一、分布式锁服务

今天，要接触有些难理解的知识点了，这也许就是涉及到当时赵致琢老师强调的在中国没人能有资格讲和讲得清的一块—分布式算法。说实话，这块看了两遍了，到现在还不敢说自己人懂了一半啊·！

Chubby

øGoogle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

——一种建议性的锁(相信看过《UNIX环境下高级编程》的人对建议性的锁这个名词不会陌生)，而不是一种强制性的锁：具有更大的灵活性

øGFS使用Chubby选取一个GFS主服务器

øBigtable使用Chubby指定一个主服务器并发现、控制与其相关的子表服务器

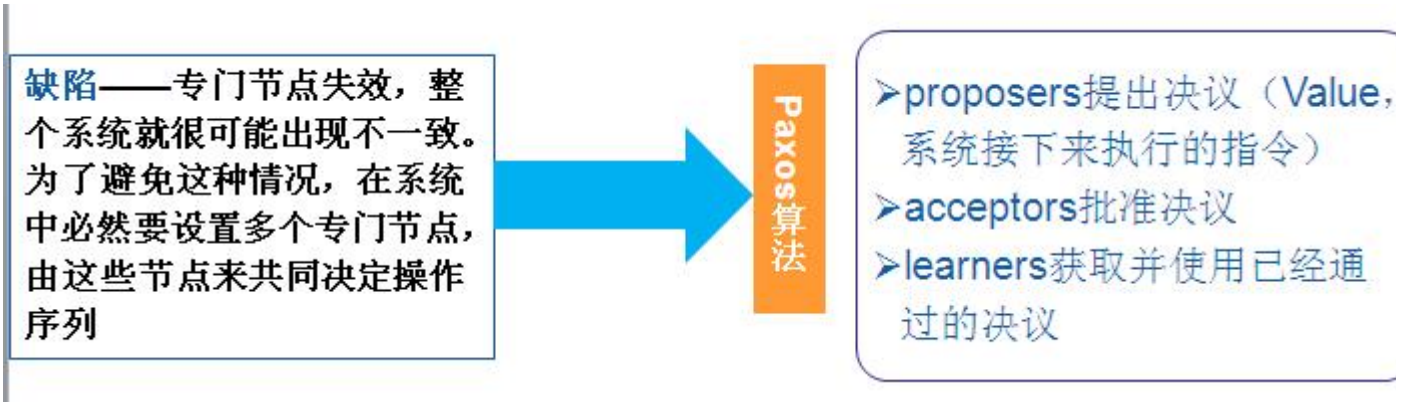
øChubby还可以作为一个稳定的存储系统存储包括元数据在内的小数据

øGoogle内部还使用Chubby进行名字服务（Name Server）

想像一下，要在大规模集群的条件下，保证所有指令和数据的一致性（即：在初始状态相同情况下，要求各结点接收到同样相同指令，且最终状态一致）会遇到什么样的困难？——这也许正是分布式算法要发挥作用的境地，很多时候设计的算法根本不可能会是十全十美。Chubby中即要用到Paxos算法

1、Paxos算法

试想想：该方案存在什么缺陷？？？？



试图由以下三点来保证数据的一致性:

（1）决议只有被proposers提出后才能批准
（2）每次只批准一个决议
（3）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

p1表明每个可以接收到多个决议，为区分，对每个决议进行编号，后得到的决议编号要大于先到的编号；p1不是很完备!!(??一个问题可能是：对于每个结点，其收到的所谓第一个编号是否都是一样？？)

P2：一旦某个决议通过，之后通过的决议必须和该决议保持一致

P1+P2——>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

P2a和P1是有矛盾的!(我的理解是：有可能这个V不是某个结点收到的第一个决议)

P2a——» 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

P1和P2b保证条件（2），彼此之间不存在矛盾。但是P2b很难通过一种技术手段来实现它，因此提出了一个蕴涵P2b的约束P2c

P2b——» 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

决议通过的两个阶段:

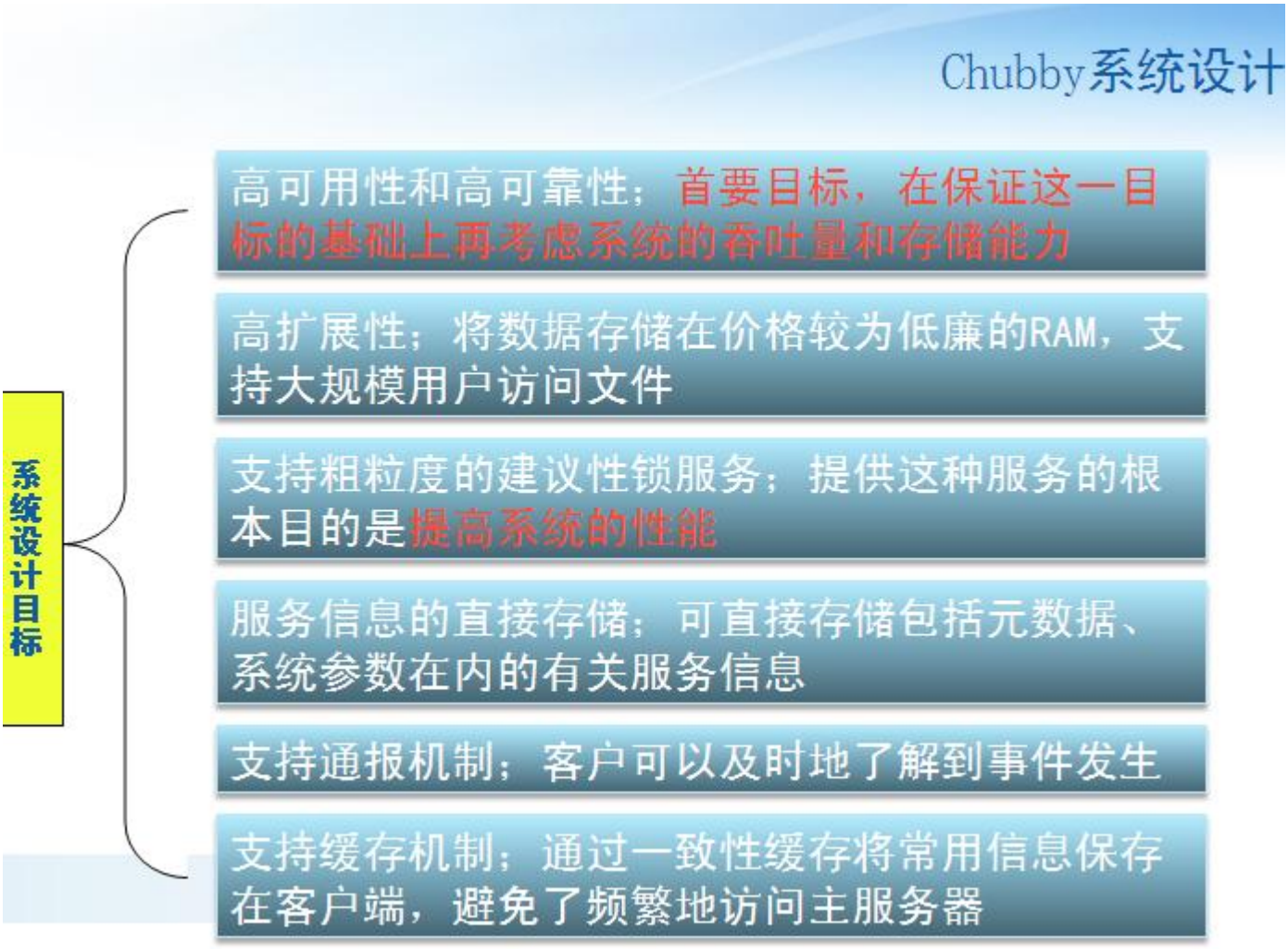
准备阶段：proposers选择一个提案并将它的编号设为n，然后将它发送给acceptors中的一个“多数派”。Acceptors收到后，如果提案的编号大于它已经回复的所有消息，则acceptors将自己上次的批准回复给proposers，并不再批准小于n的提案 （那么，可以问问：如果小于它已经回复的所有消息呢？这个思考之后，对算法的流程就有个印象——但似乎这样一想，这中间的延迟倒很是个问题，看来，这个算法还是未弄懂！！）

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

解决一致性问题算法：为了减少决议发布过程中的消息量，acceptors将这个通过的决议发送给learners的一个子集，然后由这个子集中的learners去通知所有其他的learners;

