云 计 算 (第三版)

CLOUD COMPUTING Third Edition

第 2 章

Google云计算原理与应用（二）

目录

初步了解Chubby

Chubby是Google设计的提供粗粒度锁服务的一个文件系统，它基于松耦合分布式系统，解决了分布的一致性问题。

通过使用Chubby的**锁服务**，用户可以确保数据操作过程中的一致性

Chubby作为一个稳定的**存储系统**存储包括元数据在内的小数据

Google内部还使用Chubby进行**名字服务**（Name Server）

2.3 分布式锁服务Chubby

- 2.3.1 Paxos算法
- 2.3.2 Chubby系统设计
- 2.3.3 Chubby中的Paxos
- 2.3.4 Chubby文件系统
- 2.3.5 通信协议
- 2.3.6 正确性与性能

- Paxos算法


三个节点
proposers
提出决议
acceptors
批准决议
learners
获取并使用已经通过的决议
三个条件
决议只有在被proposers提出后才能批准
每次只批准一个决议
只有决议确定被批准后learners才能获取这个决议

- **系统的约束条件**

p1：每个acceptor只接受它得到的第一个决议。

p2：一旦某个决议得到通过，之后通过的决议必须和该决议保持一致。

p2a：一旦某个决议v得到通过，之后任何acceptor再批准的决议必须是v。

p2b：一旦某个决议v得到通过，之后任何proposer再提出的决议必须是v。

p2c：如果一个编号为n的提案具有值v，那么存在一个“多数派”，要么它们中没有谁批准过编号小于n的任何提案，要么它们进行的最近一次批准具有值v。

为了保证决议的唯一性，acceptors也要满足一个约束条件：当且仅当 acceptors 没有收到编号大于n的请求时，acceptors 才批准编号为n的提案。

- 一个决议分为两个阶段

1 准备阶段

proposers选择一个提案并将它的编号设为n

将它发送给acceptors中的一个“多数派”

acceptors 收到后，如果提案的编号大于它已经回复的所有消息，则acceptors将自己上次的批准回复给proposers，并不再批准小于n的提案。

2 批准阶段

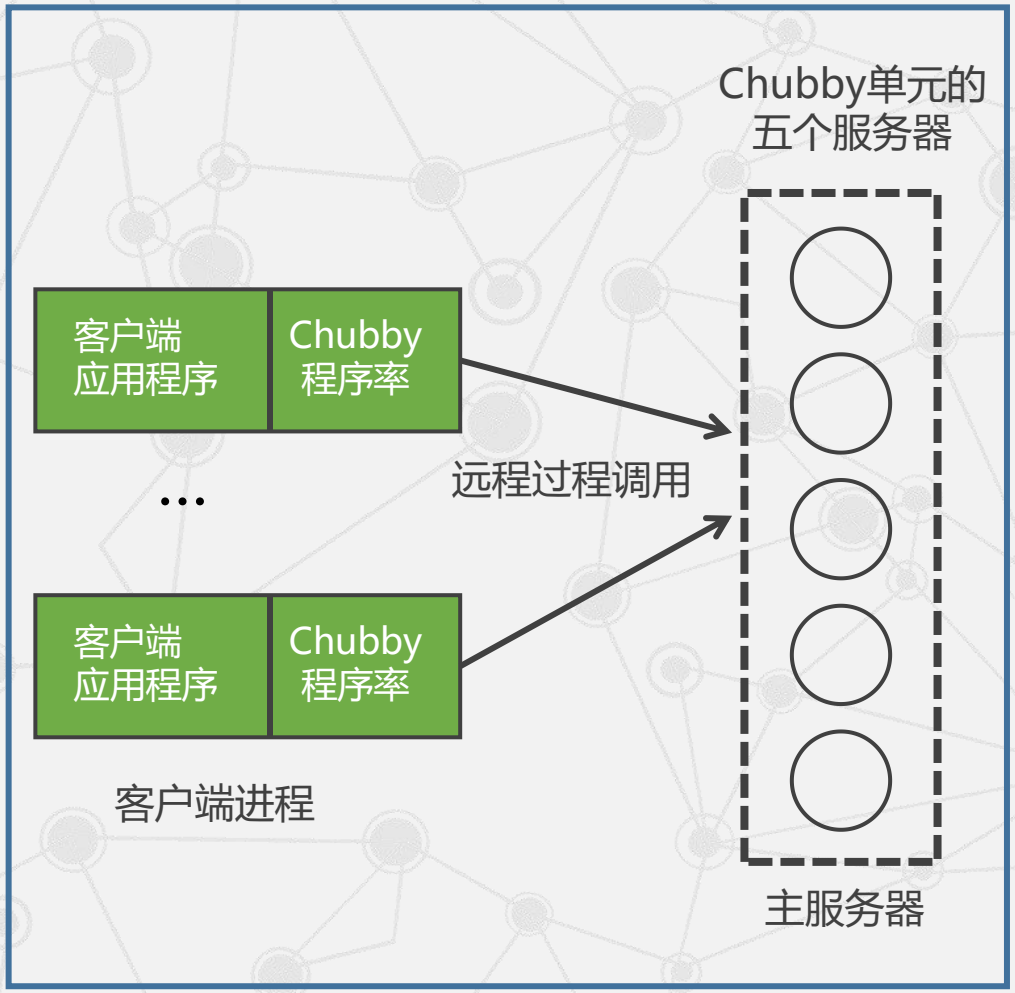
当proposers接收到acceptors 中的这个“多数派”的回复后，就向回复请求的acceptors发送accept请求，在符合acceptors一方的约束条件下，acceptors收到accept请求后即批准这个请求。

