非关系型数据库goole三驾马车理解摘要

18301041 黄林

因英文水平有限，选择中文版本进行研读，尽可能的不影响阅读质量

GFS（Google File System）论文总结

1.前言

GFS主要实现对大规模数据存储的需求是是大数据的理论的基础论文。是一个分布式文件系统，隐藏下层负载均衡，冗余复制等细节，对上层程序提供一个统一的文件系统API接口。预期的系统需要监控自身状态，将组件失效视为常态针对采用追加方式（有可能是并发追加）写入、然后再读取（通常序列化读取）的大文件进行优化，以及扩展标准文件系统接口、放松接口限制。通过统通过持续监控，复制关键数据，快速和自动恢复提供灾难冗余以及通过分离控制流和数据 流来实现在有大量的并发读写操作时能够提供很高的合计吞吐量。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。解决分布式文件存储！！！

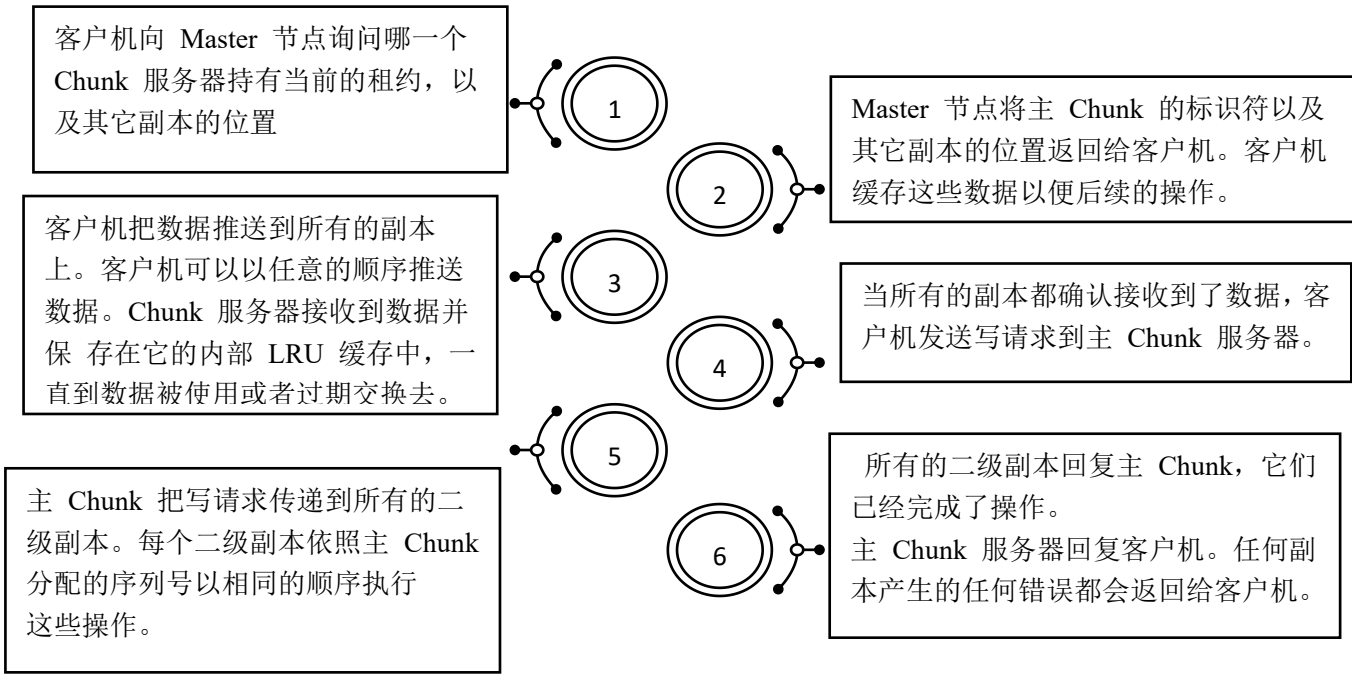
2.设计概述（架构）

由一个master和大量的chunkserver构成，并被许多客户（Client）访问。文件以 分层目录 的形式组织，用 路径名 来标识 。支持：创建新文件、删除文件、打开文件、关闭文件、读和写文件，以及快照和记录追加操作 。

3.系统交互

实现数据修改操作、原子的记录追加操作以及快照功能基础上最小化所有操作和 Master节点的交互。

租约和变更顺序 租约机制为了最小化Master节点的管理负担。变更是一个会改变 Chunk 内容或者元数据的操作。变更操作会在 Chunk的所有副本上执行。GFS使用租约（lease机制来保持多个副本间变更顺序的一致性。



