# 关于Google大数据领域的‘三驾马车’——即Bigtable、GFS、MapReduce读后有感

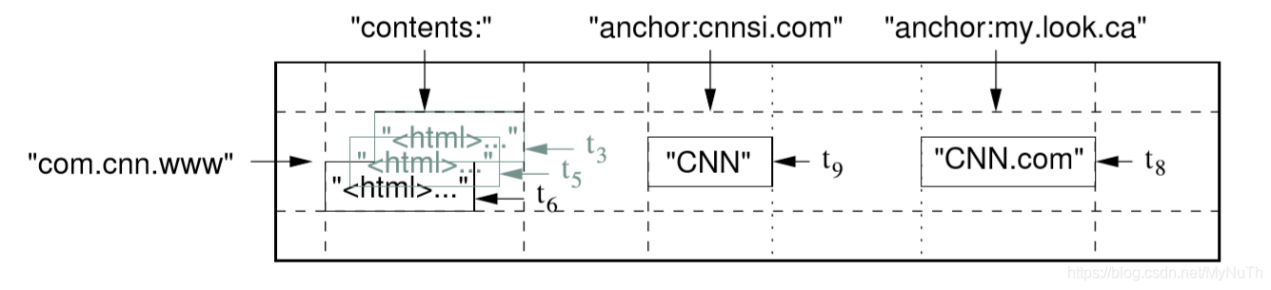
所谓的Google三驾马车，是指Google创造的3项革命性技术：GFS、MapReduce和Bigtable。Google发布的这三个产品的详细设计论文，奠定了风靡全球的大数据算法的基础，而当前流行的大数据技术，也都是在谷歌发表了这三大论文以后，不断的发展起来的。

而我们学习的非关系型数据库，就与这三驾马车有着密不可分的联系。

## 一、Bigtable

简单来说，Bigtable是一个用于管理结构化数据的分布式存储系统，主要针对大的数据规模：数千个商用服务器上的数PB数据。Google的许多项目都在Bigtable中存储数据，包括网络索引，Google Earth和Google Finance。这些应用程序会对Bigtable提出非常不同的需求，无论是在数据块大小方面（从URL链接的网页到卫星图像），抑或是延迟要求（从后端批量处理到实时数据服务）。尽管需求多种多样，但是Bigtable已经成功地为这些Google产品提供了灵活，高性能的解决方案。论文中介绍了Bigtable提供的数据模型，Bigtable不支持完整的关系数据模型; 相反，它为client提供了一个简单的数据模型，使得client可以动态控制数据的布局和格式。数据是使用行和列名称索引的，可以是任意字符串。Bigtable还将数据视为未解释的字符串，尽管client通常会将各种形式的结构化和半结构化数据序列化为这些字符串。

### 1.数据模型

Bigtable是一个稀疏的，分布式的，持久的多维有序map。 map由行键，列键和时间戳索引;每个值都是未解释的byte数组。  


Rows：表中的行键（row keys）是任意字符串（当前大小为64KB，但对于大多数用户来说，10-100字节是典型大小）。单个行键下的每次数据读取或写入都是atomic（无论行中读取或写入的不同列的数量如何）。Bigtable按字典顺序排列行键来维护数据。表的行范围（row range）是动态划分的。规划的row range也被称为tablet（简单来说，就是确定一定的行数被称作tablet），它是分配和负载平衡的单位。较短的行范围读取效率更高，并且通常仅需要与少量机器通信。

Column Families：列的key通过以下形式命名：family: qualifier。列的key被分组为Column Families，存储在Column Families中的所有数据通常具有相同的类型。Column Families创建之后，才可以将数据存放在此family的qualifier下。一个表中的不同Column Families的数量很小（最多几百个），而且这些Column Families在运行期间很少发生变化。但是table可以具有无限数量的列（通过qualifier），Column Families构成了访问控制的基本单元。