2.3 分布式锁服务Chubby

- Chubby的设计目标主要有以下几点

1	高可用性和高可靠性	4	服务信息的直接存储
2	高扩展性	5	支持缓存机制
3	支持粗粒度的建议性锁服务	6	支持通报机制

- Chubby的基本架构



客户端

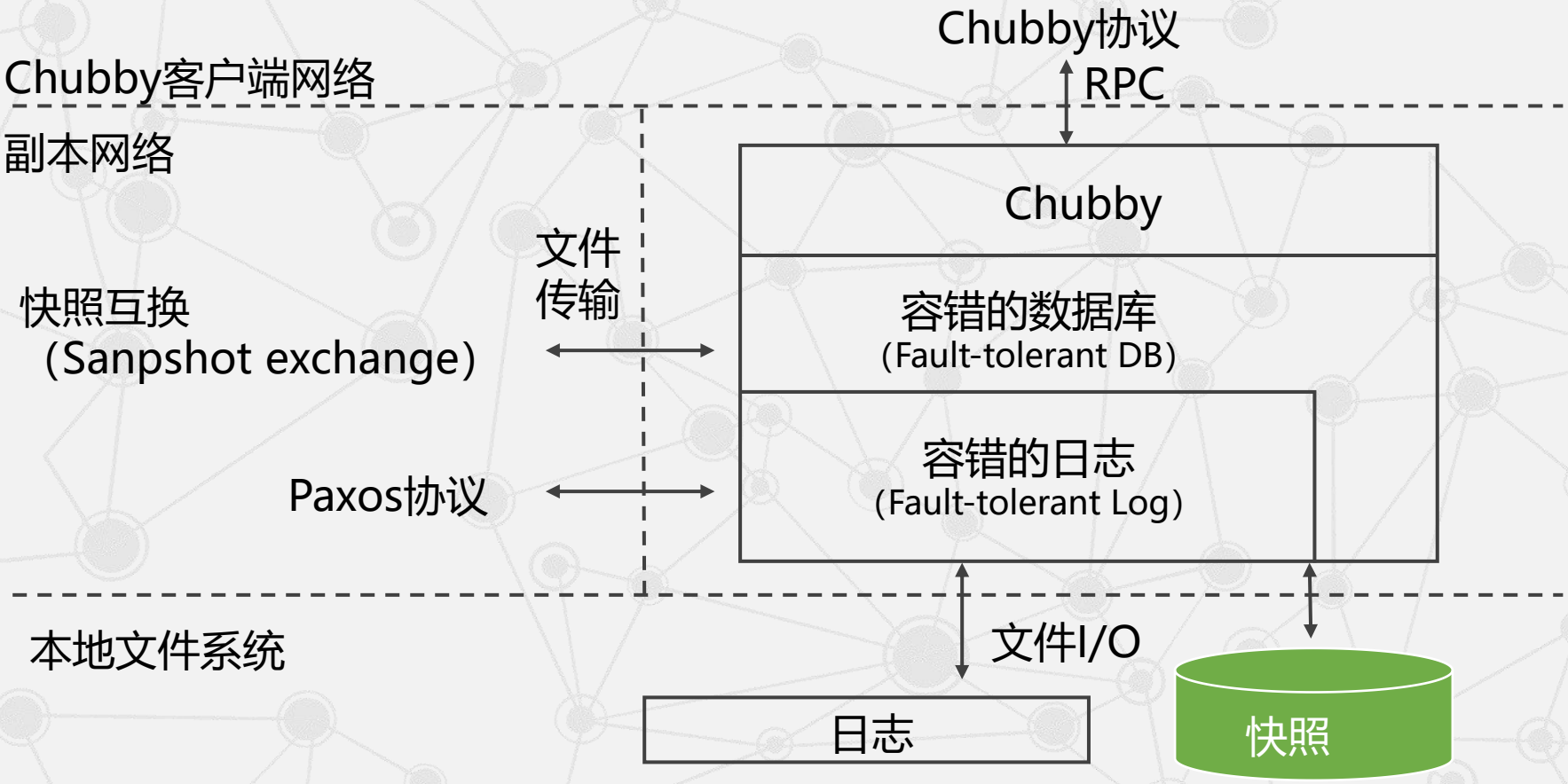
在客户这一端每个客户应用程序都有一个Chubby程序库（Chubby Library），客户端的所有应用都是通过调用这个库中的相关函数来完成的。

服务器端

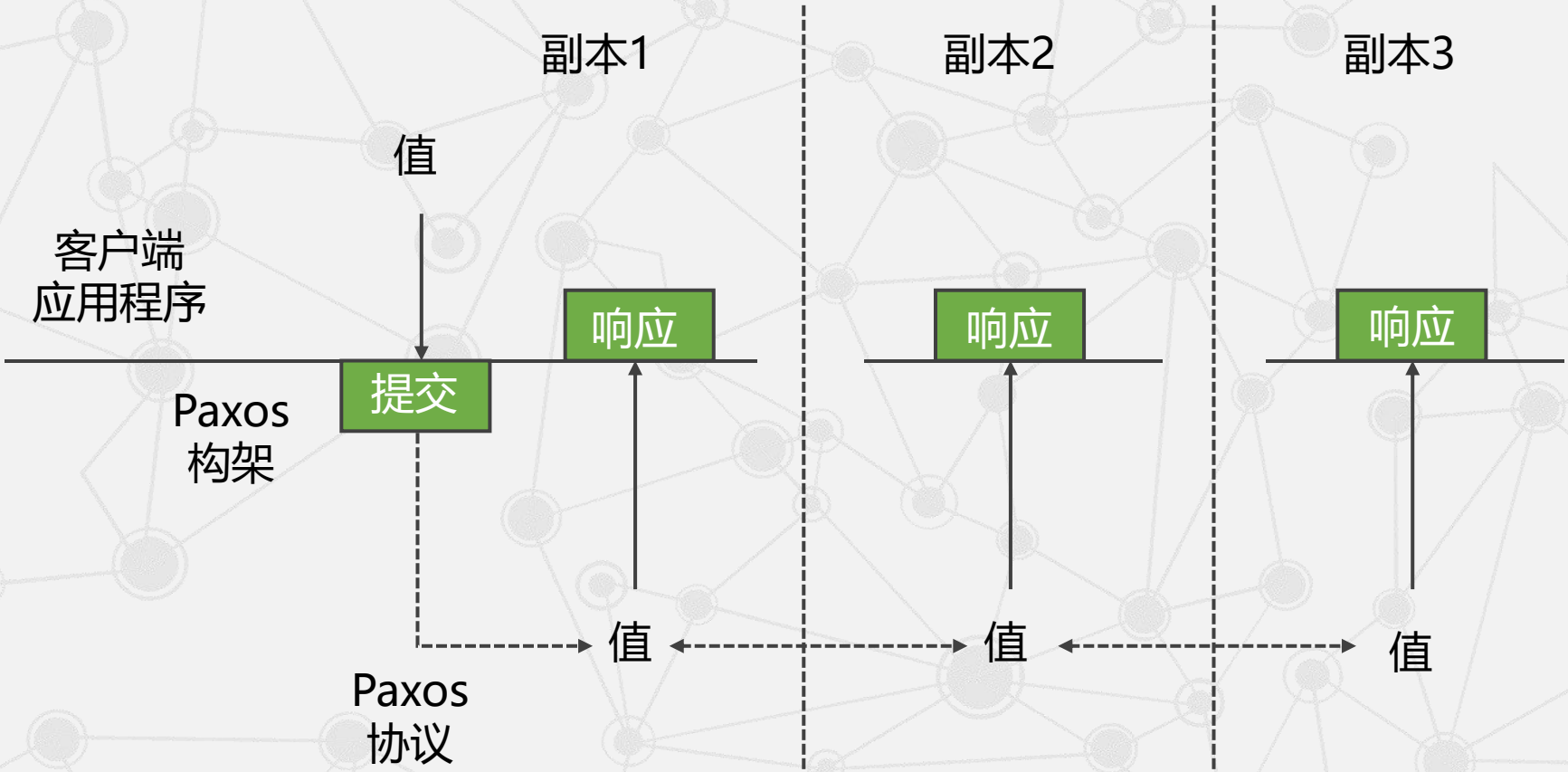
服务器一端称为Chubby单元，一般是由五个称为副本（Replica）的服务器组成的，这五个副本在配置上完全一致，并且在系统刚开始时处于对等地位。

