Bigtable 读后感

一．简介

Bigtable 是一种用于管理结构化数据的分布式存储系统，它被设计成非常大的规模：跨越数千个商品服务器的千兆字节数据。 谷歌的许多项目 在 Bigtable 中存储数据，包括网络索引、谷歌地球和谷歌金融。Bigtable 具有几个特点：广泛的适用性、可伸缩性、高性能和高可用性。 Bigtable 类似于数据库：它与数据库共享许多实现策略。它不支持完整的关系数据模型； 相反，它为客户端提供了一个简单的数据模型，它支持对数据布局和格式的动态控制，并允许客户端对所表示的数据的局部性属性进行推理 底层存储。

二．数据模型

Bigtable 是一个稀疏的、分布的、持久的多维排序地图。 映射由行键、列键和时间戳索引；映射中的每个值都是一个未解释的字节数组。

（1）map

一个 key-value 对，Bigtable 的键有三维，分别是行键（row key）、列键（column key）和时间戳（timestamp），行键和列键都是字节串，时间戳是 64 位整型；而值是一个字节串。可以用 (row:string, column:string, time:int64)→string 来表示一条键值对记录。

（2）row key

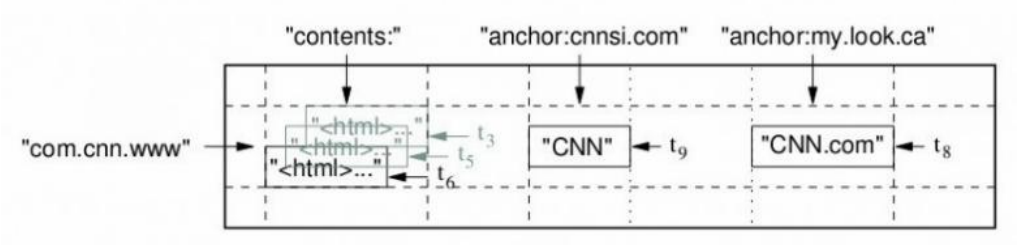
一个表中的行键，是任意的字符串。对每个行键下所包含的数据的读或写都是一个原子操作。

（3）Column key

列键被分组成称为“列家族”的集合，它成为基本的访问控制单元。存储在一个列家族当中的所有数据，通常都属于同一个数据类型（我们对同一个列家族中的数据一起进行压缩）。数据可以被存放到列家族的某个列键下面，但是，在把数据存放到这个列家族的某个列键下面之前，必须首先创建这个列家族。在创建完成一个列家族以后，就可以使用同一个家族当中的列键。

（4）Timestamp

在 BigTable 中的每个单元格当中，都包含相同数据的多个版本，这些版本采用时间戳进行索引。BitTable 时间戳是 64 位整数。BigTable 对时间戳进行分配，时间戳代表了真实时间，以微秒来计算。客户应用也可以直接分配时间戳。需要避免冲突的应用必须生成唯一的时间戳。



在 Webtable，每一行存储一个网页，其反转的 url 作为行键，比如 maps.google.com/index.html 的数据存储在键为 com.google.maps/index.html 的行里，反转的原因是为了让同一个域名下的子域名网页能聚集在一起。图 1 中的列族"anchor"保存了该网页的引用站点（比如引用了 CNN 主页的站点），qualifier 是引用站点的名称，而数据是链接

文本；列族"contents"保存的是网页的内容，这个列族只有一个空列"contents:"。图 1 中"contents:"列下保存了网页的三个版本，我们可以用("com.cnn.www", "contents:", t5)来找到 CNN 主页在 t5 时刻的内容

三．设计

（1）相关技术

Bigtable 依赖于 google 的几项技术。用 GFS 来存储日志和数据文件；按 SSTable 文件格式存储数据；用 Chubby 管理元数据。

（2）实现

包括三个主要部分：一个供客户端使用的库，一个主服务器（master server），许多片服务器（tablet server）。每个 table 包含一个 tablet 集合，每个 tablet 包含某个范围内 row 的所有数据。

每个 tablet 负责一定量的片，处理对其片的读写请求，以及片的分裂或合并。片服务器并不真实存储数据，而相当于一个连接 Bigtable 和 GFS 的代理，客户端的一些数据操作都通过片服务器代理间接访问 GFS。

主服务器负责将片分配给片服务器，监控片服务器的添加和删除，平衡片服务器的负载，处理表和列族的创建等。注意，主服务器不存储任何片，不提供任何数据服务，也不提供片的定位信息。

客户端需要读写数据时，直接与片服务器联系。因为客户端并不需要从主服务器获取片的位置信息，所以大多数客户端从来不需要访问主服务器，主服务器的负载一般很轻。

（3）Tablet

三级 B+ 树保存 tablet，root tablet 的位置存储在 Chubby，root tablet 存储 METADATA table 的所有 tablet 位置，root tablet 是 METADATA table 的第一个 tablet，永远不会分裂

tablet server 启动时创建并获取一个在 Chubby 目录上唯一命名的文件的独占锁，Master 监控这个目录来发现 tablet server。

Master 启动后执行的步骤：在 Chubby 获取唯一的 master 锁，防止多个 master 实例，扫描 Chubby server 目录找到所有活着的 server，与活着的 server 通信发现分配了哪些 tablet，扫描 METADATA table 找到 tablet 集合，有未分配的就分配给 tablet server。