Timestamps：时间戳的两个主要作用是：管理不同版本和垃圾回收。Bigtable中的每个单元格都可以包含相同数据的多个版本；这些版本由时间戳索引。Bigtable时间戳是64位整数。它们可以由Bigtable分配，在这种情况下，它们以微秒表示“实时”，或者由客户端应用程序显式分配。需要避免冲突的应用程序必须自己生成唯一的时间戳。以递减的时间戳顺序存储不同版本的单元，以便可以首先读取最新版本。可以通过两种设置，告诉Bigtable自动垃圾回收，来保证保存的数据版本不会过多。客户端可以指定仅保留单元的最近n个版本，或者仅保留足够新版本（例如，保留不超过特定时间的版本，如五天）。上面图中显示只保留每个页面的最近三个版本。

### 2.API

Bigtable提供了建立和删除表以及列族的API函数。Bigtable还提供了修改集群、表和列族的元数据的API，比如修改访问权限。

客户程序可以对Bigtable进行如下的操作：写入或者删除Bigtable中的值、从个别行中查找值、或者遍历表中的一个数据子集。

Bigtable还支持一些其它的特性，这些特性允许用户以更复杂的方法操作数据。首先，Bigtable支持单行上事务处理，利用这个功能，用户可以对存储在一个单独行关键字下的数据执行原子性的读取-更新-写入操作。虽然Bigtable提供了一个允许用户跨行关键字批量写入的接口，但是，Bigtable目前还不支持通用的跨行事务处理。其次，Bigtable允许把数据项用做整数计数器。最后，Bigtable支持在服务器的地址空间内执行脚本程序。脚本程序使用Google开发的用于数据处理的Sawzall语言书写。目前，我们基于Sawzall的API还不允许客户脚本将数据写回Bigtable，但是它允许多种形式的数据转换、基于任意表达式的数据过滤、以及通过多种操作符的汇总归纳。

Bigtable可以和MapReduce一起使用，MapReduce是Google开发的运行大规模并行计算的框架。我们已经开发了一套封装器，这些封装器使Bigtable既可以作为MapReduce作业的源输入也可以作为目标输出。

关于MapReduce，我会在之后的篇幅里继续说明。

### 3. Bigtable构件

Bigtable是建立在一些其他Google基础架构之上的。Bigtable使用GFS存储日志和数据文件。Bigtable集群往往运行在一个共享的机器池中，池中的机器还会运行其它各种各样的分布式应用程序，Bigtable的进程经常要和其它应用的进程共享机器。Bigtable依赖集群管理系统在共享机器上调度作业、管理资源、处理机器的故障、以及监视机器的状态。

#### SSTable文件格式

Bigtable数据在内部使用Google SS Table文件格式存储。SSTable提供一个从键到值的持久化的、已排序、不可更改的映射，这里的key和value的都是任意的字节串。对SSTable提供了如下操作：查询与一个指定key值相关的value，或者遍历指定key值范围内的所有键值对。从内部看，SSTable是一连串的数据块。SSTable使用块索引来定位数据块；在打开SSTable的时候，索引被加载到内存。一次查找可以通过一次磁盘搜索完成：首先执行二分查找在内存索引里找到合适数据块的位置，然后在从硬盘中读取合适的数据块。也可以选择把整个SSTable都映射到内存中，这样就可以在不用访问硬盘的情况下执行查询搜索了。

#### Chubby分布式锁服务

Bigtable还依赖一个高可用的、持久化的分布式锁服务组件，叫做Chubby。一个Chubby服务包括了5个活动的副本，其中一个副本被选为Master，并且积极处理请求。只有在大多数副本正常运行，并且彼此之间能够互相通信的情况下，Chubby服务才是可用的。当有副本失效的时候，出现故障时Chubby使用Paxos算法保证副本的一致性。Chubby提供了一个名字空间，里面包括了目录和小文件。每个目录或者文件可以当成一个锁使用，对文件的读写操作都是原子的。Chubby客户程序库提供对Chubby文件的一致性缓存。每个Chubby客户程序都维护一个与Chubby服务的会话。如果客户程序不能在租约到期的时间内重新签订会话租约，这个会话就过期失效了(A client’s session expires if it is unable to renew its session lease within the lease expiration time.)。当一个客户会话失效时，它拥有的锁和打开的文件句柄都失效了。Chubby客户程序可以在Chubby文件和目录上注册回调函数，当文件或目录改变、或者会话过期时，回调函数会通知客户程序。