写入操作的控制流程

数据流　为提高网络效率，采取把数据流和控制流分开的措施。控制流从客户机到主 Chunk、然后再到所有二级副本的同时，数据以管道的方式，顺序的沿着一个精心选择的 Chunk 服务器链推送。目标是充分利用每台机器的带宽，避免网络瓶颈和高延时的连接，最小化推送所有数据的延时。为了充分利用每台机器的带宽，数据沿着一个 Chunk 服务器链顺序的推送。为了尽可能的避免出现网络瓶颈和高延迟的链接，每台机器 都尽量的在网络拓扑中选择一台还没有接收到数据的、离自己最近的机器作为目标推送数据。利用基于 TCP 连接的、管道式数据推送方式来最小化延迟。

原子的记录追加 GFS 提供了一种原子的数据追加操作－记录追加，客户机只需要指定要写入的数据。GFS 保证至少有一次原子的写入操作成功执行写入的数据追加到 GFS 指定的偏移位置上，之后 GFS 返回这个偏移量给客户机。GFS 并不保证 Chunk 的所有副本在字节级别是完全一致的。它只保证数据作为一个整体原子的被至少写入一次。

快照 快照操作几乎可以瞬间完成对一个文件或者目录树（“源”）做一个拷贝，并且几乎不会对正在进行的其 它操作造成任何干扰。用标准的 GFS 用 copy-on-write 技术实现快照，当 Master 节点收到一个快照请求，它首先取消作快照的文件的所有 Chunk 的租约。这个措施保证了后续对这些 Chunk 的写操作都必须与 Master 交互以找到租约持有者。这就给 Master 节点一个率先创建 Chunk 的新拷贝的机会。租约取消或者过期之后，Master 节点把这个操作以日

志的方式记录到硬盘上。然后，Master 节点通过复制源文件或者目录的元数据的方式，把这条日志记录的变化反映到保存在内存的状态中。新创建的快照文件

和源文件指向完全相同的 Chunk 地址。。用户可以使用快照迅速的创建一个巨大的数据集的分支拷贝或者是在做实验性的数据操作之前，使用快照操作备份当前状态，这样之后就可以轻松的提交或者回滚到备份时的状态。

4. Master 节点的操作

Master 节点执行所有的名称空间操作。此外，它还管理着整个系统里所有 Chunk 的副本：它决定 Chunk 的存储位置，创建新 Chunk 和它的副本，协调各种各样的系统活动以保证 Chunk 被完全复制，在所有的 Chunk 服务器之间的进行负载均衡，以及回收不再使用的存储空间。

名称空间和锁，允许多个操作同时进行，使用名称空间的 region 上的锁来保证执行的正确顺序。读写锁采用惰性分配策略，在不再使用的时候立刻被删除。同样，锁的获取也要依据一个全局一致的顺序来避免死锁：首先按名称空间的层次排序，在同一个层次内按字典顺序排序。

副本的位置 Chunk 副本位置选择的策略服务两大目标：最大化数据可靠性和可用性，最大化网络带宽利用率。

创建，重新副本，重新负载均衡需要考虑的几个因素：平衡硬盘使用率、限制同一台 Chunk 服务器上进行的创建/复制的操作数量、在机架间分布副本

垃圾回收一开始只做日志标记，将文件隐藏，然后过几天再做删除。所有不能被 Master 节点识别的副本将被删除。垃圾回收会分散到各种例行操作里，同时会在 Master 节点相对空闲的时候完成。master 节点会保存副本的版本号确保访问数据的正确性。

5.容错和诊断

- 快速恢复（不区分正常关闭和异常关闭，在数秒钟内恢复它们的状态并重新启动）和 Chunk 复制（每个 Chunk 都被复制到不同机架上的不同的 Chunk 服务器上）保证系统高可用性。
- Master 复制，master 所有操作日志和快照文件都被复制到多台机器的硬盘中 。若 master 进程所在的机器或硬盘失效了，处于 GFS 系统外部的监控进程会在其他的存有完整操作日志的机器上启动一个新的 master 进程 。
- 数据完整性，每个 chunkserver 都独立维护 checksum 来检查保存的数据的完整性 。checksum 保存在内存和硬盘上，也记录在操作日志中。
- 对于读操作来说： 在把数据返回给 client 或者其它的 chunkserver 之前， chunkserver 会校验读取操作涉及的范围内的块的 checksum，若某个副本的 checksum 不正确，chunkserver 返回请求者一个错误消息，并通知 master 这个错误消息，作为回应，请求者从其他副本读取数据，master 服

