为小数据来进行操作。

其实这种处理方法很像上学期算法课学到的分治算法将一个复杂的问题，分为若干个简单的子问题进行解决。然后，对子问题的结果进行合并，得到原有问题的解。 将一个大任务分成晓得子任务，并完成子任务的计算，这个过程叫Map，将中间结果合并成最终结果，这个过程叫Reduce。

这允许不具备任何并行和分布式系统经验的程序员也能轻松地利用一个大型分布式系统的资源。作者的MapReduce实现运行在一个大型PC机集群上，且具有很好的扩展性：一个典型的MapReduce计算要在数千台机器上处理若干TB的数据。程序员可以很轻松的使用这一系统：目前已经实现的MapReduce程序数以百计，每天有上千个MapReduce作业运行在Google的集群上。

这篇文章指出，这种抽象受到了Lisp中的Map和Reduce以及其他函数式编程语言的启发。作者认为，大多数相关的计算都涉及对输入中的每个记录应用映射操作，以计算一组中间键/值对，然后对共享相同键的所有值应用Reduce操作。用户可以计算map和Reduce函数，系统在大型集群机器上自动并行计算。程序员会发MapReduce系统很容易使用。从2004年到2008年，谷歌内部实现了超过10,000个不同的MapReduce程序。在谷歌集群上，平均每天执行100,000个MapReduce任务，每天处理超过20 petabytes的数据。它的应用非常广泛。作者认为，这项工作的最大贡献是为大规模商业PC集群(如GFS)提供了一个简明而高效的并行计算接口。在计算任务中，输入是键/值对，输出是键/值对。用户可以使用MapReduce的两个功能:Map和Reduce。映射由用户编写，它接受输入对并生成一组中间键/值。MapReduce库将与同一个中间键(如键L)相关联的所有中间值组合起来，并将它们传递给Reduce函数。reduce函数也是由用户编写的，它接受一个中间键L及其对应的值。它将这些值合并在一起，形成一个可能更小的值集。通常，每个Reduce调用产生0或1个输出值。中间值value是通过迭代器提供给用户的

reduce函数。

不同的MapReduce接口实现是可能的，正确的选择需要根据特定的环境。一台机器通常有1千兆/秒的网络带宽。计算集群包含数千台计算机，因此机器故障是常见的。存储由直接连接到单个机器的廉价IDE磁盘提供。GFS是一个内部开发的分布式文件系统，用于管理存储在这些磁盘上的数据。文件系统使用复制(通常是三重复制)在不可靠的硬件上提供可用性和可靠性。用户将Job提交给调度系统。每个作业由一组任务组成，这些任务由调度系统映射到集群中的一组可用计算机上。

MapReduce 编程模型在 Google 内部成功应用于多个领域。我们把这种成功归结为几个方面：首先，由于 MapReduce 封装了并行处理、容错处理、数据本地化优化、负载均衡等等技术难点的细节，这使得 MapReduce 库易于使用。即便对于完全没有并行或者分布式系统开发经验的程序员而言；其次，大量不同类型的问题都可以通过 MapReduce 简单的解决。比如，MapReduce 用于生成 Google 的网络搜索服务所需要的数据、用来排序、用来数据挖掘、用于机器学习，以及很多其它的系统；第三，我们实现了一个在数千台计算机组成的大型集群上灵活部署运行的 MapReduce。这个实现使得有效利用这些丰富的计算资源变得非常简单，因此也适合用来解决 Google 遇到的其他很多需要大量计算的问题。 我们也从 MapReduce 开发过程中学到了不少东西。首先，约束编程模式使得并行和分布式计算非常容易，也易于构造容错的计算环境；其次，网络带宽是稀有资源。大量的系统优化是针对减少网络传输量为目的的：本地优化策略使大量的数据从本地磁盘读取，中间文件写入本地磁盘、并且只写一份中间文件也节约了网络带宽；第三，多次执行相同的任务可以减少性能缓慢的机器带来的负面影响（alex注：即硬件配置的不平衡），同时解决了由于机器失效导致的数据丢失问题。