Bigtable使用Chubby完成以下各种任务：保证在任意时间最多只有一个活动的Master；存储Bigtable数据的引导程序的位置；发现tablet服务器，以及在Tablet服务器失效时进行善后；存储Bigtable的模式信息；以及存储访问控制列表。如果Chubby长时间无法访问，Bigtable就会失效。最近我们在跨越11个Chubby服务实例的14个Bigtable集群上测量了这个影响。Bigtable服务器时钟的平均比率是0.0047%,在这期间由于Chubby不可用而导致Bigtable中的部分数据不能访问。单个集群里受Chubby失效影响最大的百分比是0.0326%。

### 4.实现

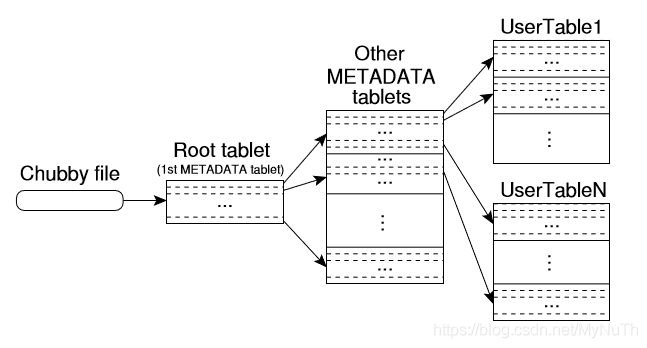
Bigtable实现有三个主要组件：链接到每个client的库，一个master server和许多tablet服务器。可以在集群中增加或删除Tablet server，以适应工作负载的变化。HBase可以看作是Bigtable的开源实现

master负责将tablet分配给Tablet server，检测Tablet server的添加和到期，平衡Tablet server负载以及GFS中文件的垃圾收集。此外，它还处理架构更改，例如表和column family的创建。

每个Tablet server都管理一组tablet。Tablet server处理对已加载的tablet的读写请求，并且还会拆分具有较大容量的tablet。

与许多single-master分布式存储系统一样，client数据不通过master：client直接与Tablet server通信以进行读取。因为Bigtable的client不依赖于master来获得定位信息，很多client从来不与master通信。

Bigtable集群存储了许多表。每个表包含一组tablet，每个tablet包含与行范围相关的所有数据。最初，每个表只包含一个tablet。随着表的增长，它会自动分成多个tablet，每个tablet大小约为100-200MB。



**Tablet location：**使用类似于B+树的数据结构（三层），来存储tablet的位置信息。第一级是存储在Chubby中的文件，其中包含root tablet的位置。root tablet包含METADATA表的位置。每个METADATA tablet都包含一组用户tablet的位置。root tablet只是METADATA表中的第一个tablet，但是经过特殊处理（永远不会拆分）以确保tablet位置层次结构不超过三层。METADATA表将tablet的位置存储在行键下，该行键是tablet的表标识符及其结束行的编码。每个METADATA行在内存中存储大约1KB的数据。限制 METADATA为128MB，三级定位方案足以满足234个tablet定位。