2.3 分布式锁服务Chubby

- 单个Chubby副本结构



- 容错日志的API



2.3 分布式锁服务Chubby

- **单调递增的64位编号**

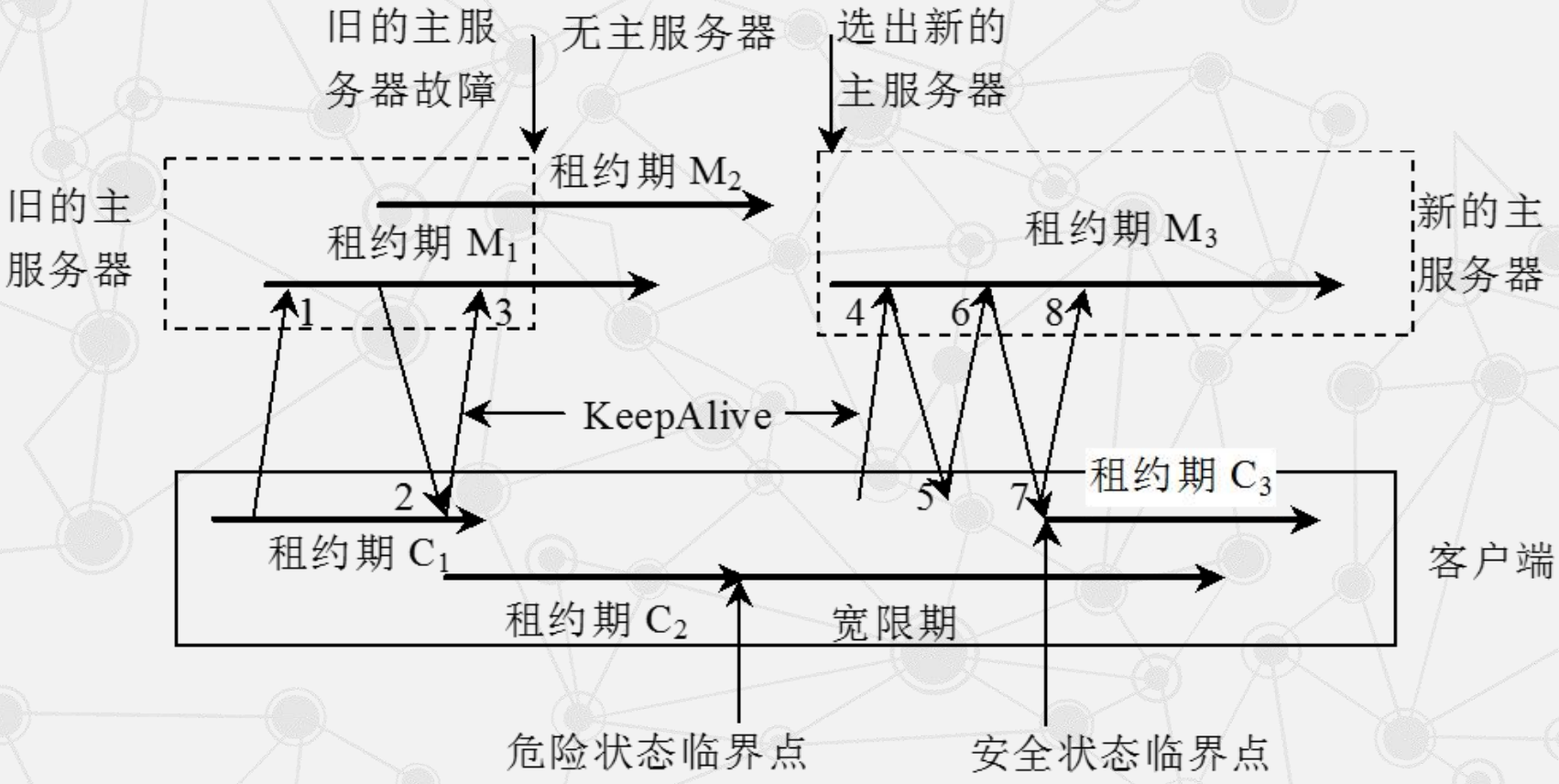
❶ 实例号 Instance Number	新节点实例号必定大于旧节点的实例号。
❷ 内容生成号 Content Generation Number	文件内容修改时该号增加。
❸ 锁生成号 Lock Generation Number	锁被用户持有时该号增加。
❹ ACL生成号 ACL Generation Number	ACL名被覆写时该号增加。

- 常用的句柄函数及作用

函数名称	作　用
Open()	打开某个文件或者目录来创建句柄
Close()	关闭打开的句柄，后续的任何操作都将中止
Poison()	中止当前未完成及后续的操作，但不关闭句柄
GetContentsAndStat()	返回文件内容及元数据
GetStat()	只返回文件元数据
ReadDir()	返回子目录名称及其元数据
SetContents()	向文件中写入内容
SetACL()	设置ACL名称
Delete()	如果该节点没有子节点的话则执行删除操作
Acquire()	获取锁
Release()	释放锁
GetSequencer()	返回一个sequencer
SetSequencer()	将sequencer和某个句柄进行关联
CheckSequencer()	检查某个sequencer是否有效

