云计算（第三版）

CLOUD COMPUTING Third Edition

第 2 章

Google云计算原理与应用（三）

主编：刘鹏 教授

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

可用性：用于描述系统能够正确工作的时间，指的是系统能够在给定的时间内确保可以运行的能力。
可用性 = 正常运行时间（up time）/ 总时间（total time）

可靠性：描述系统是否能够正确工作，或者说系统是否具有提供持续正确服务的能力，也即软件或硬件是否能够按照其规范持续提供服务。
常用故障率（FIT）或平均失效间隔时间（MTBF）来度量。1FIT等于每十亿（Billion）小时内产生一个错误，1MTBF大约等于11.4万年。

- Chubby的设计目标主要有以下几点

1. 高可用性和高可靠性

建议性的锁不会阻止访问，而强制性的锁则会阻止访问。
方便系统组件之间的信息交互

2. 高扩展性

3. 支持粗粒度的建议性锁服务

细粒度的锁持有时间很短，常常只有几秒甚至更少，而粗粒度的锁持有的时间可长达几天，选择粗粒度的锁可以减少频繁换锁带来的系统开销，提高系统的**性能**

- Chubby的设计目标主要有以下几点

1. 高可用性和高可靠性
2. 高扩展性
3. 支持粗粒度的建议性锁服务
4. 服务信息的直接存储

可以直接存储包括元数据、 系统参数在内的有关服务信息， 而不需要再维护另一个服务

- Chubby的设计目标主要有以下几点

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

通过一致性缓存将常用信息保存在客户端， 避免频繁地访问主服务器

- 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

客户可以及时地了解到事件的发生

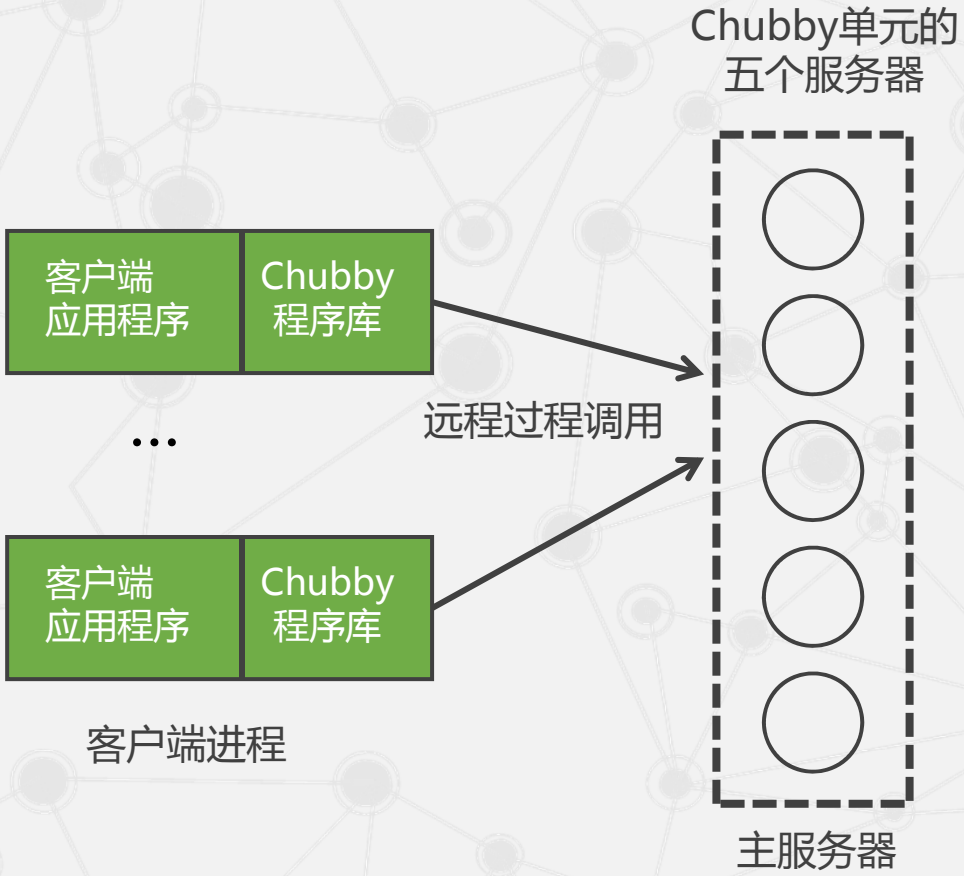
- **Google没有直接实现一个包含了Paxos算法的函数库，而是在Paxos算法的基础上设计了一个全新的锁服务Chubby**

（1）通常情况下开发者在开发的初期很少考虑系统的一致性问题，但是随着开发的不断进行，这种问题会变得越来越严重。单独的锁服务可以保证原有系统的架构不会发生改变，而使用函数库的话很可能需要对系统的架构做出大幅度的改动。

（2）系统中很多事件的发生是需要告知其他用户和服务器的，使用一个基于文件系统的锁服务可以将这些变动写入文件中。这样其他需要了解这些变动的用户和服务器直接访问这些文件即可，避免了因大量的系统组件之间的事件通信带来的系统性能下降。