**tablet 分配：**每个tablet一次分配到一台tablet服务器。master会跟踪tablet服务器的集合，以及tablet到tablet服务器的当前分配，包括哪些tablet未分配。如果tablet未分配，并且tablet服务器具有足够的空间，则master会通过向tablet服务器发送tablet加载请求来分配tablet。Bigtable使用Chubby来跟踪tablet服务器。当tablet服务器启动时，它会在特定的Chubby目录中创建并获取一个唯一命名文件的独占锁。master监视此目录以发现tablet服务器。如果一个tablet服务器失去其独占锁，则该服务器将停止为其tablet提供服务：例如，由于网络分区导致服务器丢失其Chubby会话。Chubby提供了一种有效的机制，允许tablet服务器在不引起网络流量的情况下检查它是否仍然保持锁定。tablet服务器将尝试重新获取锁。如果文件不再存在，则tablet服务器将永远无法再次服务，因此它会自行终止。每当tablet服务器终止时，它就会尝试释放锁，以便master能够更快地重新分配其tablet。

**Tablet Serving：**tablet的持久状态存储在GFS中。更新被提交到提交日志。在这些更新中，最近提交的更新存储在称为memtable的排序缓冲区的内存中; 之前的更新存储在一系列SSTable中。要恢复tablet，tablet服务器会从METADATA表中读取其元数据。 此元数据包含组成tablet的SSTable列表和一组重做点（redo points），这些重做点是指向可能包含tablet数据的任何提交日志的指针。服务器将SSTable的索引读入内存，并通过应用自重做点以来已提交的所有更新来重构memtable。

**compactions：**当写操作执行时，memtable的大小增加。当memtable大小达到阈值时，memtable被冻结，同时创建一个新的memtable，冻结的memtable转换为SSTable并写入GFS。这个compaction过程有两个目标：它缩小了tablet服务器的内存使用量，并减少了恢复期间必须从提交日志中读取的数据量（如果此服务器死机）。compaction发生时，传入的读写操作可以继续.

## 二、GFS

The Google File System，是分布式文件系统的奠基性文章，之后很多实际的系统以此为基础进行了实现。Google设计GFS的出发点是为了满足Google快速增长的数据存储和处理需求。GFS与之前的分布式文件系统不同之处在于：GFS的设计主要是以Google应用的工作负载和对Google技术环境的观察为驱动，简单来说就是GFS是针对Google的需求开发的专用平台，而不是适合所有应用的通用平台。GFS适用于大型、分布式、数据密集型的应用程序。在普通的商用硬件上运行，提供容错功能。  
实际的效果就是：GFS成功满足了公司的存储需求。GFS作为存储平台，广泛的部署在Google中，用以产生和处理数据，以供服务和研究使用。最大的集群在超过一千台计算机的数千个磁盘上提供了数百TB的存储，可以由数百个客户端并发访问。

### 1.设计概览

设计假设：  
（1）系统构筑在普通的商用计算机之上，而这些计算机出现故障是常态。分布式文件系统包括数百乃至上千的普通商用计算机，同时为相当数量的客户机提供服务。文章中展示的故障有：应用的bug、操作系统bug、人为失误、磁盘失效、内存失效、连接失效、网络失效和停电。因此，必须为系统提供完整的：持续监控、错误诊断、自动恢复功能。  
（2）系统存储大量文件，预期为数百万量级，典型的是100MB大小或更大，多GB大小文件也很常见，同时GFS要支持小文件。  
（3）工作量主要包括两种read（读）操作：大量顺序读和少量随机读。  
（4）工作量也包括顺序写（write），文件一旦写好，就很少进行修改。