2.3 分布式锁服务Chubby

- Chubby客户端与服务器端的通信过程



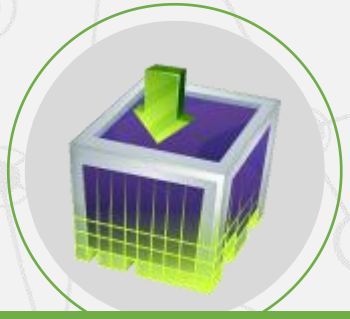
- 可能出现的两种故障

1 客户端租约过期

2 主服务器出错

2.3 分布式锁服务Chubby

- 正确性与性能



一致性

每个Chubby单元是由五个副本组成的，这五个副本中需要选举产生一个主服务器，这种选举本质上就是一个一致性问题

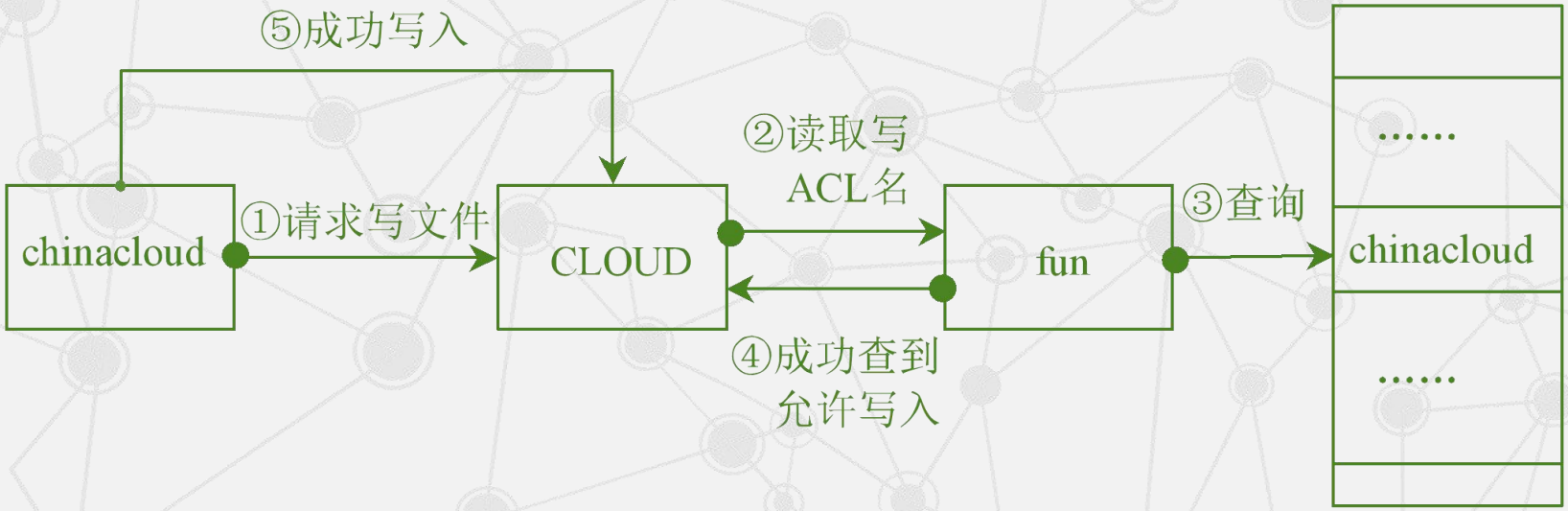
安全性

采用的是ACL形式的安全保障措施。只要不被覆写，子节点都是直接继承父节点的ACL名

性能优化

提高主服务器默认的租约期、使用协议转换服务将Chubby协议转换成较简单的协议、客户端一致性缓存等

- Chubby 的 ACL 机制



用户chinacloud提出向文件CLOUD中写入内容的请求。CLOUD首先读取自身的写ACL名fun，接着在fun中查到了chinacloud这一行记录，于是返回信息允许chinacloud对文件进行写操作，此时chinacloud才被允许向CLOUD写入内容。其他的操作和写操作类似。

目录

2.4 分布式结构化数据表Bigtable

- 2.4.1 设计动机与目标
- 2.4.2 数据模型
- 2.4.3 系统架构
- 2.4.4 主服务器
- 2.4.5 子表服务器
- 2.4.6 性能优化

- Bigtable 的设计动机

Google运行着目前世界上最繁忙的系统，它每时每刻处理的客户服务请求数量是普通的系统根本无法承受的

包括URL、网页内容、用户的个性化设置在内的数据都是Google需要经常处理的

一方面现有商用数据库的设计着眼点在于其通用性。另一方面对于底层系统的完全掌控会给后期的系统维护、升级带来极大的便利

Bigtable 应达到的基本目标

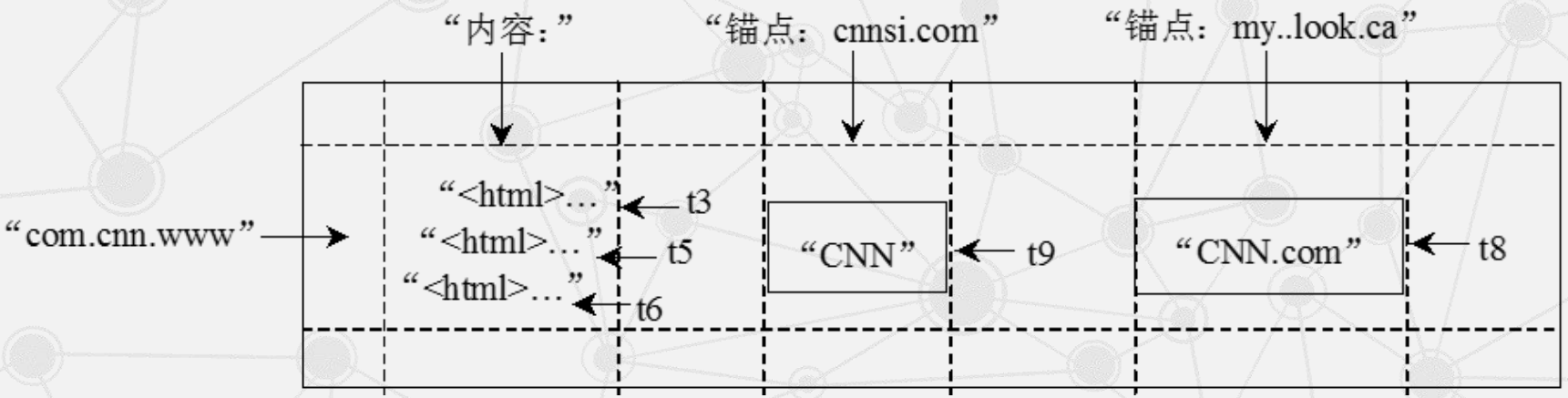
广泛的适用性	Bigtable是为了满足一系列Google产品而并非特定产品的存储要求。
很强的可扩展性	根据需要随时可以加入或撤销服务器
高可用性	确保几乎所有的情况下系统都可用
简单性	底层系统的简单性既可以减少系统出错的概率，也为上层应用的开发带来便利

2.4 分布式结构化数据表Bigtable

- **Bigtable数据的存储格式**

Bigtable是一个分布式多维映射表，表中的数据通过**一个行关键字（Row Key）、一个列关键字（Column Key）以及一个时间戳（Time Stamp）进行索引**