务器从其他副本克隆数据进行恢复，新副本就绪后，master 通知 chunkserver 删掉错误的副 chunkserver 空闲的时候，也会扫描和校验不活动 chunk 的内容 。一旦发现数据损坏，master 创建新的正确的副本，且把损坏的副本删除掉 。

- 诊断工具，GFS 的服务器会产生大量的日志，记录了大量关键的事件（比如，Chunk 服务器启动和关闭）以及所有的 RPC 的请求和回复。且日志对性能的影响很小。

6.测量（度量）

一些具体的 GFS 集群的操作以及基准数据。

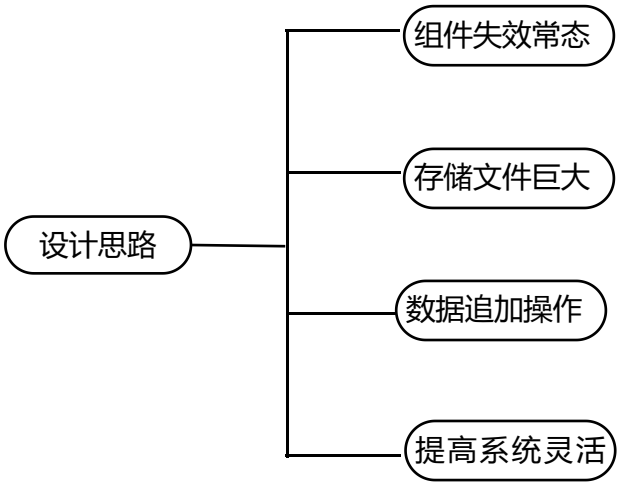
7.GFS 优缺点总结

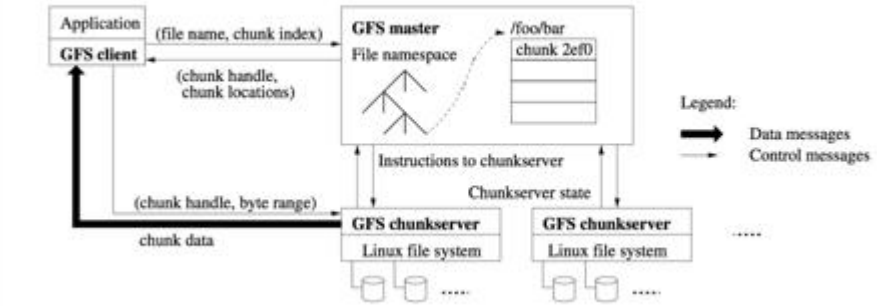
(1) 优点

- master 和 chunkserver 的设计，将文件管理和文件存储分离
- 将文件分割成 chunk 存储，可并发访问，吞吐量较大
- 修改数据时控制流和数据流分离，充分利用每台机器的带宽
- 使用 lease 降低 master 工作负载，防止 split-brain 问题
- 对文件追加和顺序读的功能有优化以及好的容错性

(2) 缺点

- 只有一个 master ，元数据过多的话可能内存不够用
- client 量很大的话，一个 master 负载过大
- master 不能出错自动重启，出故障后人工切换 master 比较耗时
- master 通过浏览所有的 chunk 进行垃圾回收效率太低
- 不擅长处理随机写问题、海量小文件存储
- 一致性过松，无法处理对一致性要求高的任务
- GFS 被设计用于运行在单个数据中心的系统


设计思路
组件失效常态
存储文件巨大
数据追加操作
提高系统灵活


Application
GFS client
(file name, chunk index)
(chunk handle,
chunk locations)
GFS master
File namespace
/foo/bar
chunk 2ef0
Instructions to chunkserver
Chunkserver state
(chunk handle, byte range)
chunk data
GFS chunkserver
Linux file system
GFS chunkserver
Linux file system
Legend:
Data messages
Control messages

图一

MapReduce 论文总结

1. **前言**

大多数分布式运算可以抽象为 MapReduce 操作。Map 是把输入 Input 分解成中间的 Key/Value 对，Reduce 把 Key/Value 合成最终输出 Output。这两个函数由程序员提供给系统，下层设施把 Map 和 Reduce 操作分布在集群上运行，并把结果存储在 GFS 上。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，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 库易于使用且大量不同类型的问题都可以通过 MapReduce 简单的解决。在数千台计算机组成的大型集群上灵活部署运行的 MapReduce 使得有效利用其上丰富的计算资源变得简单。解决海量数据计算！！！

2. **编程模型** MapReduce 库的用户用两个函数 ：Map 和 Reduce.。MapReduce 编程模型的原理是:利用一个输入 key/value pair 集合来产生一个输出的 key/value pair 集合.