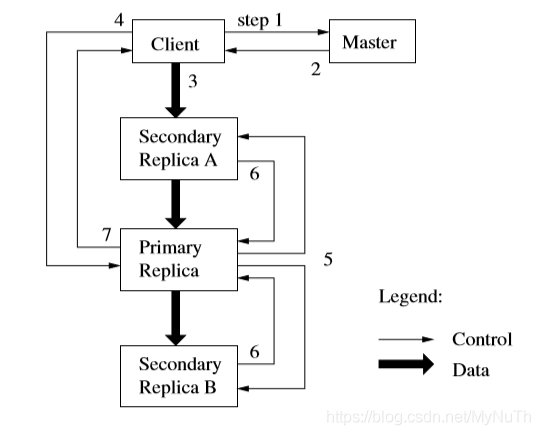
设计架构：

（1）文件：文件分为固定大小的块（chunk）。 每个块由master在创建块时分配不可变且全局唯一的64位块句柄（chunk handle）识别。 Chunk servers将块作为Linux文件存储在本地磁盘上，并通过明确块句柄（chunk handle）和字节范围来读写块数据。为了可靠性，每个块都在多个hunk server上复制。 默认情况下，存储三个副本。  
（2）单一的master：单一的master简化设计、便于管理。通过将控制流和数据流分开，降低单一master的压力。Clients不通过master读写文件，而只是询问master应该与哪一个chunkserver通讯，进而与文件进行交互。维护所有文件系统元数据。 这包括命名空间，访问控制信息，从文件到块的映射以及块的当前位置。 它还控制系统范围的活动，例如块租约管理，孤立块的垃圾回收以及块服务器之间的块迁移。master定期与Heartbeat消息中的每个chunkserver通信，以便为其提供指令并收集其状态。  
（3）chunk大小：chunk大小是关键设计参数之一。Google选择了64 MB，这比典型的文件系统块大小要大得多。加上惰性空间分配。  
（4）chunkserver：是典型的Linux机器。文章之后提到，Linux源码的可获得性帮助工程师来理解和探索系统行为，适当的时候，对内核进行修改。最开始选择的是Linux 2.2 内核版本，后来又迁移到Linux 2.4。chunkserver将chunk保存在本地，如同普通的Linux文件，可通过chunk handle和字节范围进行读和写。  
（5）Metadata（元数据）：Master存储三种主要的元数据：文件和块命名空间，从文件到块的映射，以及每个块的副本的位置。所有元数据都保存在master的内存中。由于元数据存储在内存中，因此master操作很快。此外，master在后台周期性地扫描其整个状态是容易和高效的。定期扫描用于实现块垃圾回收，在chunkserver故障时重新复制，以及块迁移以平衡块服务器之间的负载和磁盘空间使用。元数据存在内存的方法的一个潜在问题是，块的数量以及整个系统的容量受到master具有多少内存的限制。这在实践中不是严重的问题。master为每个64 MB的块维护少于64个字节的元数据。例如，在文章后面可以看到在两个集群中master元数据只占内存48MB和60MB，这对于普通服务器是可以轻易达到的。如果需要支持更大的文件系统，为主机添加额外内存的成本也很小。因此，通过将元数据存储在内存中来可以获得简单性，可靠性，性能和灵活性。  
（6）操作日志：操作日志包含关键元数据更改的历史记录。它是GFS的核心。它不仅是元数据的唯一持久记录，而且还充当了定义并发操作顺序的逻辑时间线。文件和块以及它们的版本都是由它们创建的逻辑时间唯一且永久地识别的。由于操作日志至关重要，因此必须可靠地存储它，并且在元数据更改持久之前不会对客户端进行更改。GFS在多台远程计算机上复制它，并且只有在本地和远程将相应的日志记录发送到磁盘后才响应客户端操作。master将多个日志记录一起批处理，从而减少了浮动和复制对整体系统吞吐量的影响。master通过重放（replaying）操作日志来恢复其文件系统状态。为了最大限度地缩短启动时间，我们必须保持日志不能过大。 只要日志超过一定大小，master就会检查其状态，以便通过从本地磁盘加载最新检查点（checkpoint）并在此之后仅重放（replaying）有限数量的日志记录来恢复。检查点（checkpoint）采用紧凑的B树形式，可以直接映射到内存中，用于命名空间查找而无需额外的解析。这进一步加快了恢复并提高了可用性。

### 2.一个简单的读操作

如GFS架构图所示，数据操作和控制操作分离。在一个简单的读操作中，首先，使用固定的chunk大小，client将文件名和由应用程序指定的字节转换为文件中的块索引（chunk index）。然后，它向master发送包含文件名和块索引的请求。master使用相应的块句柄（chunk handle）和副本的位置进行回复。client使用文件名和块索引作为key来缓存此信息。其次，client向其中一个副本发送请求，最大概率是距离最近的副本（根据系统网络拓扑结构，从IP地址信息进行精确估计）。该请求指定了块句柄和该块内的字节范围。在缓存的信息到期或文件重新打开之前，对同一块的进一步读取不再需要client-master交互。