（3）基于锁的开发接口容易被开发者接受。虽然在分布式系统中锁的使用会有很大的不同，但是和一致性算法相比，锁显然被更多的开发者所熟知。

- 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。

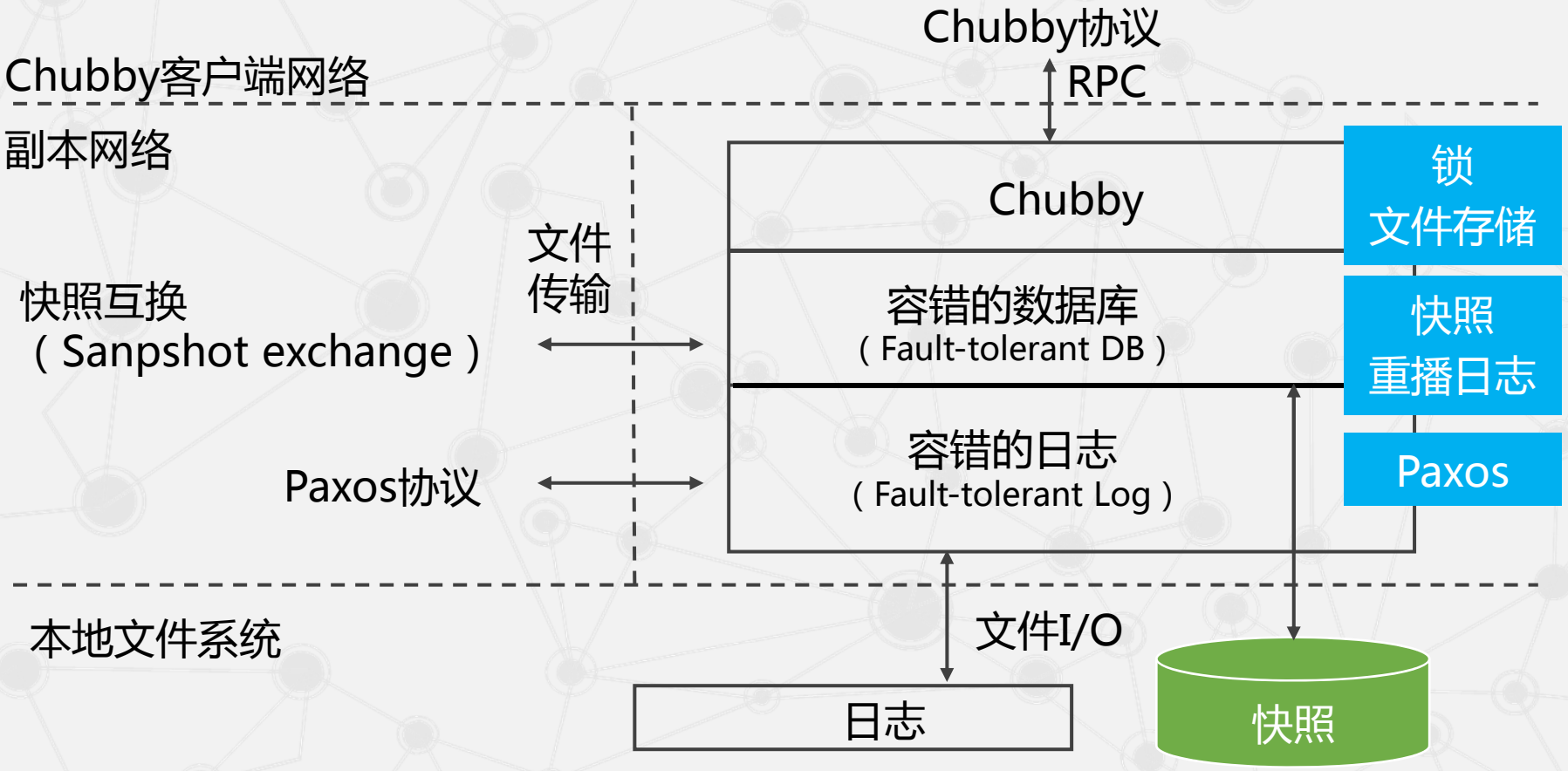
服务器端

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。

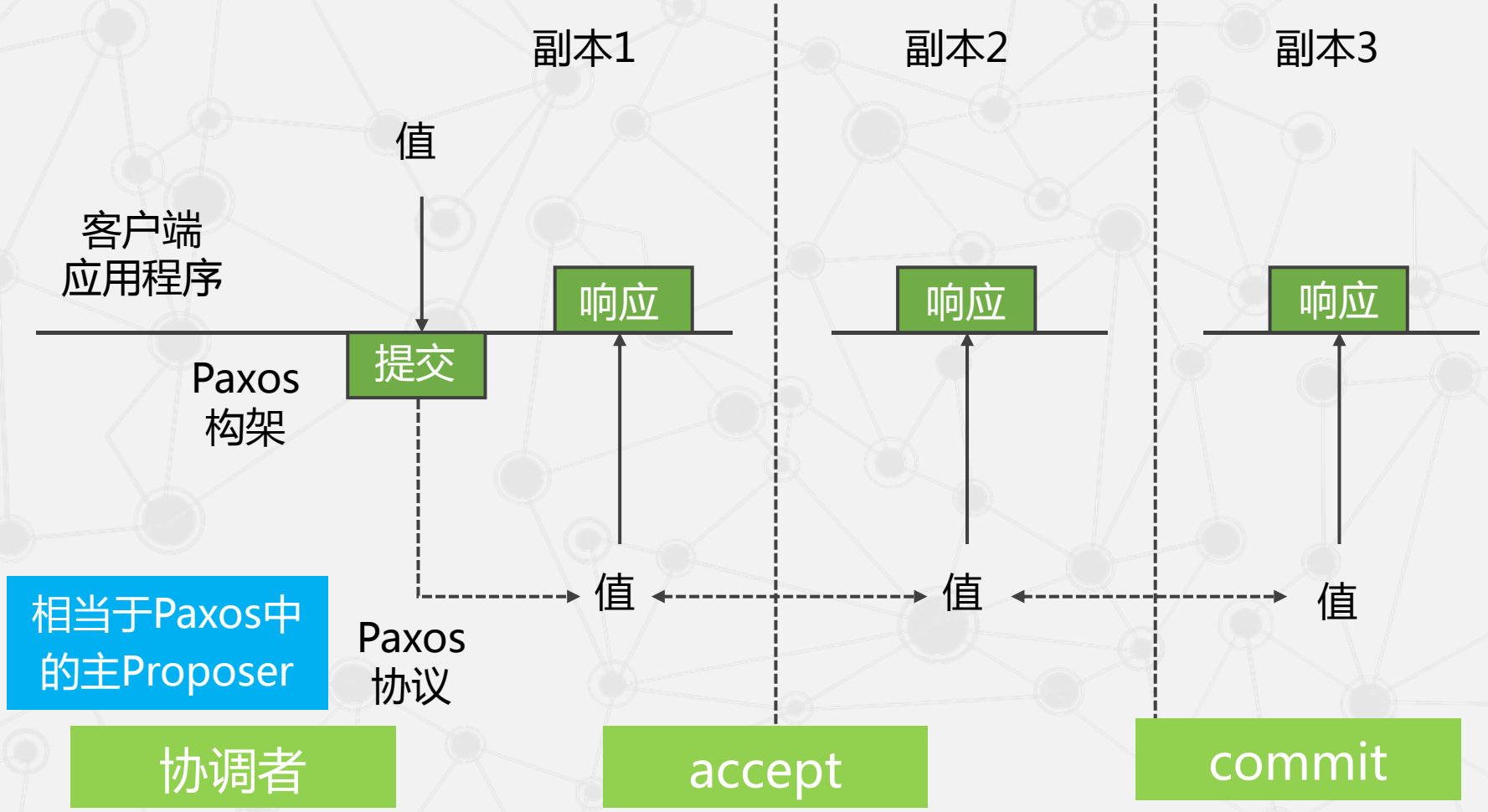
2.3 分布式锁服务Chubby

- 单个Chubby副本结构

客户端通过特定的Chubby协议和单个的Chubby副本进行通信



- 容错日志的API


副本1
副本2
副本3
值
客户端
应用程序
响应
响应
响应
提交
Paxos
构架
值
值
值
相当于Paxos中
的主Proposer
Paxos
协议
协调者
accept
commit

单个协调者可能失效 —— 多个协调者

1）给协调者指派序号 ——P27

2）Paxos强制新的协调者必须选择和前任相同的值

为了提高系统的效率，Chubby做了一个重要的优化，那就是在选择某一个副本作为协调者之后就长期不变

主服务器 租约期

1）租续期内所有的客户请求都由主服务器处理

2）如果某个服务器被连续推举为主服务器的话， 这个租约期就会不断地被更新

Chubby不是一个完全经过理论上验证的系统

2.3 分布式锁服务Chubby

Chubby系统本质上就是一个分布式的、存储大量小文件的文件系统，它所有的操作都是在文件的基础上完成

- Chubby最常用的锁服务中，每一个文件就代表一个锁，用户通过打开、关闭和读取文件，获取共享（Shared）锁或独占（Exclusive）锁
- 选举主服务器过程中，符合条件的服务器都同时申请打开某个文件并请求锁住该文件
- 成功获得锁的服务器自动成为主服务器并将其地址写入这个文件夹，以便其他服务器和用户可以获知主服务器的地址信息

- **单调递增的64位编号**

① 实例号 Instance Number	新节点实例号必定大于旧节点的实例号。
② 内容生成号 Content Generation Number	文件内容修改时该号增加。
③ 锁生成号 Lock Generation Number	锁被用户持有时该号增加。
④ ACL生成号 ACL Generation Number	ACL名被覆写时该号增加。