而有关任务粒度，选择M和R的大小。首先M和R的数量应大于worker数量很多，这样可以提高负载均衡，并且发生故障可以尽快恢复。其次，在实践中，倾向于选择M使得每一个任务为16MB或者64MB，R则为worker的数倍。文中提到经常采用的worker数量为2000，M为200000，R为5000。延长MapReduce操作所花费的总时间的常见原因之一是落后的机器（straggler），即，在计算中完成最后几个map或reduce任务之一需要非常长时间的机器。Stragglers可能出于各种原因而出现。例如，具有坏磁盘的计算机可能会遇到频繁的可纠正错误，从而将其读取性能从30MB / s降低到1MB / s。当MapReduce操作接近完成时，master会调度剩余正在进行的任务的备份执行。无论主要执行还是备份执行完成，任务都会标记为已完成。它通常会将操作使用的计算资源增加不超过百分之几，但这种操作大大减少了完成大型MapReduce操作的时间。例如，当禁用备份任务机制时，排序程序需要多花44%的时间才能完成。

从实际集群例子出发，在大型机器集群上运行的两个计算来测量MapReduce的性能。一个计算搜索大约1TB的数据以寻找特定模式。另一个计算对大约1TB的数据进行排序。有关集群配置：程序在由大约1800台机器组成的集群上执行。每台机器都有两个2GHz Intel Xeon处理器，支持超线程，4GB内存，两个160GB 的IDE磁盘和一个千兆以太网链路。在4GB内存中，保留了大约1-1.5GB的内存为其他任务使用。

另外了解到MapReduce算法的优缺点：

MapReduce优点：

提供了抽象接口，分布式过程完全隐藏，使得程序员容易使用；
使得大规模处理数据变得可能；
自动的负载均衡；
自动的容错性；

MapReduce缺点：

极其严格的数据流；
很多常见的操作也必须手写代码；
程序内部实现隐藏，优化比较困难。

GFS诞生是为了设计一个高效的顺序读写的分布式文件系统。

“21世纪的竞争是数据的竞争，谁掌握了数据，谁就掌握了未来。” 马云曾经这样说过。由此可见，大数据在当今社会中是多么重要。从百度的检索信息，淘宝的购买数据到自媒体的新闻数据……我们的生活中似乎处处都充斥着数据，仿佛我们已经被数据包围。GFS主要讲述了GFS文件系统的一些基本情况和原理，GFS文件系统是一个面向大规模数据密集型应用的、可伸缩的分布式文件系统，它满足Google迅速增长的数据处理需求，而在论文的开始，我了解到它的一个特点，GFS是由普通的廉价设备组装的，它能以廉价的设备发挥出高性能我认为这是一个最大的特点，然后一个GFS集群包含一个单独的Master节点和多台Chunk服务器，而且还能够被多个客户端访问，它能够减少客户端和Master节点通讯的需求。这让我想到了，我们可以利用这个论文论述的东西去制作一个信息收集系统，就比方说我国吧。众所周知，我国是一个人口大国，因此在人口普查时，就会有比较麻烦的问题了。一方面是去调查，另一方面是如何将这些众多的信息收集在一个比较小的系统中。而google-file-system给了我们很好的榜样。这种高性能的存储，极大化的利用了文件系统。而且我们还能用它在单位时间内进行高次数的数据访问，因此给予了我们在处理大量数据及获取大批数据的问题以答案。不仅是在这个邻域，想必在这个数据如此爆炸的时代，必定是有其他的运用的。

GFS是Google File System的缩写，字面意义上就是Google的文件系统，技术层面上来讲，GFS是Google在2003年前后创建的可扩展分布式文件系统 ，用来满足 Google 不断扩展的数据处理需求。GFS 为大型网络和连接的节点提供容错、可靠性、可扩展性、可用性和性能。GFS 由多个由低成本商品硬件组件构建的存储系统组成。它经过优化以适应 Google 的不同数据使用和存储需求，例如其搜索引擎会生成大量必须存储的数据。在我看来这个系统最大的亮点有两个，一个是使用集群的概念，将硬件设施连接起来共同进行文件存储任务；另外一个是充分利用了现成服务器的优势，同时最大限度地减少了硬件弱点。

参考网上的中文翻译，总体设计大概为

1.一个master（有复数个副本）。master保存着文件名对应到位置的映射表。

2.n个trunk server （同样也是有副本的），trunk server存储了实际的文件数据。文件的分片单位为trunk。

3.master知道所有的文件，所有的trunk server，所有的trunk。

GFS由单个主服务器和多个数据块服务器组成，并由多个客户端访问。文件被分成固定大小的64MB数据块。每一个数据块都有一个不可变且全局唯一的快处理程序，由主控服务器在创建块的时候分配的。默认情况下，每个文件块都会被复制到3个不同的块服务器上。