### 3.系统交互：Leases 和 Mutation Order

Mutation是对块的内容或元数据的写入或追加操作。每个改变都在所有块的复制品上执行。Lease 中文叫租约，是一种广泛应用于分布式系统领域的协议，它是一种维护分布式系统一致性的有效工具。master将一个块租约（lease）授予其中三个存储中的一个副本，我们将其称为primary（主副本）。primary选择块的所有mutation的顺序。当数据写的顺序确定时，三个副本的储存内容就可以保持一致，因为所有副本都遵循此mutation顺序。lease机制旨在最大限度地减少master的管理开销。lease的初始设定为为60秒。如果时间到达仍然不能够完成操作，则可以进一步申请lease。  
因此，全局mutation顺序首先由master选择哪一个副本作为primary确定，其次，在租约内由primary设备分配的序列号确定。  
写入操作示意图：  
  
写操作步骤如下所示：

1. client询问master哪一个chunkserver持有当前块的lease（租约）和副本的位置。如果没有任何一个持有lease，master选择三个中的一个进行授权。
2. master回复primary的位置和secondary位置，client缓存这些数据为之后的mutations做准备。
3. client将数据发送给三个副本（即三个chunkserver，一个primary，两个secondary）。三个副本将数据存在LRU buffer中。通过将数据流和控制流分离，可以通过网络拓扑结构将数据以任何顺序快速发送给三个副本。
4. 一旦三个副本都完全接收全部数据信息，client发送给primary写请求。primary将收到的mutation操作分配连续的序列号，然后将操作应用在自身的数据之上。
5. primary将写请求发送给secondary，secondary将primary的mutation操作序列用于自身，则可以确保三个副本进行同样的mutation。
6. secondary回复primary已经完成操作。
7. primary回复client。  
   如果应用程序的写入内容很大或跨越块边界，client会将其分解为多个写入操作。

这与我们在课上所学知识相同。

## 三、MapReduce

MapReduce：Simplified Data Processing on Large Clusters最初发表在2004年，本次分享的是2008年的版本，内容较2004版本进行了精简和补充。  
在建立MapReduce之前，Google工程师会实现数百种特定的、大规模数据的计算，如：网上爬取文档，计算派生的数据（如数据图结构计算）等等。大多数此类计算在概念上是直截了当的，但是实现起来要分门别类的实现。输入数据通常很大，并且计算必须分布在数百或数千台机器上，以便在合理的时间内完成。为了应对这种复杂性，计算机工程中经常采用在复杂性之上抽象出一层，这样就可以很好的面对复杂的场景。MapReduce就是提供了一种抽象，允许使用的人将计算表达出来，同时隐藏了复杂的实现。  
文中提到，这种抽象受到Lisp语言中map和reduce，以及其他函数式编程语言的启发。文章作者认为，大多数相关计算都包括一个map操作，应用于输入中的每个record，以便计算一组intermediate key/value对，然后对所有共享相同key的value应用reduce操作。使用中，用户明确map和reduce函数的计算，运行系统自动在大规模集群机器上并行计算。文章认为程序员会发现MapReduce系统易于使用，从2004年到2008年，Google内部实现了超过一万个不同的MapReduce程序，平均每天在Google集群上执行十万个MapReduce作业，每天处理的数据超过二十PB，应用非常广泛。作者认为这项工作的最大贡献是一个简洁和高效的接口，可以在大规模商用PC集群（如GFS）上来进行并行化计算。