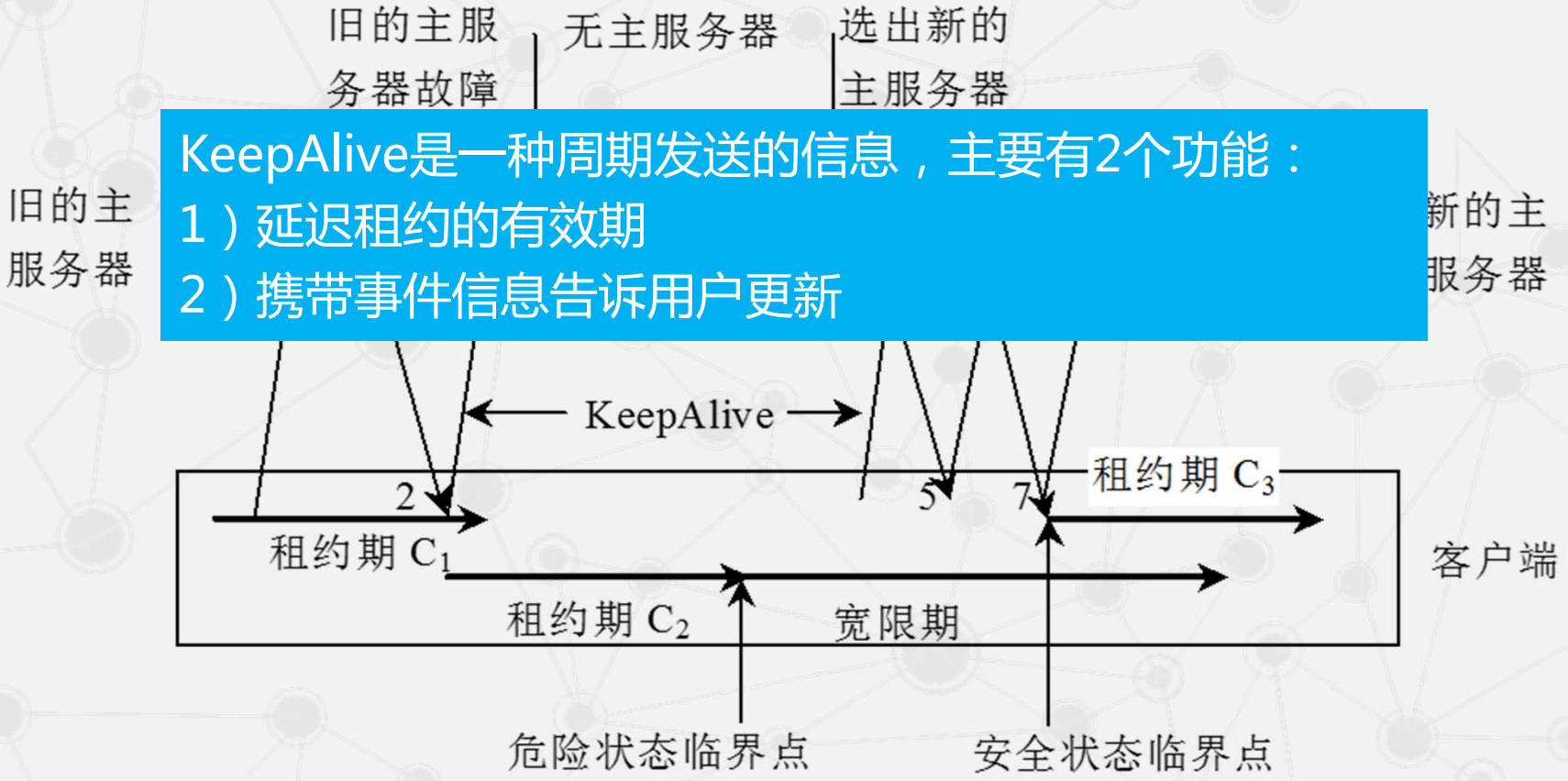
常用的句柄函数及作用

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

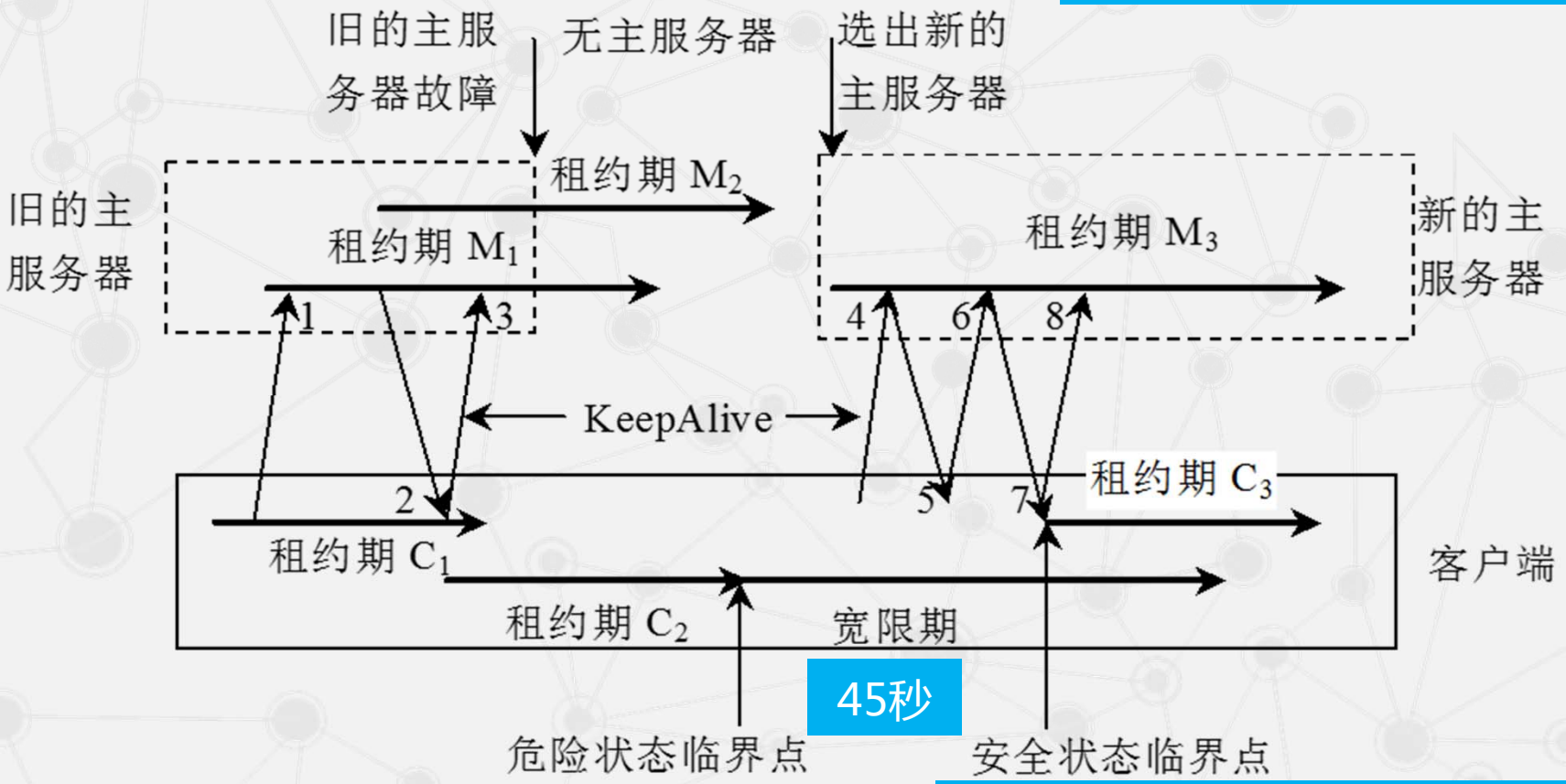
KeepAlive握手协议



KeepAlive是一种周期发送的信息，主要有2个功能：
1）延迟租约的有效期
2）携带事件信息告诉用户更新

- Chubby客户端与服务器端的通信过程

KeepAlive握手协议



危险事件

纪元号（Epoch Number）

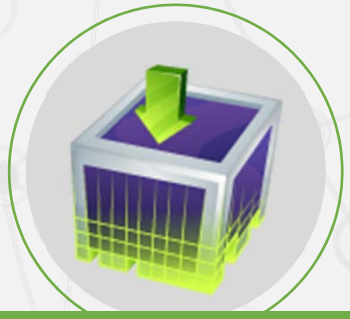
- 可能出现的两种故障

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

Paxos

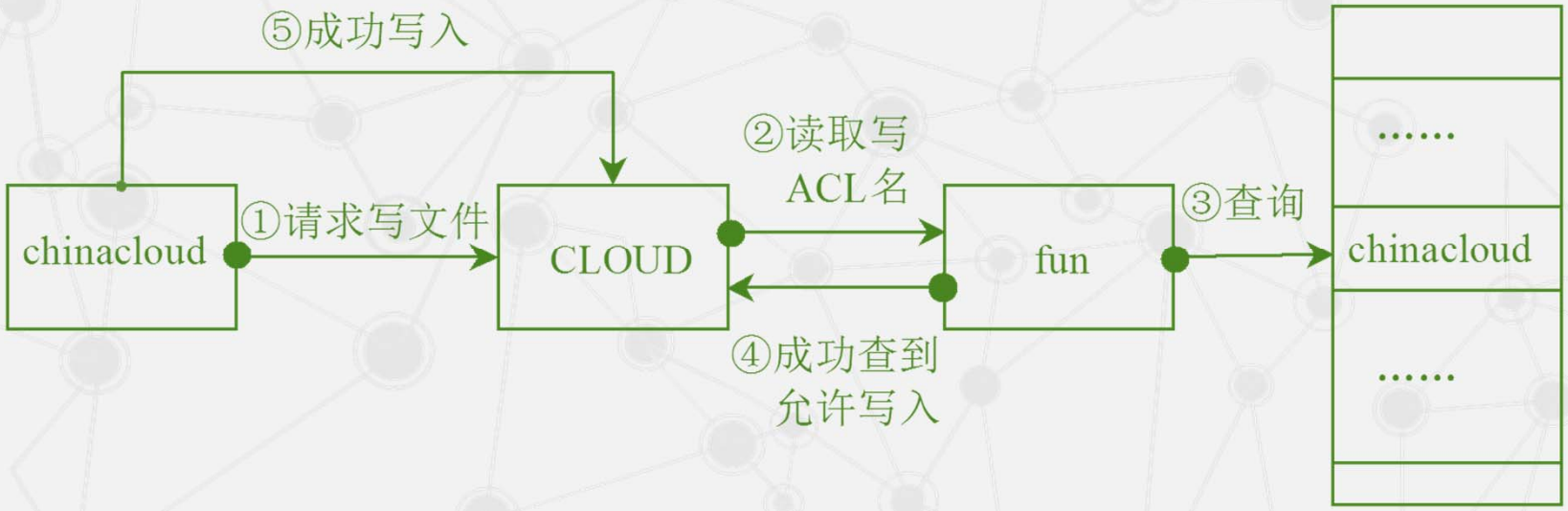
安全性

采用的是ACL（访问控制表）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（访问控制表）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

