Google“三驾马车”读后感

18301140 徐奕珂

一、前言

应课程学习要求，我阅读了被称为Google“三驾马车”的三篇论文。我按照Bigtable、GFS、MapReduce的顺序进行了阅读，并同步查阅了相关资料，以便更清晰了解其中知识。在初步阅读完三篇论文后，我又对Bigtable论文进行了更深入地学习，本篇读后感的重心也将放在Bigtable部分。

以下为我的主要学习收获。

二、概述

Google”三架马车“的发布顺序，为GFS（2003）, MapReduce（2004）和Bigtable（2006）。后面两篇论文都是以GFS为基础，而BigTable 是建立在 GFS 和 MapReduce 之上的。 如此，三大基础核心技术构建出了完整的分布式运算架构。

谷歌的三篇论文分别介绍了Google Bigtable, Google File-System和Google MapReduce三个重要工具。它们有一个共同点——分布式系统。而这个分布式就是将一个业务分拆多个子业务，化整为零，分别放在不同的服务器上。经过输入、分块、整合、输出四个步骤，将一个数量级的大任务，分配给多个处理单元共同分工、合作，最后汇总完成。采用分割和拆分的方法去处理大量的数据，把问题分解成为大量的“小”任务，很好地起到了化繁为简的作用。这种“化整为零”再“化零为整“的思想，可以使很多复杂的问题得到简单的解决。

三、Bigtable

Bigtable是一个分布式的结构化存储系统，被Google用来处理海量数据（通常是PB级），这些项目对Bigtable需求差异很大，但Bigtable都能提供很灵活的服务，可以体现出其功能的强大。

Bigtable包括了三个主要的组件：链接到客户程序中的库、一个Master服务器和多个Tablet服务器。针对系统工作负载的变化情况，BigTable可以动态的向集群中添加（或者删除）Tablet服务器。Master服务器主要负责以下工作：为Tablet服务器分配Tablets、检测新加入的或者过期失效的Table服务器、对Tablet服务器进行负载均衡、以及对保存在GFS上的文件进行垃圾收集。除此之外，它还处理对模式的相关修改操作，例如建立表和列族。每个Tablet服务器都管理一个Tablet的集合（通常每个服务器有大约数十个至上千个Tablet），每个Tablet服务器负责处理它所加载的Tablet的读写操作，以及在Tablets过大时，对其进行分割。

读完该篇论文后我印象最深的是，Bigtable对故障处理机制考虑得非常周全。

BigTable使用Chubby（一种高可用的、序列化的分布式锁服务组件）跟踪记录 Tablet 服务器的状态。Master服务器负责检查一个Tablet服务器是否已经不再为它的Tablet 提供服务了，并且要尽快重新分配它加载的Tablet。Master服务器通过轮询Tablet服务器文件锁的状态来完成检测。假设，一旦Tablet服务器在Chubby上的服务器文件被删除了，Master服务器就把之前分配给它的所有的Tablet放入未分配的Tablet集合中。为了确保Bigtable集群在Master服务器和Chubby之间网络出现故障的时候

仍然可以使用，Master服务器在它的Chubby会话过期后主动退出。无论Master服务器发生何种故障，都不会改变现有Tablet在Tablet服务器上的分配状态。

Tablet 的持久化状态信息保存在 GFS 上。更新操作提交到 REDO 日志中。在这些更新操作中，最近提交的那些存放在一个排序的缓存中，称为memtable。当对Tablet服务器进行读操作时，Tablet服务器会作类似的完整性和权限检查。一个有效的读操作在一个由一系列SSTable和memtable合并的视图里执行。由于SSTable和memtable是按字典排序的数据结构，因此可以高效生成合并视图。当进行 Tablet 的合并和分割时，正在进行的读写操作能够继续进行，提高了效率。

由于Bigtable用于处理大量的数据，对存储空间的需求量很大，因而，Bigtable对于空间的缩减与回收也达成了良好的实现。Minor Campaction收缩Tablet服务器使用的内存，Merging Campaction合并SSTable文件并限制文件数量，Major Campaction合并所有的SStable并生成一个新的SSTable，从而达到回收空间重复使用的目的。