### 1.编程模型

计算任务中，输入为key/value对，输出也是key/value对。后面会介绍这些key/value对的异同。用户使用MapReduce可以通过两个函数：map和reduce。  
由用户编写map，采用输入对，并产生一组intermediate key/value。MapReduce库将同一中间key，比如key L关联的所有中间value组合在一起，并将它们传递给reduce函数。  
reduce函数也是由用户编写的，它接受中间key L，以及key L对应的value。它将这些value合并在一起以形成可能更小的value集合。通常，每次reduce调用只产生零个或一个输出值。中间值value通过迭代器提供给用户的reduce函数。  
对于MapReduce接口的不同实现是可能的，正确的选择是基于需要的具体环境来考虑。论文发表之后，有几个不同的关于Map Reduce的开源实现：Hadoop，The Phoenix system等。  
文章关于MapReduce的实现，是针对Google大规模采用的计算环境：通过交换式千兆以太网连接在一起的大型商用PC集群。在Google中，机器通常是运行在x86处理器上的Linux系统，每台机器有4-8GB内存。单个机器通常具有1 gigabit/second的网络带宽。计算集群包含数千台计算机，因此机器故障很常见。存储由直接连接到各个机器的廉价IDE磁盘提供。 GFS是一种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（通常为三份），在不可靠的硬件之上提供可用性和可靠性。用户将Job提交到调度系统。每个Job由一组task组成，并由调度系统映射到集群中的一组可用计算机。

### 2.MapReduce使用例子

考虑计算大量文档中每个单词出现次数的问题。即WordCount问题。map函数发出(emit)每个单词（word）加上相关的出现计数（在这个简单的例子中为1）。reduce函数将特定单词发出的所有计数汇总在一起。此外，用户可以编写代码，使用输入文件和输出文件的名称以及可选的调整参数，来填充mapreduce的规范对象（mapreduce specification object）。然后，用户调用MapReduce函数，将其传递给规范对象。用户的代码与MapReduce库（用C ++实现）链接在一起。  
2004的MapReduce论文包含此示例的完整程序文本。  
使用Google的MapReduce实现了超过一万个不同的程序，包括用于大规模图形处理，文本处理，数据挖掘，机器学习，统计机器翻译以及许多其他领域的算法。

### 3.执行概览

当用户调用MapReduce函数，如下过程将被执行（每一步对应于图中数字序号）：

用户程序调用MapReduce库，首先将输入文件拆分为M件，每件通常为16-64MB（可由用户通过可选参数控制）。然后，它会在一组计算机上启动该程序的许多副本。

该程序的副本之一-master-是特别的。其余的则称为worker。有M个map任务和R个Rreduce任务要分配。 master挑选闲置的worker并为每个worker分配一个map任务或reduce任务。

被分配了map任务的worker，读相应的输入split。worker从输入数据中解析key/value对，将其传入用户定义的map函数。intermediate key/value缓存在内存中。

周期性地，缓存对被写入本地磁盘，通过partitioning函数划分为R区域。位置信息将传递给master，进而在需要时候传递给reduce worker。

master将位置信息发送给reduce worker。当reduce worker读取其中间数据时，它会通过intermediate key对其进行排序，以便将所有出现的相同key组合在一起。如果中间数据量太大而无法容纳在内存中，则使用外部排序。

reduce worker遍历排好序的中间键值对，对每一个不同的key，将相应的key/value发送给reduce函数。reduce函数的输出附加到此reduce分区的最终输出文件。

### 4.本地性

在Google的计算环境中，网络带宽是一种相对稀缺的资源。利用输入数据存储在集群机器的本地磁盘上，从而可以节省网络带宽。GFS将每个文件划分为64MB块，并在不同的机器上存储每个块的几个副本（通常为3个副本）。 MapReduce中的master会考虑输入文件的位置信息，并尝试在包含相应输入数据副本的计算机上计划map任务。如果这个行不通，它会尝试在该任务的输入数据的副本附近安排map任务（例如，位于同一交换机的机器上）。运行大型MapReduce操作时，对于集群中的大部分worker，大多数输入数据在本地读取，不会消耗网络带宽。

### 5.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。