基于GFS和Chubby的分布式存储系统

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，因为对于客户来说，短暂的服务中断也不能忍受
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

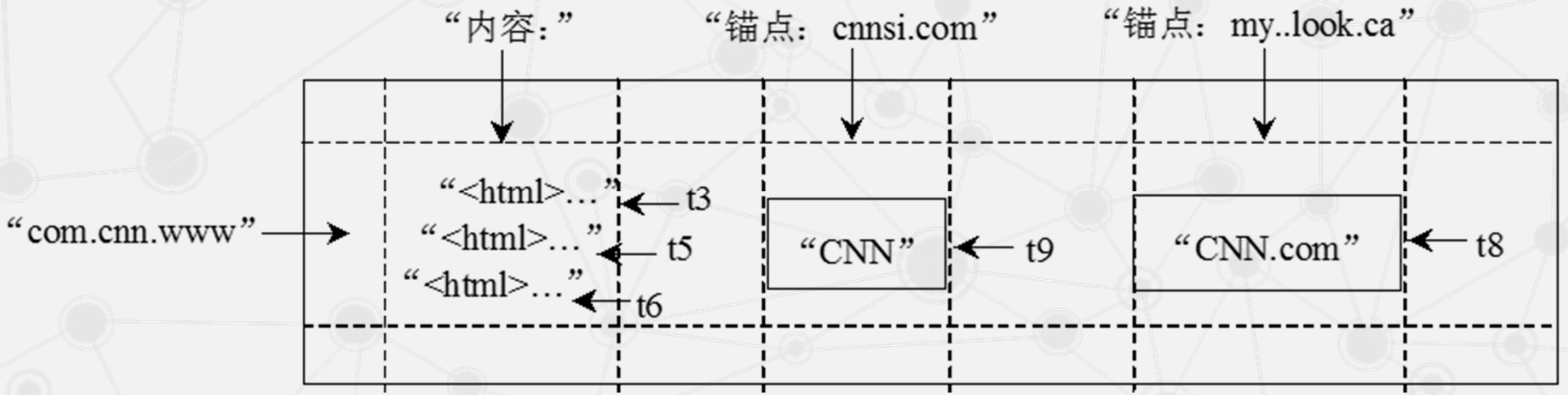
2.4 分布式结构化数据表Bigtable

- **Bigtable数据的存储格式**

Bigtable对存储的数据不做解析，一律看作字符串，具体数据结构由用户实现

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

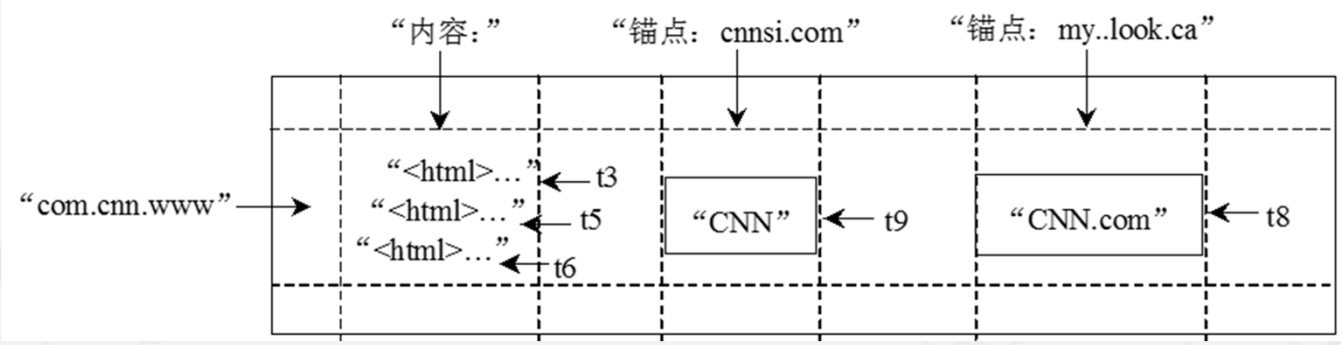


行

- 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
- 网页地址倒排，便于数据压缩，可以大幅提高压缩率
- 同一地址域的网页会被存储在表中的连续位置