Bigtable的存储逻辑可以表示为：
(row:string, column:string, time:int64)→string



行

- Bigtable的行关键字可以是任意的字符串，但是大小不能够超过64KB
- 表中数据都是根据行关键字进行排序的，排序使用的是词典序
- 同一地址域的网页会被存储在表中的连续位置
- 倒排便于数据压缩，可以大幅提高压缩率

列

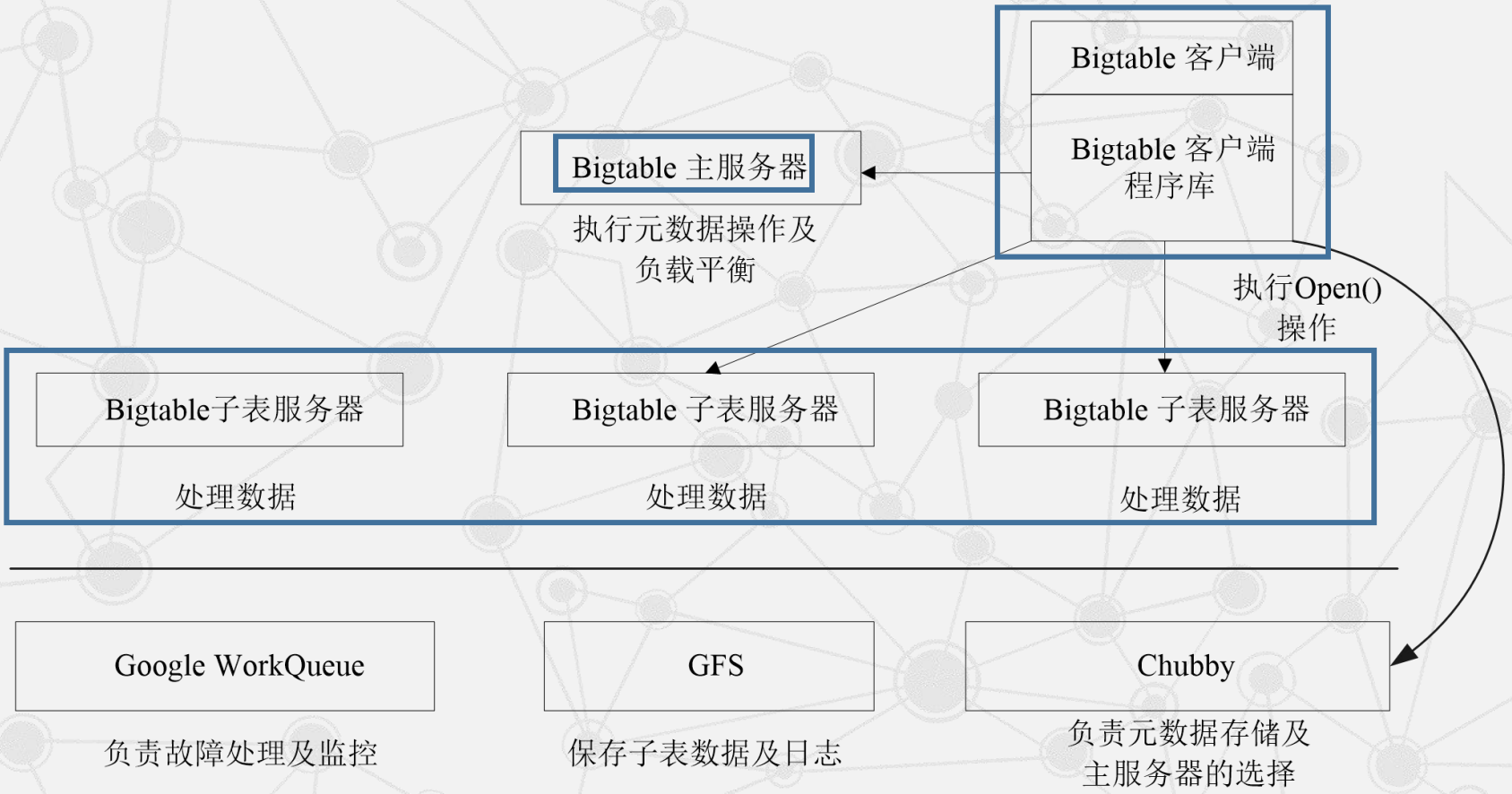
- 将其组织成所谓的列族（Column Family）
- 族名必须有意义，限定词则可以任意选定
- 组织的数据结构清晰明了，含义也很清楚
- 族同时也是Bigtable中访问控制（Access Control）的基本单元

时间戳

- Google的很多服务比如网页检索和用户的个性化设置等都需要保存不同时间的数据，这些不同的数据版本必须通过时间戳来区分。
- Bigtable中的时间戳是64位整型数，具体的赋值方式可以用户自行定义

2.4 分布式结构化数据表Bigtable

- Bigtable 基本架构



- Bigtable 中 Chubby 的主要作用

作用一	选取并保证同一时间内只有一个主服务器（Master Server）。
作用二	获取子表的位置信息。
作用三	保存Bigtable的模式信息及访问控制列表。

2.4 分布式结构化数据表Bigtable

- 当一个新的子表产生时，主服务器通过一个加载命令将其分配给一个空间足够的子表服务器。
- 创建新表、表合并以及较大子表的分裂都会产生一个或多个新子表。
- 分割完成之后子服务器需要向主服务发出一个通知。
- 主服务器必须对子表服务器的状态进行监控，以便及时检测到服务器的加入或撤销