3.实现 用以太网交换机连接、由普通 PC 机组 成的大型集群。

执行流程 用户程序需要调用 MapReduce 库将输入文件分割成 M 个数据片段。然后在机器集群中创建大量程序副本。其中有个特殊副本为 master, 其他副本用于执行任务。

然后 master 将 M 个 Map 任务和 R 个 Reduce 分配到一个空闲的 worker 上。

被分配的 Map 任务(如编号 m)开始读取 m 号的数据片段，解析对应的<key, value="">然后调用用户自定义的 map 函数生成<key, value="">并缓存到内存中。

缓存的<key, list(value)="">根据 hash(key) mod R 运算分到 R 个区域写入本地磁盘上。master 负责将这些存储位置传送给 Reduce worker

Reduce worker 读取 Map worker 存储的数据首先对 key 进行排序将相同的 key 值进行聚合。<key, set(value)="">

Reduce worker 遍历排序后的中间数据将<key, set(value)="">传递给用户自定义的 Reduce 函数，将结果追加到输出。

所有 Map 和 Reduce 任务完成后, 则 master 唤醒用户任务执行完成。

MapReduce 的容错机制：针对 worker 故障，master 周期性的 ping 每个 worker，当 worker A 失效时,则调度 worker B 执行 A 的任务;针对 master 故障,让 master