子表（Tablet）

每个子表包含多个行，是数据划分和负载均衡的基本单位



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
- 网页地址倒排，便于数据压缩，可以大幅提高压缩率
- 同一地址域的网页会被存储在表中的连续位置

列

- 不是简单地存储列关键字，而是将其组织成所谓的列族
- 列关键字
  族名：限定词(family: qualifier)
  族名必须有意义，限定词可任选
- 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**

行

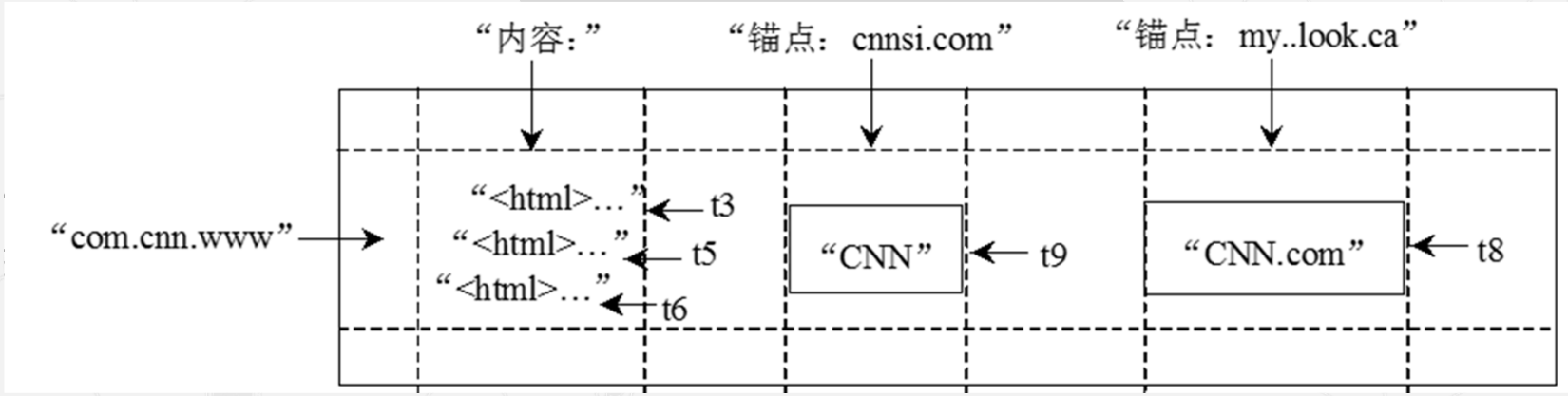
- Bigtable的行关键字可以是任意的字符串，但是大小不能够超过64KB
- 表中数据都是根据行关键字进行排序的，排序使用的是词典序

列

- 不是简单地存储列关键字，而是将其组织成所谓的列族
- 列关键字

  族名：限定词(family: qualifier)

  族名必须有意义，限定词可任选



2.4 分布式结构化数据表Bigtable

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
- 网页地址倒排，便于数据压缩，可以大幅提高压缩率
- 同一地址域的网页会被存储在表中的连续位置

列

- 不是简单地存储列关键字，而是将其组织成所谓的列族
- 列关键字

  族名：限定词(family: qualifier)