Bigtable 中 Chubby 的主要作用

步骤 1：从Chubby中获取一个独占锁，确保同一时间只有一个主服务器

步骤 2：扫描服务器目录，发现目前活跃的子表服务器

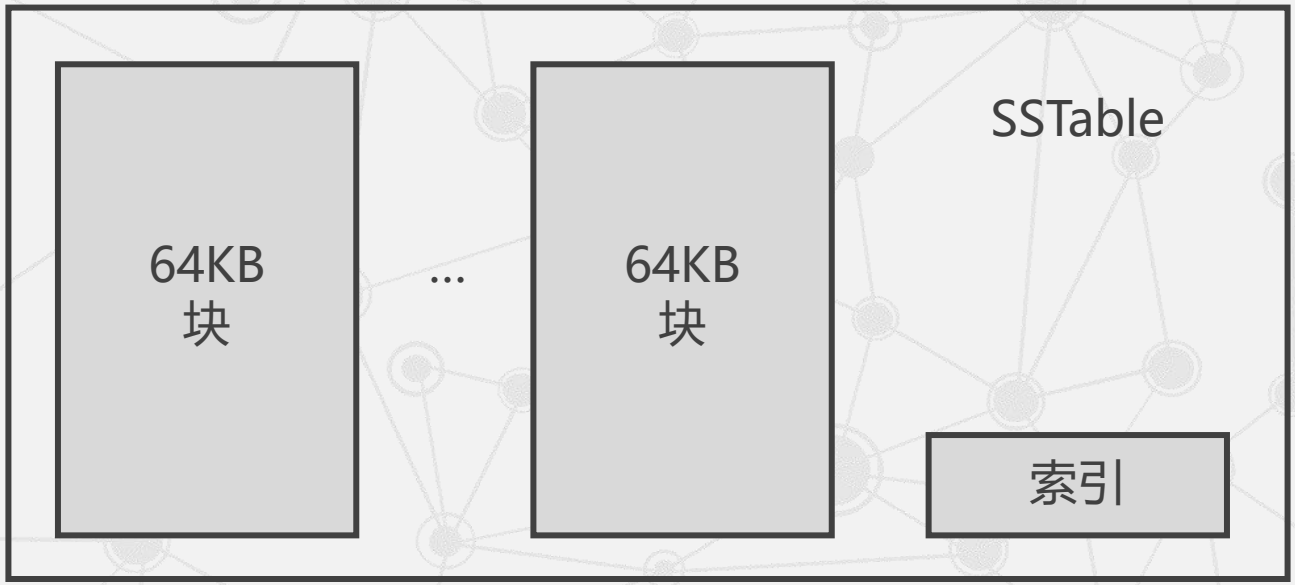
步骤 3：与所有的活跃子表服务器取得联系以便了解所有子表的分配情况

步骤 4：通过扫描元数据表（Metadata Table），发现未分配的子表并将其分配到合适的子表服务器

2.4 分布式结构化数据表Bigtable

- **SSTable 格式的基本示意**

SSTable是Google为Bigtable设计的内部数据存储格式。所有的SSTable文件都存储在GFS上，用户可以通过键来查询相应的值。



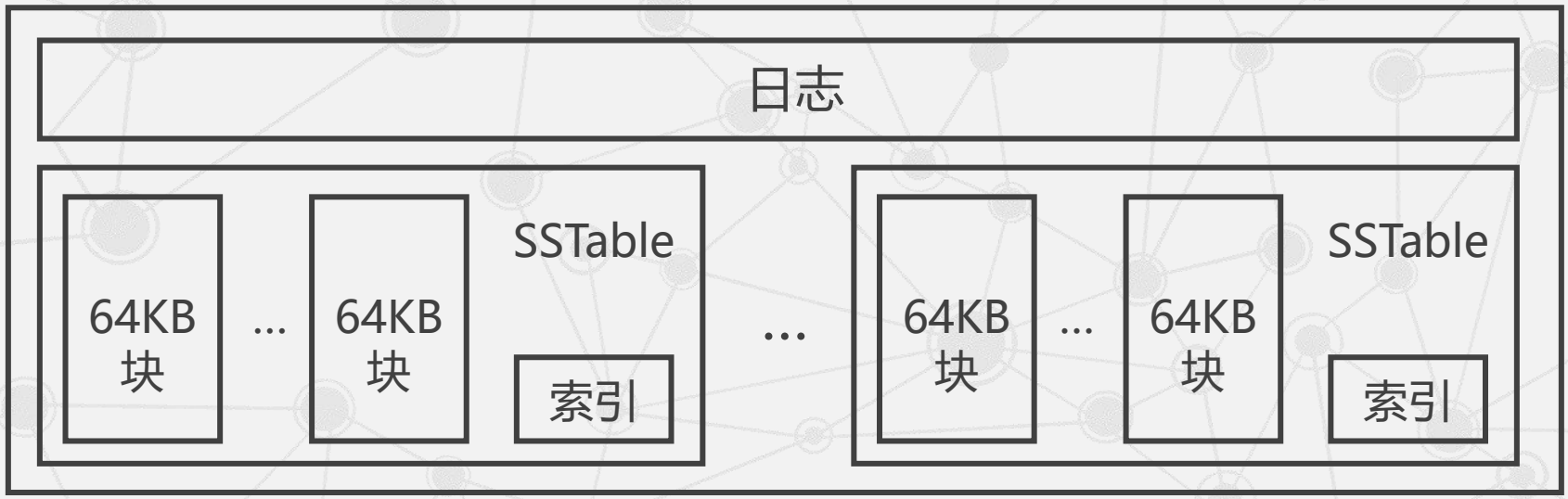
- **子表实际组成**

不同子表的SSTable可以共享

每个子表服务器上仅保存一个日志文件

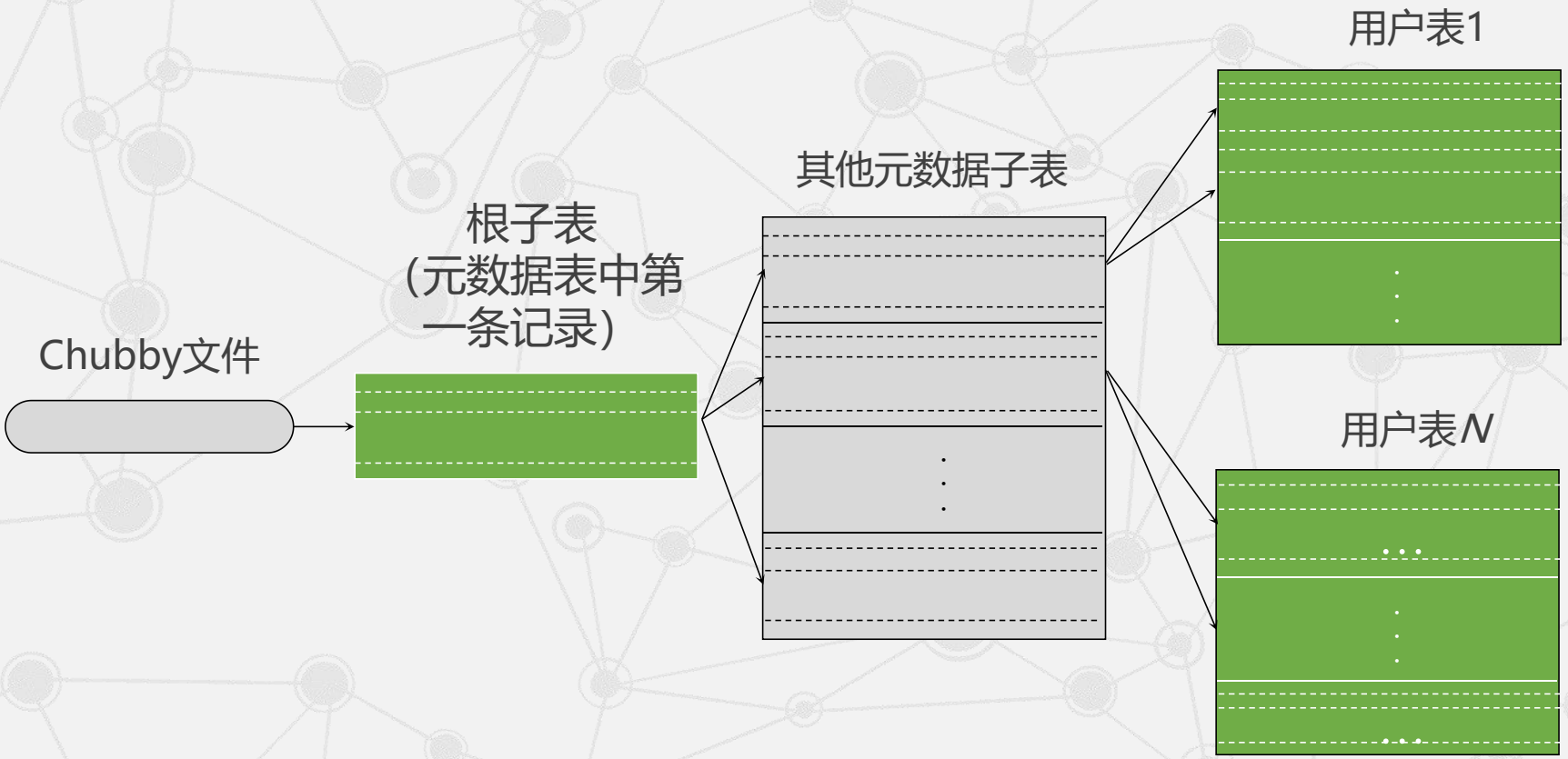
Bigtable规定将日志的内容按照键值进行排序