周期性将 checkpoint 写入磁盘，master 失效后，从最后一个 checkpoint 启动另一个 master 进程。

4.技巧

扩展功能 分区函数、顺序保证、Combiner 函数、输入和输出类型、副作用、跳过损坏的记录、本地执行、状态信息、计数器。

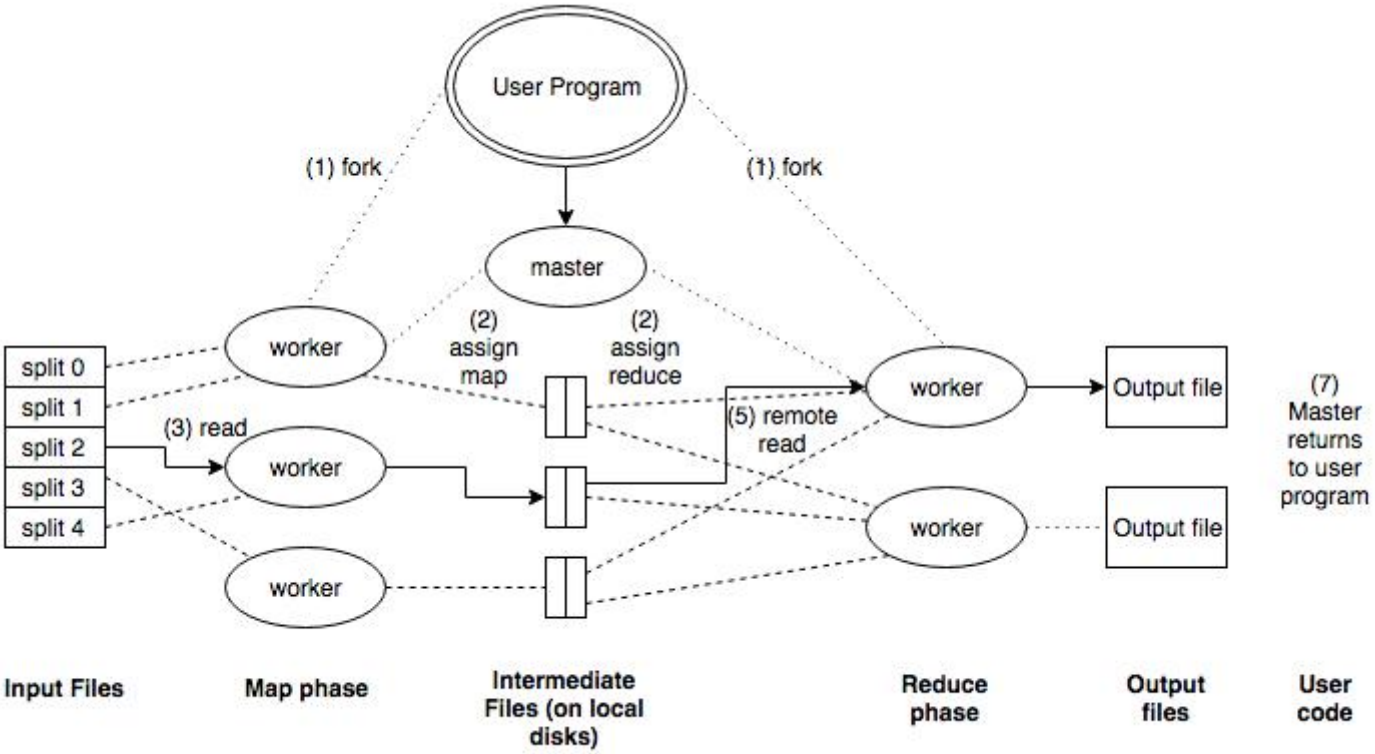
5.性能

在一个大型集群上运行的两个计算来衡量 MapReduce 的性能

6.总结

MapReduce 的一些优化设计：（1）输入数据本地化优化策略：考虑到网络带宽相对匮乏，尽量把输入数据存储在集群中机器的本地磁盘，将 map 任务调度在数据包含在本地磁盘的机器上执行，从而减少网络带宽的消耗；（2）把 Map 拆分成了 M 个片段、把 Reduce 拆分成 R 个片段执行。理想情况下，M 和 R 应当比集群中 worker 的机器数量要多得多，让每个 worker 处理大量任务，有利于实现负载均衡；（3）影响一个 MapReduce 的总执行时间最通常的因素是“落伍者”，应对措施：当一个 MapReduce 操作接近完成的时候，master 调度备用（backup）任务进程来执行剩下的、处于处理中状态（in-progress）的任务。无论是最初的执行进程、还是备用（backup）任务进程完成了任务，都把这个任务标记成为已经完成。

MapReduce 的应用大规模机器学习问题、Google Newe 和 Froogle 产品的集群问题、从公众查询产品（比如 Google 的 Zeitgeist）的报告中抽取数据、从大量的新应用和新产品的网页中提取有用信息（比如，从大量的位置搜索网页中抽取地理位置信息）、大规模的图形计算、大规模的索引。


User Program
(1) fork
(1) fork
master
(2)
assign
map
(2)
assign
reduce
worker
worker
worker
worker
worker
split 0
split 1
split 2
split 3
split 4
(3) read
(5) remote
read
Output file
Output file
(7)
Master
returns
to user
program
Input Files
Map phase
Intermediate
Files (on local
disks)
Reduce
phase
Output
files
User
code

执行流程

BigTable 论文总结

BigTable 是一个大型的分布式数据库，这个数据库不是关系式的数据库。像它的名字一样，就是一个巨大的表格，用来存储结构化的数据。存储实时在线应用的海量结构化数据！！！
整篇论文包括数据模型的细节介绍，写了 API 客户端，写了 bigtable 的主要构件——使用 GFS 来存储日志文件和数据文件，内部存储数据的文件是 Google SSTable 格式的，它还依赖一个高可用的、序列化的分布式锁服务组件叫 Chubby.另外还对 bigtable 进行了介绍，它包括三个主要的组件:链接到客户程序中的库、Master 服务器和多个 Tableht.最后一部分有关了 bigtable 的实际应用。
Bigtable 中的行关键字可以是任意的字符串，并且每行的读写操作都是原子的；Bigtable 中的行关键字是按照字典顺序排序存储的，表中的行都可以进行动态分区，每个分区叫 tablet，tablet 是数据分布和负载均衡的最小单位。由于行键是按照字典序存储的，所以查询时以行关键字作为条件查询速度毫秒级。

列关键字组成的集合叫做“列族”，列族时访问控制的基本单位。列关键字的命名语法如下：列族：限定词。

每行数据都会有用来当作版本号的时间戳，可以系统自动赋值，也可以用户自己指定。最新的数据行排在最前面。还可以利用时间戳来进行垃圾收集。

Bigtable 使用 Google 的分布式文件系统 GFS 存储日志文件和数据文件。
BigTable 内部存储数据的问文件时 Google SSTable 格式的。SSTable 是一个持久化的、排序的、不可更改的 Map<key,value>结构，其值都是任意的 byte 串，因此使用 key 查询速度很快。Big Table 还依赖一个高可用的、序列化的分布式锁服务组件——Chubby。BigTable 使用 Chubby 完成以下几个任务：