  族名必须有意义，限定词可任选
- 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

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排序的，排序使用的是词典序
-

列

- 不是简单地存储列关键字，而是将其组织成所谓的列族
- 族名必须有意义，限定词可任选

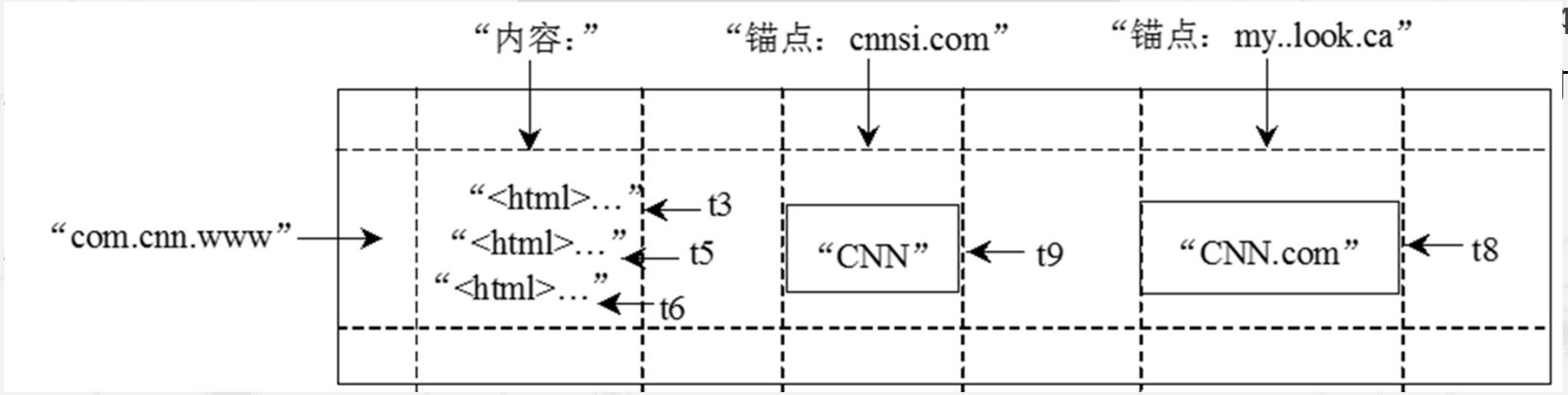
- 保留最近N个不同版本
- 保留限定时间内的所有不同版本

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。

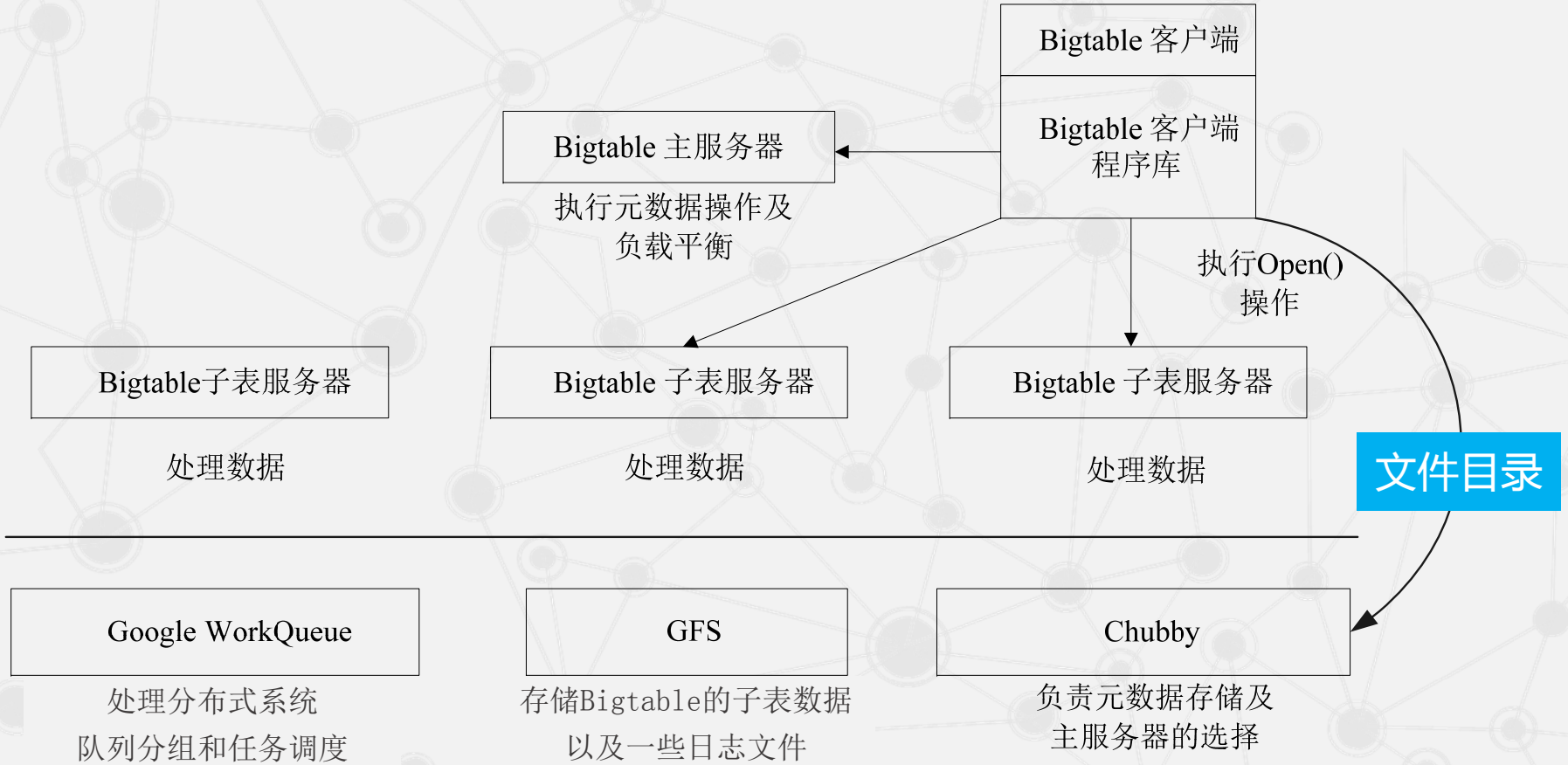
位整

以用户



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

- 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

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。

创建新表、表合并是由主服务器发起的，主服务器会自动检测到。

较大子表的分裂是由子表服务器发起的，主服务器无法自动检测到，需要分割完成之后子服务器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——保证良好的扩展性

Bigtable中主服务器对子表服务器的监控是通过Chubby完成的

- 子表（Tablet）服务器初始化时，会在Chubby的一个指定目录（服务器目录）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；
- 所有的子表服务器基本信息都被保存在服务器目录中；
- Master服务器实时监控着这个目录（服务器目录），因此Master服务器能够知道有新的Tablet服务器加入了。
- 主服务器会定期询问每个具体的子表服务器独占锁的状态
- 如果子表服务器的锁丢失或没有回应——Chubby提供一种高效机制，在不增加网络负担的情况下会知道它是否还持有锁
  - Chubby服务问题——主服务器首先自己尝试获取独占锁，若失败说明Chubby服务出了问题，需要等待Chubby服务的恢复。
  - 子表服务器问题——主服务器获取独占锁成功，说明子表服务器出现了问题。主服务器就中止这个子表服务器并将其上的子表全部移至其他子表服务器。

系统出故障是一种常态的设计理念，所以每个主服务器都有会话时间的限制。某个主服务器到时退出后，系统会指定一个新的主服务器

新的主服务器启动步骤

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

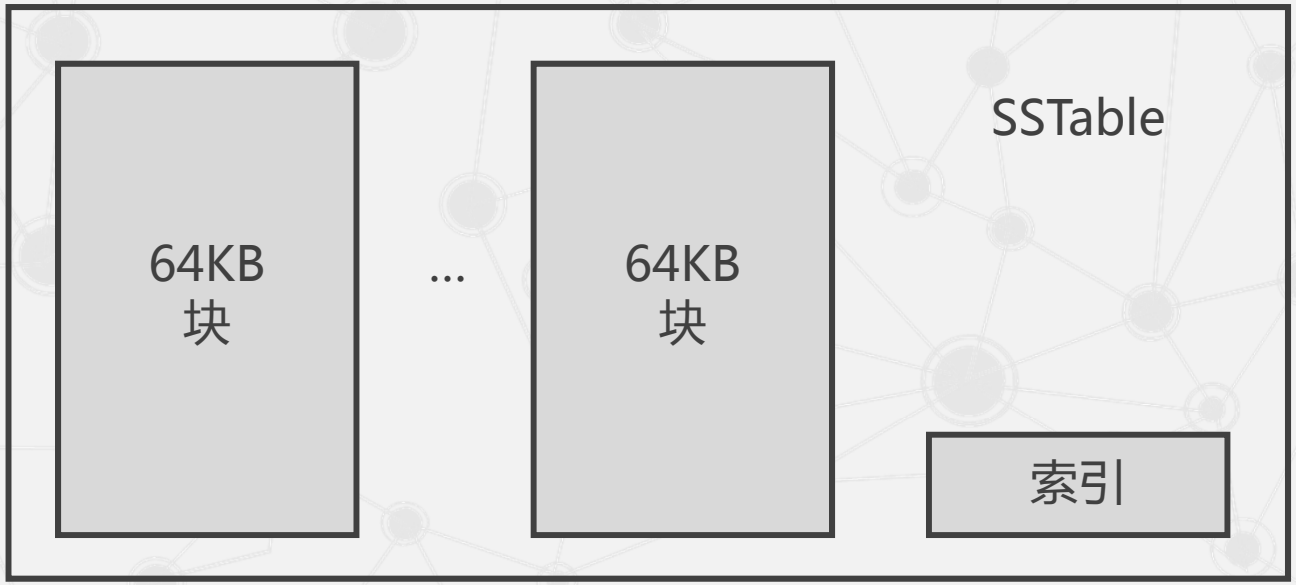
- 2.4.1 设计动机与目标
- 2.4.2 数据模型
- 2.4.3 系统架构
- 2.4.4 主服务器
- ▶ 2.4.5 子表服务器
- 2.4.6 性能优化