当对 Tablet 服务器进行写操作时，Tablet 服务器首先要检查这个操作格式是否正确、操作发起者是否有执行这个操作的权限。权限验证的方法是通过一个从 Chubby 文件里读取出来的具有写权限的操作者列表来进行验证。成功的修改操作会记录在提交日志里。当一个写操作提交后，写的内容插入到 memtable 里面。

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。

Google File System 读后感

一．简介

GFS 是一个可扩展的分布式文件系统，用于大型分布式数据密集型应用程序。GFS 的服务器都是普通的商用计算机，并不那么可靠，集群出现结点故障是常态。因此必须时刻监控系统的结点状态。系统存储适当数量的大文件。理想的负载是几百万个文件，文件一般都超过 100MB，GB 级别以上的文件是很常见的，必须进行有效管理。支持小文件，但不对其进行优化。GFS 期望的应用场景应该是大文件，连续读，不修改，高并发。

二．设计体系

GFS 系统的节点可以分为三种角色：GFS Master（主控服务器）；GFS ChunkServer（CS，数据块服务器）；GFS 客户端。

GFS 文件被分为固定大小的数据块（chunk），由主控服务器在创建时分配一个 64 位全局唯一的 chunk 句柄。CS 以普通的 Linux 文件的形式将 chunk 存储在磁盘中。为了保证可靠性，chunk 在不同的机器中复制多份，默认为三份。

主控服务器维护了系统的元数据，包括文件以及 chunk 命名空间、文件到 chunk 之间的映射、chunk 位置信息。它也负责整个系统的全局控制，如 chunk 租约管理、垃圾回收无用 chunk、chunk 的复制等。主控服务器会定期与 CS 通过心跳的方式交换信息。

客户端是 GFS 提供给应用程序的访问接口，它是一组专用接口，不遵循 POSIX 规范，以库文件的形式提供。客户端访问 GFS 时，首先访问主控服务器节点，获取与之进行交互的 CS 信息，然后直接访问 CS，完成数据的存取工作。

三．出错处理

GFS 的服务器都是普通的商用计算机，并不那么可靠，集群出现结点故障是常态，如何恢复是一个大问题。

（1）容错

Master 容错：通过操作日志加 checkpoint 的方式进行，并且有一台称为”Shadow Master“的实时设备；Master 上保存三种元数据信息：命令空间：整个文件系统的目录结构以及 chunk 基本信息、文件到 chunk 之间的映射、chunk 副本的位置信息，每个 chunk 通常有三个副本 GFS Master 的修改操作总是新纪录操作日志，然后修改内存；Master 需要持久化前两种元数据，即命令空间及文件到 chunk 之间的映射，对于第三种元数据，ChunkServer 维护了这些信息，即使 Master 发生故障，也可以在重启时通过 ChunkServer 汇报来获取。

ChunkServer 容错：GFS 采用复制多个副本的方式实现 ChunkServer 的容错；ChunkServer 会对存储的数据维持校验和。

四．特点及总结

GFS 采用中心服务器的模式，该模式的最大优点是便于管理，因为中心服务器可以获知所有子服务器的状态、各个子服务器的负载状况等。但是缺点即单点故障。

（1）采用中心服务器模式

优点：可以方便的增加 Chunk Server；Master 可以掌握系统内所有 Chunk Server 的情况，方

便进行负载均衡；不存在元数据的一致性问题；
（2）不缓存数据
优点：文件操作大部分是流式读写，不存在大量重复的读写，因此即使使用 cache 对系统性能的提高也不大；Chunk Server 上的数据存储在本地文件系统，若真的出现频繁存取，那么本地文件系统的 cache 也可以支持；若建立系统 cache，那么 cache 中的数据与 Chunk Server 中的数据的一致性很难保证

Mapreduce 读后感

一．简介

Map Reduce 是用于处理和生成大型数据集的编程模型和相关实现，一种并行计算的编程模型，用于作业调度。GFS 和 BigTable 已经为我们提供了高性能、高并发的服务。如果我们的应用本身不能并发，那 GFS、BigTable 也都是没有意义的。Mapreduce 主要贡献是一个简单而强大的接口，使大规模 COMP 的自动并行化和分发成为可能。MapReduce 就是将一个大作业拆分为多个小作业的框架。