需要解决的问题都是分布式要解决的性能，容错，复制和一致性。

在性能方面，使用多台机器的资源来完成工作可以显著提高性能。因此，自然的想法是将数据切片，将其分解成复杂的部分，并将其分布在不同的机器上，这样就可以同时从不同的机器读取数据。在容错方面，一旦同时运行成百上千台机器，小的错误就会被放大，低概率的错误可能会变成频繁的错误。此时，需要一个良好的容错系统。容错的一个极其简单的概念是，有多个数据备份，如果一个备份失败，则切换到另一个备份。但是，一旦使用了复制，系统可能会导致数据不一致。为了解决一致性问题，需要做更多的工作，这可能会影响性能。GFS使用弱一致性，它们在性能和一致性之间取得了平衡。

所以就其本质，还是在CAP三大块儿上进行取舍

数据更改后文件区域（file region）的状态依赖于更改的类型，是否成功或者失败，或者是否是并发的更改。如表1所示，如果所有客户端无论从哪个副本读取到的数据都是相同的，则文件区域是一致的。如果文件区域是一致的，并且客户端可以读取到所更改的内容，则是已定义的。

谷歌文件系统的文件读写模式不同于传统的文件系统。在诸如Search之类的谷歌应用程序中，对文件所做的大多数更改都不会覆盖原始数据，而是将新数据追加到文件的末尾。对文件的随机写入几乎不存在。对于如此大的文件访问模式，客户端没有意义阻塞缓存，并且追加成为性能优化和原子性(将事务视为程序)。它要么完全实现，要么根本不实现)保证的焦点。写操作是将数据写入用户指定的偏移量，而记录追加操作是将数据追加到GFS选择的偏移量。相反，常见的追加操作只是将数据写到客户端考虑的文件的末尾。如果存在并发，则追加至少以原子方式进行一次。在append之后，GFS将偏移量返回给客户端，并标记包含该记录的已定义区域的开头。在连续的更改成功之后，被更改的文件区域将被定义，并包含上次更改的数据内容。GFS确保(1)在所有副本上以相同的顺序进行更改。(2)使用chunk版本号检测旧副本，旧副本可能不会因为Chunkserver的异常而改变。旧的拷贝在适当的时候被回收。因为客户端缓存了关于块位置的信息，所以它可能会在缓存的信息失效之前读取旧的副本。此窗口由缓存信息的超时时间决定。

GFS构建时，一个GFS集群包含一个主节点、多个块服务器，同时被多个客户端访问。存储在GFS中的文件被分成固定大小的块。为了方便识别存储的数据，在创建每个块并将其作为Linux文件存储在本地硬盘上时，为其分配一个全局惟一的64位块标识符。单个主节点极大地简化了GFS的设计，因为客户端使用主节点来查找要查询的块服务器。客户端将元数据缓存一段时间，然后直接从chunk服务器读取数据。对于存储在主内存中的元数据，有三种主要类型的元数据:文件和块命名空间、文件和块映射以及每个块副本的位置。我自己的理解是，这些类型的元数据将容易阅读，保持主服务器与块同步，不会导致主服务器崩溃。

文件的读取和写入，对于读取和写入效率，当读取和写入的客户机增加时，多个客户机同时读取和写入的几率也增加，导致整体的读取和写入效率降低。感觉就像我们平时用网线，当多个人用同一条网线时，你的网速就会变得很慢，放视频打游戏动不动就卡机。gfs系的读取有两种操作，一种是大规模的流式读取，大规模的流式读取通常一次读取数百KB的数据，甚至更多。而另外一种是小规模的随机读取，并且在gfs中对于记录追加通过chunk和chunk的副本进行操作，保证至少有一次原子的写入操作成功执行。

GFS变相向我们证明了弱一致性在实际应用中也是可行的，但是需要搭配某种形式的应用程序级检查机制。分布式文件系统作为基础设施对于很多数据密集型应用程序非常有用。这也就印证了随着数据规模的日益扩大，为什么所有的大厂都会自研一套分布式文件系统了。另外，如果我们将主服务器中的元数据与块服务器中的存储分开，并最小化客户端对主服务器的调用，那么单个主服务器模式是可行的。我们也可以使用文件分块来实现并行操作，或是使用主副本块服务器对所有的副本的写入操作进行排序。