SSTable 格式的基本示意

Bigtable中实际的数据都是以子表的形式保存在子表服务器上的。

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。



索引：记录块的位置信息。

SSTable打开时，索引被加载到内存

- **子表实际组成**

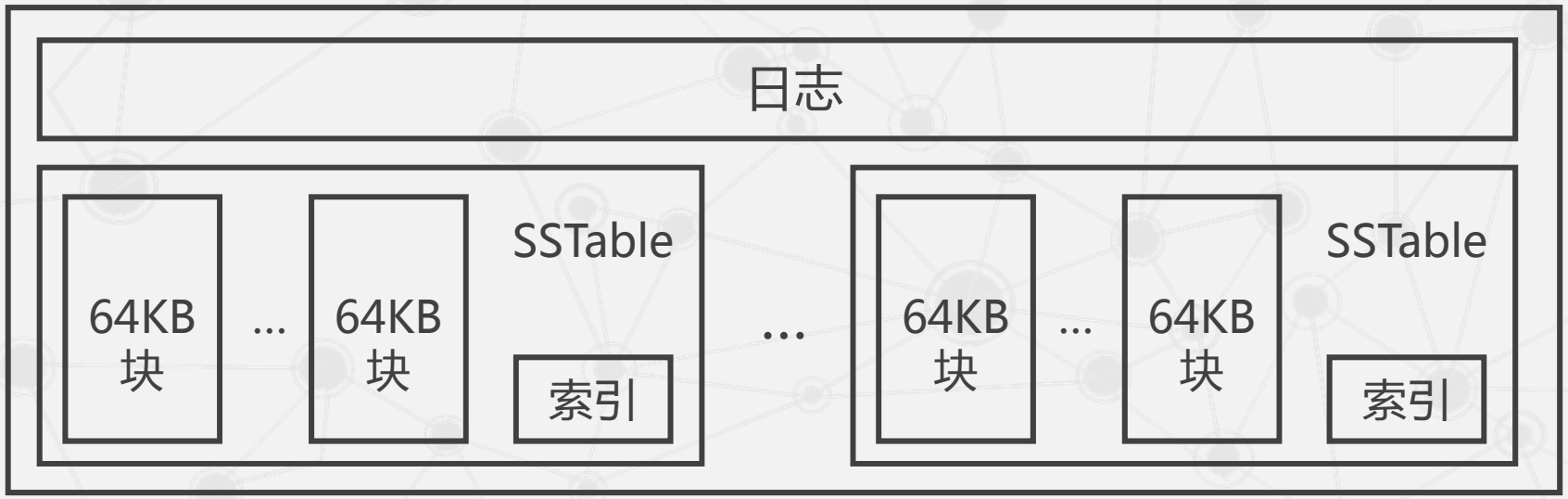
子表是一系列行的集合，每个子表由多个SSTable及日志文件构成

不同子表的SSTable可以共享

共享日志，每个子表服务器上仅保存一个日志文件，不是每个子表建立一个日志

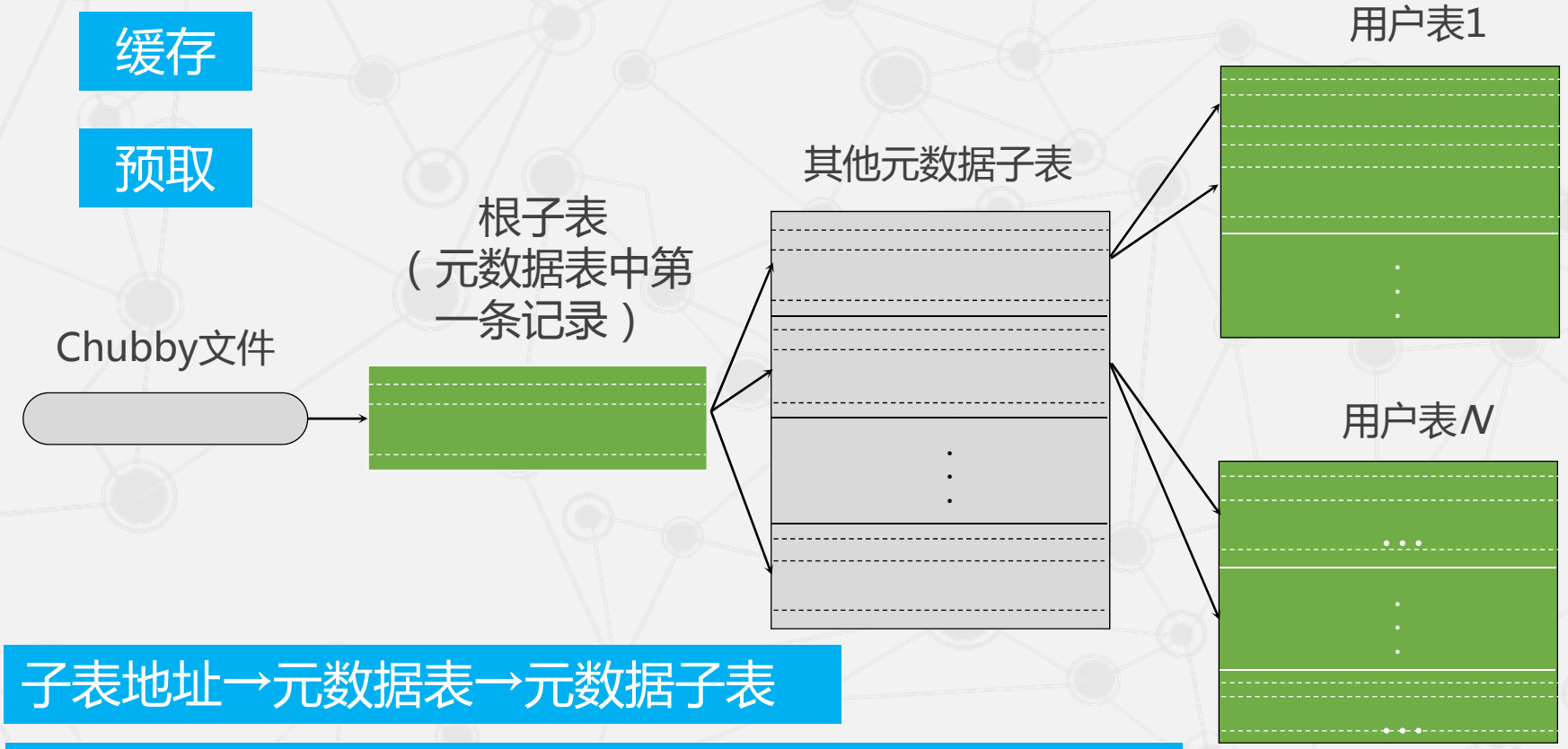
Bigtable规定将日志的内容按照键值进行排序，不同的子表服务器可以连续读取日志