每个子表服务器上保存的子表数量可以从几十到上千不等，通常情况下是100个左右



- **子表地址组成**

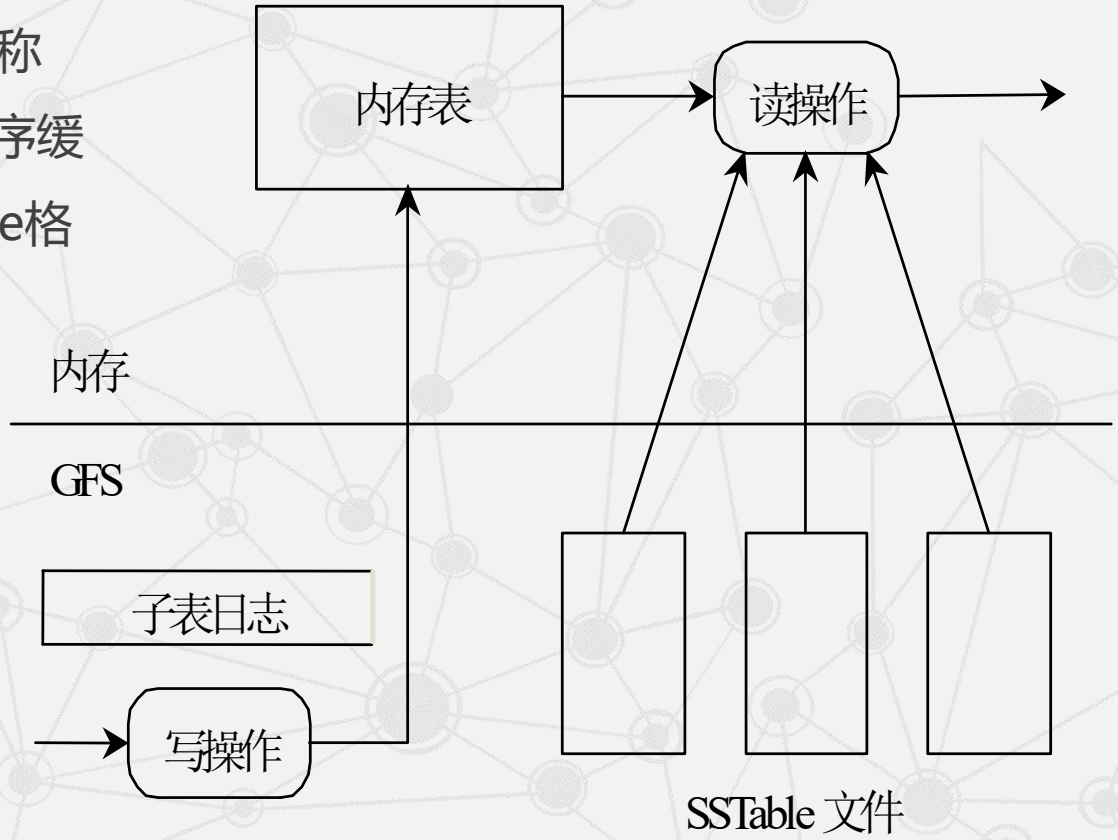
Bigtable系统的内部采用的是一种类似B+树的三层查询体系



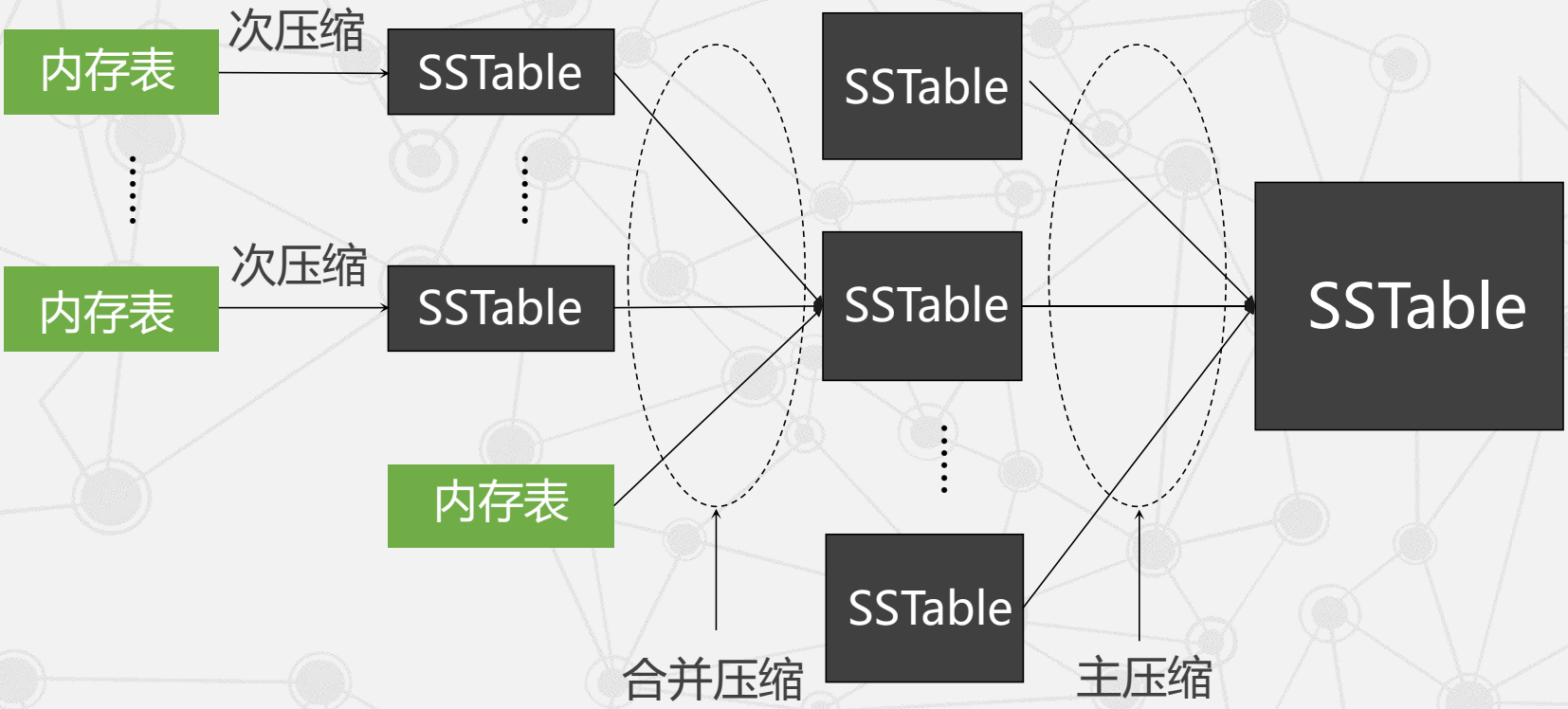
- **Bigtable 数据存储及读/写操作**

较新的数据存储在内存中一个称为内存表（Memtable）的有序缓冲里，较早的数据则以SSTable格式保存在GFS中。

读和写操作有很大的差异性



- 三种形式压缩之间的关系

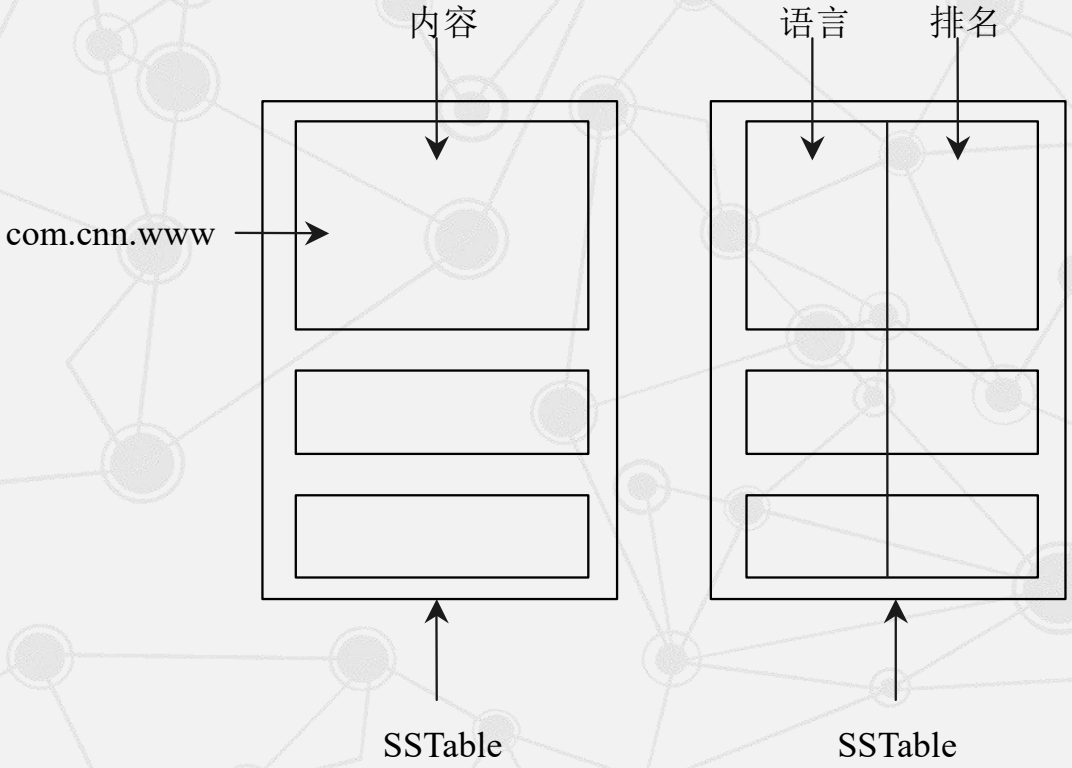


2.4 分布式结构化数据表Bigtable

- 2.4.1 设计动机与目标
- 2.4.2 数据模型
- 2.4.3 系统架构
- 2.4.4 主服务器
- 2.4.5 子表服务器
- ▶ 2.4.6 性能优化

- **局部性群组**

Bigtable允许用户将原本并不存储在一起的数据以列族为单位，根据需要组织在一个单独的SSTable中，以构成一个局部性群组。



用户可以只看自己感兴趣的内容。

对于一些较小的且会被经常读取的局部性群组，明显地改善读取效率。

- **压缩**

压缩可以有效地节省空间，Bigtable中的压缩被应用于很多场合。首先压缩可以被用在构成局部性群组的SSTable中，可以选择是否对个人的局部性群组的SSTable进行压缩。

1. 利用Bentley & McIlroy方式（BMDiff）在大的扫描窗口将常见的长串进行压缩

2. 采取Zippy技术进行快速压缩，它在一个16KB大小的扫描窗口内寻找重复数据，这个过程非常快

布隆过滤器

Bigtable向用户提供了一种称为**布隆过滤器**的数学工具。布隆过滤器是巴顿•布隆在**1970**年提出的，实际上它是一个很长的**二进制向量**和一系列**随机映射函数**，在读操作中确定子表的位置时非常有用。

优点

- 布隆过滤器的速度快，省空间
- 不会将一个存在的子表判定为不存在

缺点

- 在某些情况下它会将不存在的子表判断为存在

本章未完待续

云计算 (第三版)

CLOUD COMPUTING Third Edition

第 2 章

谢 谢 观 看