确保在任何时间内最多只有一个活动的 Master 副本；
存储 BigTable 数据的自引导指令的位置
查找 Tablet 服务器，以及在 Tablet 服务器失效时进行善后；
存储 BigTable 模式信息
存储访问控制列表。

BigTable 包括了三个主要的组件：链接到客户程序的库、一个 Mater 服务器和多个 Tablet。针对系统工作负载的变化情况，BigTable 可以动态的向集群添加或者删除 Tablet 服务器。

Master 服务器主要为 Tablet 服务器分配 Tablets、检测新加入的或者过期失效的 Tablet 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，还处理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，每个 Tablet 的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。

客户端读取的数据都不经过 Master 服务器；客户程序直接和 Tablet 服务器通信进行读写操作。

在任何一个时刻，一个 Tablet 只能分配给一个 Tablet 服务器。Master 服务器记录了当前有那些活跃的 Tablet 服务器、那些 Tablet 分配给了那些 Tablet 服务器、那些 Tablet 还没有被分配。

BigTable 使用 Chubby 跟踪记录 Tablet 服务器的状态。当一个 Tablet 服务器启动时，它在 Chubby 的一个指定目录下建立一个有唯一性名字的文件，并且获取该文件的独占锁。Master 服务器实时监控着这目录，因此 Master 服务能够知道有新的 Tablet 服务器加入了。只要文件存在 Tablet 服务器就会试图重新获得对该文件的独占锁，如果文件不存在了，那么
Tablet 服务器就不能在提供服务了。

Master 服务器从 Chubby 获取一个唯一的 Master 锁，用来阻止创建其它的 Master 服务器实例；

Master 服务器扫描 Chubby 的服务器文件锁存储目录，获取当前正在运行的服务器列表；

Master 服务器和所有的正在运行的 Tablet 表服务器通信，获取每个 Tablet 服务器上 Tablet 的分配信息；

Master 服务器扫描 METADATA 表获取所有的 Tablet 的集合。

在扫描的过程中，当 Master 服务器发现了一个还没有分配的 Tablet，Master 服务器就将这个 Tablet 加入

未分配的 Tablet 集合等待合适的时机分配。

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。这些更新操作中，最近提交的那些放在一个排序的缓存中，我们称这个缓存为 memtable；较早更新存放在一系列的 SSTable 中。

随着写操作的执行，memtable 的大小不断增加。当 memtable 的尺寸到达一个门限值的时候，这个 memtable 就会被冻结，然后创建一个新的 memtable；被冻结住的 memtable 会被转换成 SSTable，然后写入 GFS。

客户程序可以将多个列族组合成一个局部性群族。对 Tablet 中的每个局部性群族都生成一个单独的 SSTable。将同城不会一起访问的列族分割成不同的局部性群族可以提高读取操作的效率

客户程序可以控制一个局部性群族的 SSTable 是否需要压缩，一般使用两遍的、可定制的压缩

为了提高读操作的性能，Tablet 服务器使用二级缓存的策略，一级用来缓存 Tablet 服务器通过 SSTable 接口的 Key-Value 对；Block 是二级缓存，用来缓存从 GFS 读取的 SSTable 的 Block。

BigTable 将数据存在一个三维有序的表中，这个表除了传统二维表的 row，column 以外还增加了第三维 TimeStamp，用来表示版本。这样 rowid，colume family 和 timestamp 就构成了一个三维有序的大表，数据就存储在这张大表。从上层来看的话，一个数据表就是一个三维有序的表的样子，而在底层来说，这个大表的实现方式则比较巧妙。每个大表被切分成若干个部分称为 tablet，各个 tablet 分布在各个不同的 tablet 服务器上，这些 tablet 服务器都包含了缓存，日志和持久存储（这里的持久存储是将数据存储到 GFS(Google File System)上去）。在每个服务器上缓存，日志和持久存储相互协作，最大程度的保证了数据的存取性能和安全性能。tablet 服务器之间的负载均衡是通过合并与切分 tablet 来动态实现的，保证了服务器的高效利用。

当然为了提高性能和提高安全性，BigTable 有一些其他机制。除了 Tablet 服务器以外还有 Root 服务器和 Meta 服务器，从 Root 服务器到 Meta 服务器再到 Tablet 服务器使用了 B+树的数据结构，使得 PV 性能提高。在 Tablet 服务器之间存在一个 Mater 服务器用于统筹管理所有服务器的状态。BigTable 还与 Chubby 紧密联系，添加了 BigTable 的安全性能。