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



子表地址→元数据表→元数据子表

根子表：元数据表的第一条记录和其他元数据子表的地址

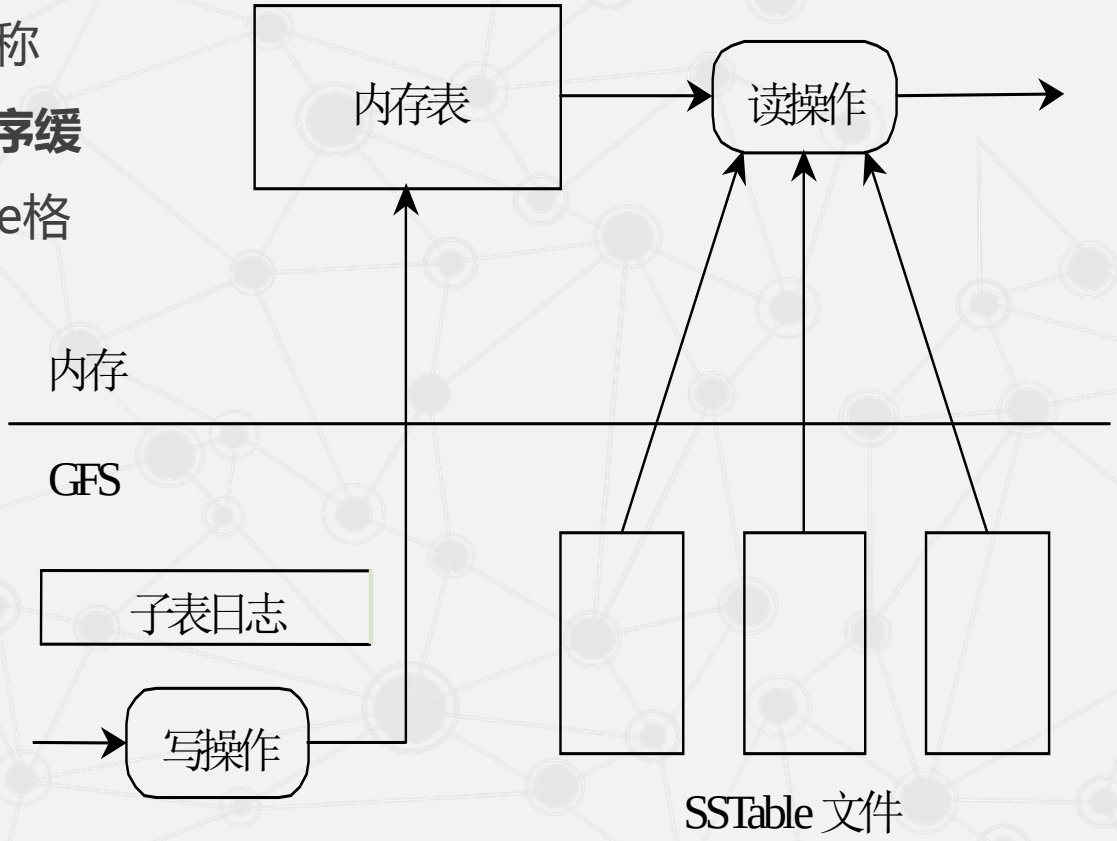
- **子表数据存储及读/写操作**

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

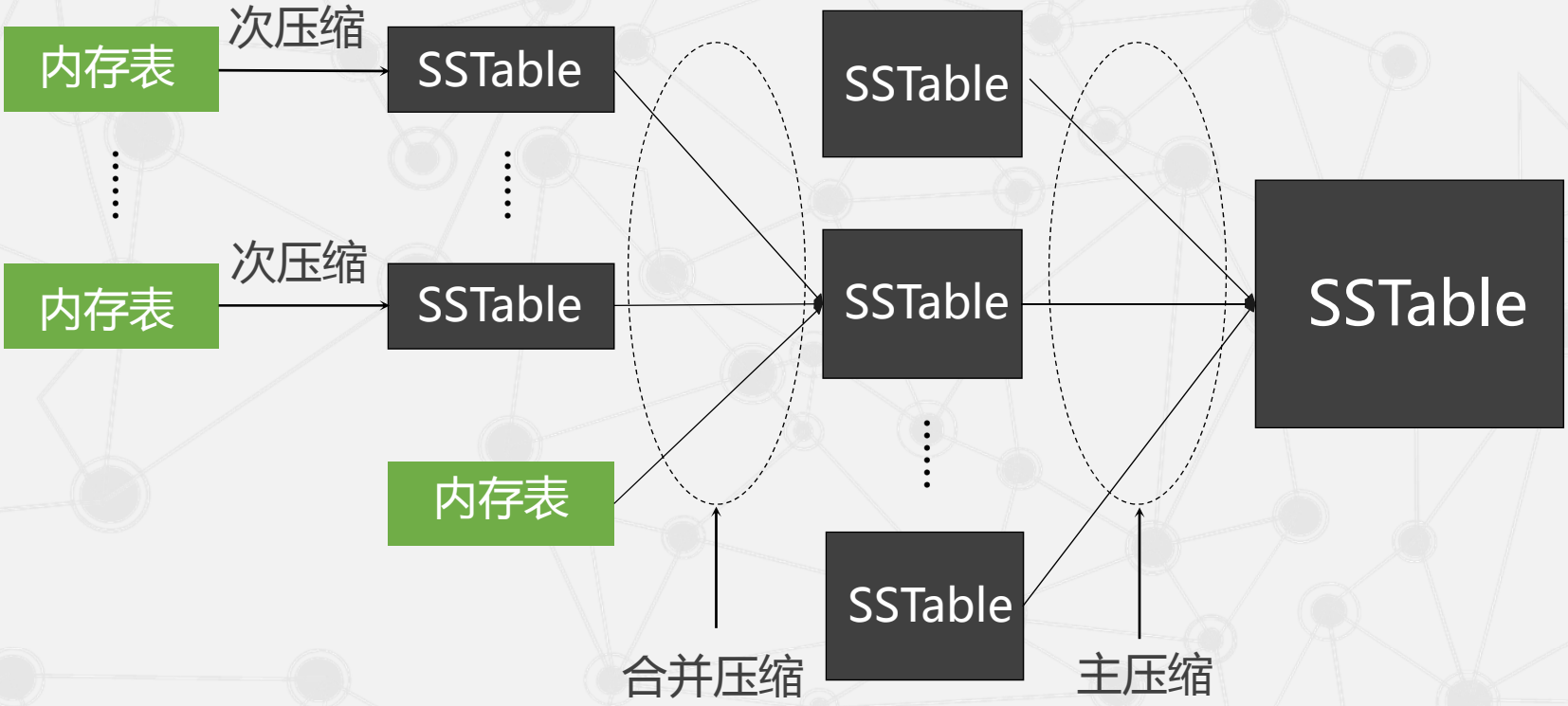
SSTable数量过多，会对读或写哪种操作影响较大？

读操作



- 三种形式压缩之间的关系

主压缩是合并压缩的一种



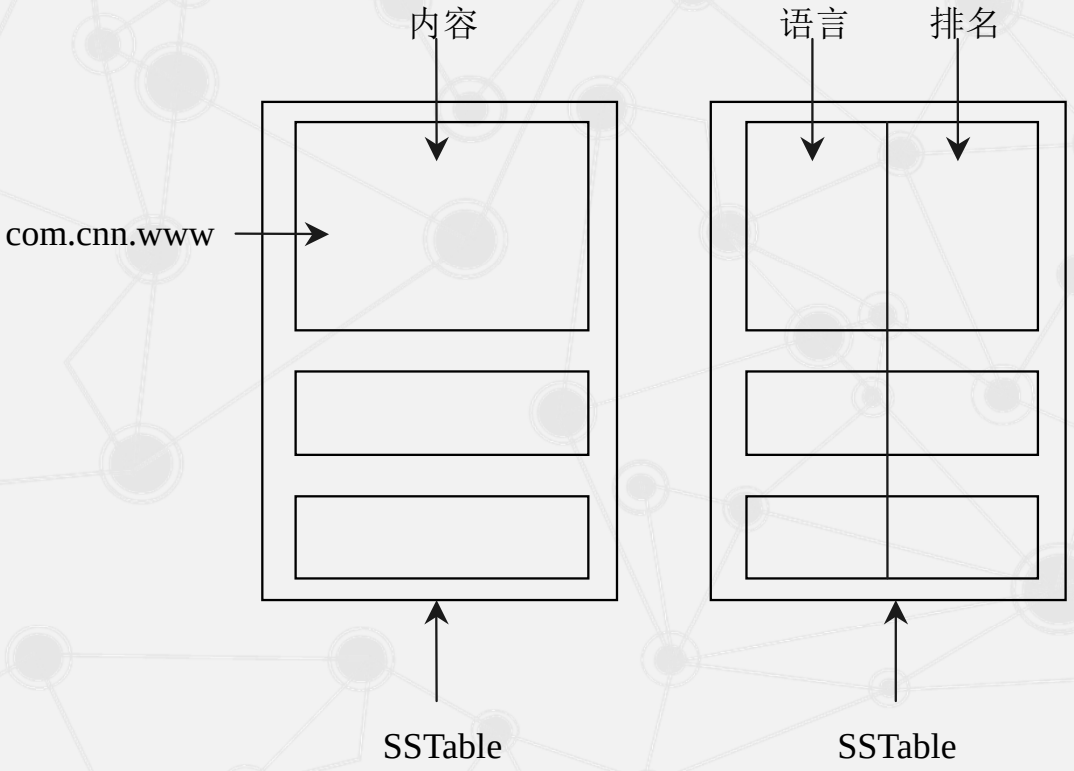
执行一次主压缩后可以保证所有的被压缩数据彻底删除

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