Bigtable已经实现了下面的几个目标：适用性广泛、可扩展、高性能和高可用性。Bigtable的高性能和高可用性，让其能够满足各种应用需求，让开发人员可以根据的自己的系统对资源的需求增加情况，通过简单的增加机器，就可以扩展系统的承载能力。

在论文的最后，开发人员也分享了在设计、实现、维护和支持Bigtable的过程中的一些教训：很多类型的错误都会导致大型分布式系统受损，这些错误不仅仅是通常的网络中断、或者很多分布式协议中设想的 fail-stop 类型的错误；在彻底了解一个新特性会被如何使用之后，再决定是否添加这个新特性是非常重要的；系统级的监控对 Bigtable 非常重要。通过从雏形阶段到逐渐发展壮大，每一种设计都必须在真实环境中应用，某些设计对于特定环境适合，在其他环境却有差错，针对不同的需求需要做出不同东西与之匹配。

整个BigTable设计符合大部分大数据程序的需求，打破了关系型数据库的结构化存储，能够部署在成千上万台服务器上，可以存储PB级数据，对互联网行业的快速发展提供了坚实的理论基础与成功案例。

四、GFS

Google的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。GFS虽然运行在廉价的普遍硬件设备上，但是它依然提供了容错功能，为大量客户机提供了高性能的服务。

GFS与传统的分布式文件系统有着很多相同的设计目标，比如，性能、可伸缩性、可靠性以及可用性。但是，GFS和早期文件系统的假设有明显的不同。GFS文件系统重新审视了传统文件系统，设计出了完全不同的设计思路。首先针对组件频繁失效的问题，将持续的监控、错误侦测、容错功能以及自动恢复机制集成在GFS中，设计了快速恢复和复制机制。快速恢复，通过创建Chunk和Master副本的方式；其次由于文件巨大，假设条件和参数，I/O操作和Block的尺寸都经过重新考虑，为了保持数据的完整性每个用于存储数据的Chunk服务器都使用Checksum来检查保存的数据的完整性；然后是由于海量文件的访问模式，采用在文件末尾追加数据，而不是覆盖原有数据，通过记录追加操作实现了生产者-消费者队列模式；最后应用程序和文件系统API协同设计，提高系统的灵活性。

GFS上的文件往往非常大，通常在几千兆字节（GB）范围内。访问和操作大的文件将占用网络的大量带宽。GFS通过将文件分成每块64兆字节（MB）来解决这个问题，每个块接收一个称为块句柄的唯一64位标识号。虽然GFS可以处理较小的文件，但是它的开发人员没有为这些类型的任务优化系统。通过要求所有文件块大小相同，GFS简化了资源应用程序，很容易看出系统中哪些计算机接近容量，哪些计算机使用不足，从而将块从一个资源移植到另一个资源，也能够很容易地平衡整个系统的工作负载。

同时，Google针对不同的应用部署了多套GFS集群，一个集群包含一个单独的Master节点、多台Chunk服务器，能够同时被多个客户端访问。Master节点管理所有的文件系统的元数据，Chunk服务器用于将GFS存储的文件的以Linux文件形式保存到本地磁盘中，用Chunk标识数据和管理数据，GFS客户端代码以库的形式链接到客户程序中，客户端代码实现了GFS文件系统的API接口函数、应用程序与

Master节点和Chunk服务器通讯、以及对数据进行读写操作，控制流是在Mastr服务器处理，而数据流在Chunk服务器和客户端处理，这样的方式能够将中心的Master服务器的负担降到最低。

总结得出，GFS架构的设计优点有：（1）GFS实现了控制流与数据流的分离；（2）采用中心服务器模式；（3）无论客户端还是Chunk服务器都不需要缓存文件数据；等等。

结合于传统的文件系统，和更加详细全面的考虑，Google设计出了这样一个交互更可靠稳定、具有自己容错和诊断的机制、并能够在大量的并发读写操作时提高合计吞吐量的文件系统，GFS，从而提高了文件系统整体性能和容错的能力。