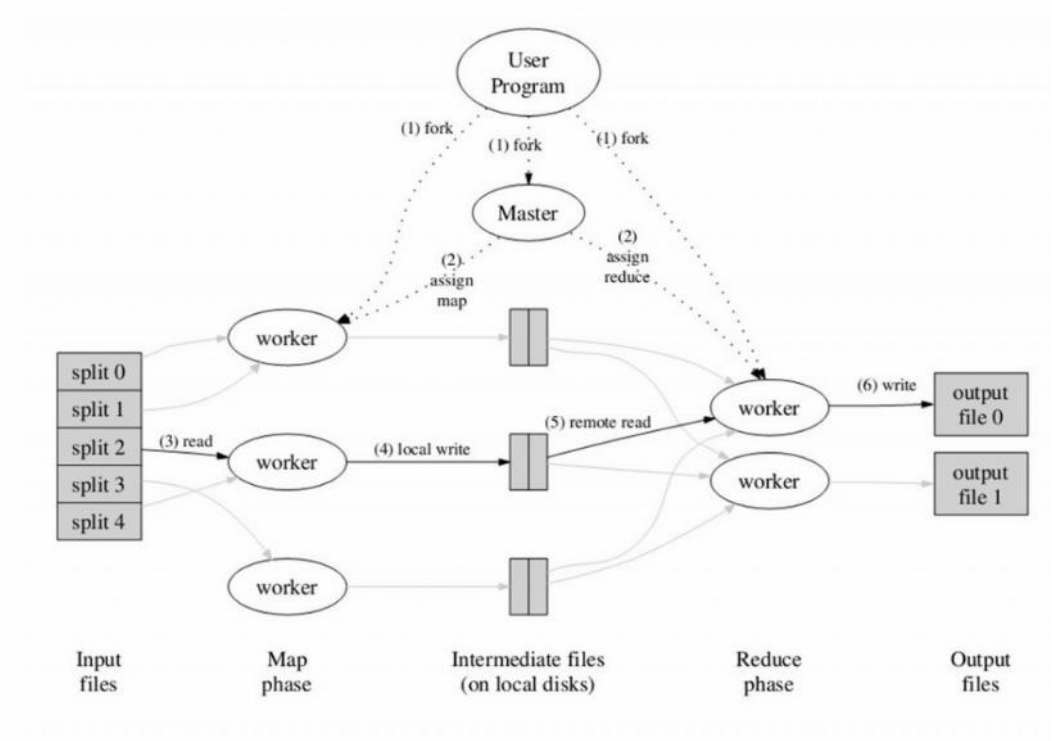
二．Map 和 Reduce

两个核心函数 Map 和 Reduce。

Map，由用户编写，接收一个输入对，产生一组中间 key/value 对。MapReduce 库将具有相同中间 key I 的聚合到一起，然后将它们发送给 Reduce 函数。

Reduce，也是由用户编写的，接收中间 key I 和这个 key 的值的集合，将这些值合并起来，形成一个尽可能小的集合。通常，每个 Reduce 调用只产生 0 或 1 个输出值。这些中间值经过一个迭代器（iterator）提供给用户的 reduce 函数。

三．流程设计



从 user program 开始的，user program 链接了 MapReduce 库，实现了最基本的 Map 函数和 Reduce 函数。

MapReduce 库先把 user program 的输入文件划分为 M 份（M 为用户定义），每一份通常有 16MB 到 64MB，如图左方所示分成了 split0~4；然后使用 fork 将用户进程拷贝到集群

内其它机器上。

user program 的副本中有一个称为 master，其余称为 worker，master 是负责调度的，为空闲 worker 分配作业（Map 作业或者 Reduce 作业），worker 的数量也是可以由用户指定的。被分配了 Map 作业的 worker，开始读取对应分片的输入数据，Map 作业数量是由 M 决定的，和 split 一一对应；Map 作业从输入数据中抽取出键值对，每一个键值对都作为参数传递给 map 函数，map 函数产生的中间键值对被缓存在内存中。

缓存的中间键值对会被定期写入本地磁盘，而且被分为 R 个区，R 的大小是由用户定义的，将来每个区会对应一个 Reduce 作业；这些中间键值对的位置会被通报给 master，master 负责将信息转发给 Reduce worker。

master 通知分配了 Reduce 作业的 worker 它负责的分区在什么位置，当 Reduce worker 把所有它负责的中间键值对都读过来后，先对它们进行排序，使得相同键的键值对聚集在一起。因为不同的键可能会映射到同一个分区也就是同一个 Reduce 作业，所以排序是必须的。

reduce worker 遍历排序后的中间键值对，对于每个唯一的键，都将键与关联的值传递给 reduce 函数，reduce 函数产生的输出会添加到这个分区的输出文件中。

当所有的 Map 和 Reduce 作业都完成了，master 唤醒正版的 user program，MapReduce 函数调用返回 user program 的代码。

所有执行完毕后，MapReduce 输出放在了 R 个分区的输出文件中。用户通常并不需要合并这 R 个文件，而是将其作为输入交给另一个 MapReduce 程序处理。整个过程中，输入数据是来自 GFS 的，中间数据是放在本地文件系统的，最终输出数据是写入 GFS 的。而且我们要注意 Map/Reduce 作业和 map/reduce 函数的区别：Map 作业处理一个输入数据的分片，可能需要调用多次 map 函数来处理每个输入键值对；Reduce 作业处理一个分区的中间键值对，期间要对每个不同的键调用一次 reduce 函数，Reduce 作业最终也对应一个输出文件。