五、MapReduce

MapReduce是一个编程模型，也是一个处理和生成超大数据集的算法模型的相关实现。用户首先创建一个Map函数处理一个基于key/value pair的数据集合，输出中间的基于key/value pair的数据集合，然后再创建一个Reduce函数用来合并所有的具有相同中间key值的中间value值。它在运行时只关心：如何分割输入数据，在大量计算机组成的集群上的调度，集群中计算机的错误处理，管理集群中计算机之间必要的通信。

MapReduce模型能够处理因为数据量大而将计算分布在成百上千的主机上产生的并行计算、分发数据、错误处理等问题，它将并行计算、容错、数据分布、负载均衡等复杂的细节封装在一起，也能让那些并没有并行计算和分布式处理系统开发经验的程序员有效利用分布式系统的丰富资源，这也是它的一大优点所在。

MapReduce模型可根据当前的具体环境而采用不同的实现方式，如：小型的共享内存方式的机器、大型NUMA架构的多处理器的主机、大型的网络连接集群。MapReduce架构的程序能够在大量的普通配置的计算机上实现并行化处理，通过Map调用的输入数据自动分割为M个数据片段的集合，然后Map调用被分布到多台机器上执行，用户程序在机群中创建大量的副本(一个主副本程序Master，其它的是Worker程序)，Master分配M个Map任务和R个Reduce任务，而Worker程序则负责处理Map或Reduce任务，读取相关的输入数据片段解析出key/value pair后，传递给用户自定义的Map函数，生成并输出中间的key/value pair，将通过分区函数将其缓存在内存中的R个区域，之后周期性的写入到磁盘中，Master会记录其在磁盘上的位置并传递给Reduce Worker, R个区域的主机则会从Map Worker所在的主机读取数据，进行相同key值的数据聚合排序，这个过程由Reduce Worker完成，当所有的任务完成之后，Master唤醒用户程序，用户对MapReduce的输出进行返回。

任务处理过程中用到了成千上百的机器，会出现一定的问题，所以Master还负责周期性的ping每个Worker，监测机器故障。这个过程中产生的存储问题则是通过GFS文件系统来管理。

MapReduce编程模型将并行处理、容错处理、数据本地化、负载均衡等问题在一个库里面方便开发人员使用，能够解决网络搜索服务、排序、数据挖掘和机器学习等各个方面的需求，可以在一个数千台计算机组成的大型集群上部署，让开发人员可以更好地利用计算资源和解决大量计算问题。

六、总结

Google在2003年到2006年公布了关于GFS、MapReduce和BigTable三篇技术论文，这也成为后来云计算发展的重要基石，如今Google已经发布了后Hadoop时代的新“三驾马车”——Caffeine、Pregel、Dremel，再一次影响着全球大数据技术的发展潮流。

但对于初学者来说，从经典的三大论文开始学起，非常有助于了解大数据、云计算的相关基础知识。我在读完三篇论文之后，对介绍的Bigtable, GFS和MapReduce三项技术搭建起了大致的概念，并借此机会，对其中涉及的众多计算机领域专业术语、用词有了了解。

除此之外，耐心读完这样经典的、优秀的技术介绍论文，虽然过程略有枯燥，但坚持读完后，能让我详细了解到一个实际应用的大型项目是如何设计、构造与优化的，对我今后的项目开发给了很大的启迪。有时，读到困难问题的解决方案部分，我甚至要忍不住点头称赞。我发现，在Google这三篇论文中都有提到一点经验分享：简洁的设计和编码给维护和调试能带来巨大好处（”The most important lesson we learned is the value of simple designs.“）。这对于我这种编程常常停留在”学生思维“而不是”开发者思维“的软件人来说是很值得深思的，我们经常为了向老师展示自己的编程能力，会用复杂的过程去实现功能，而并未考虑其可靠性、鲁棒性。

技术的更新迭代非常迅速，如果想跟住科技的步伐不被落下，我需要坚持不断地学习。