特殊情况：如果两个proposer在这种情况下都转而提出一个编号更大的提案，那么就可能陷入活锁。此时需要选举出一个president，仅允许 president提出提案

2、Chubby的系统设计



Chubby中还添加了一些新的功能特性；这种设计主要是考虑到以下几个问题：

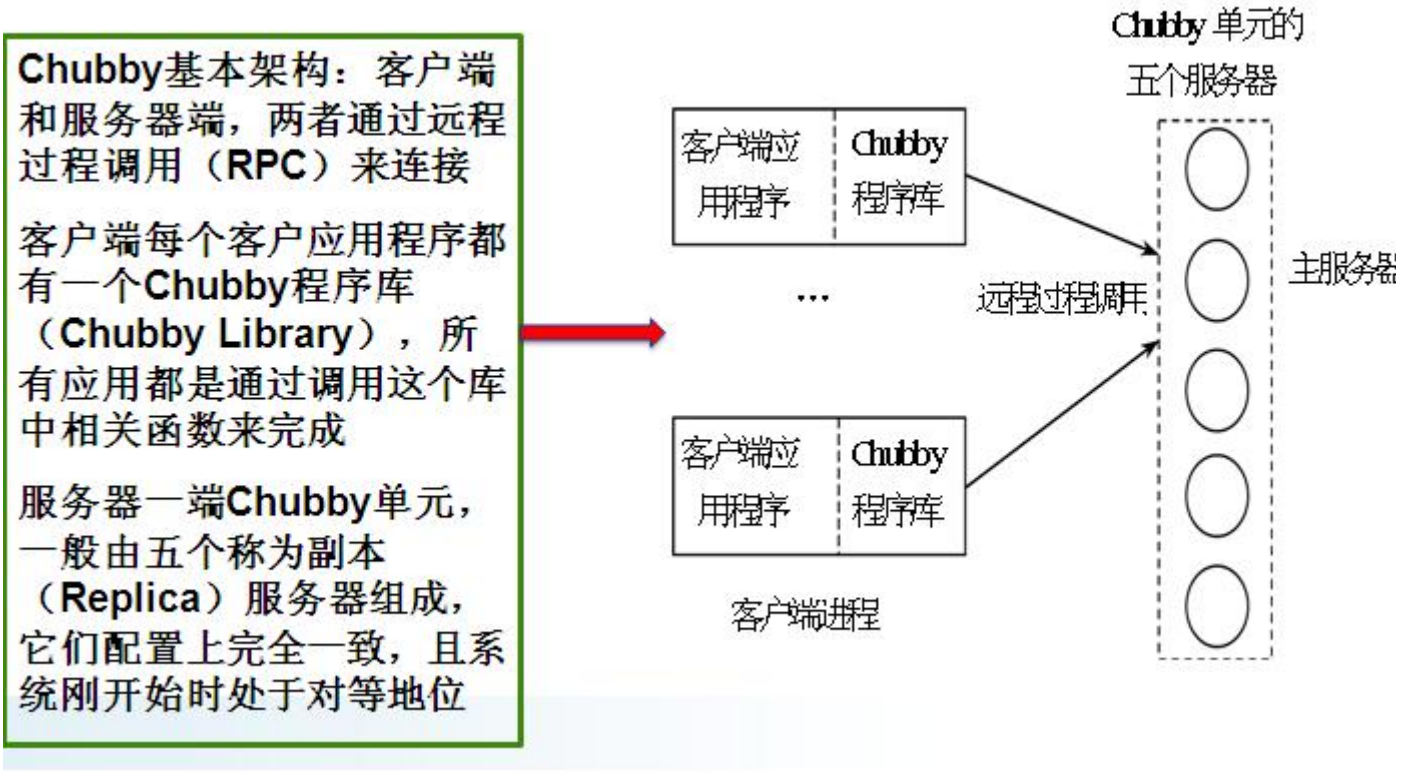
1、开发者初期很少考虑系统的一致性，但随着开发进行，问题会变得越来越严重。单独的锁服务可以保证原有系统架构不会发生改变，而使用函数库很可能需要对系统架构做出大幅度的改动

2、系统中很多事件发生是需要告知其他用户和服务器，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。有需要的用户和服务器直接访问这些文件即可，避免因大量系统组件之间事件通信带来系统性能下降

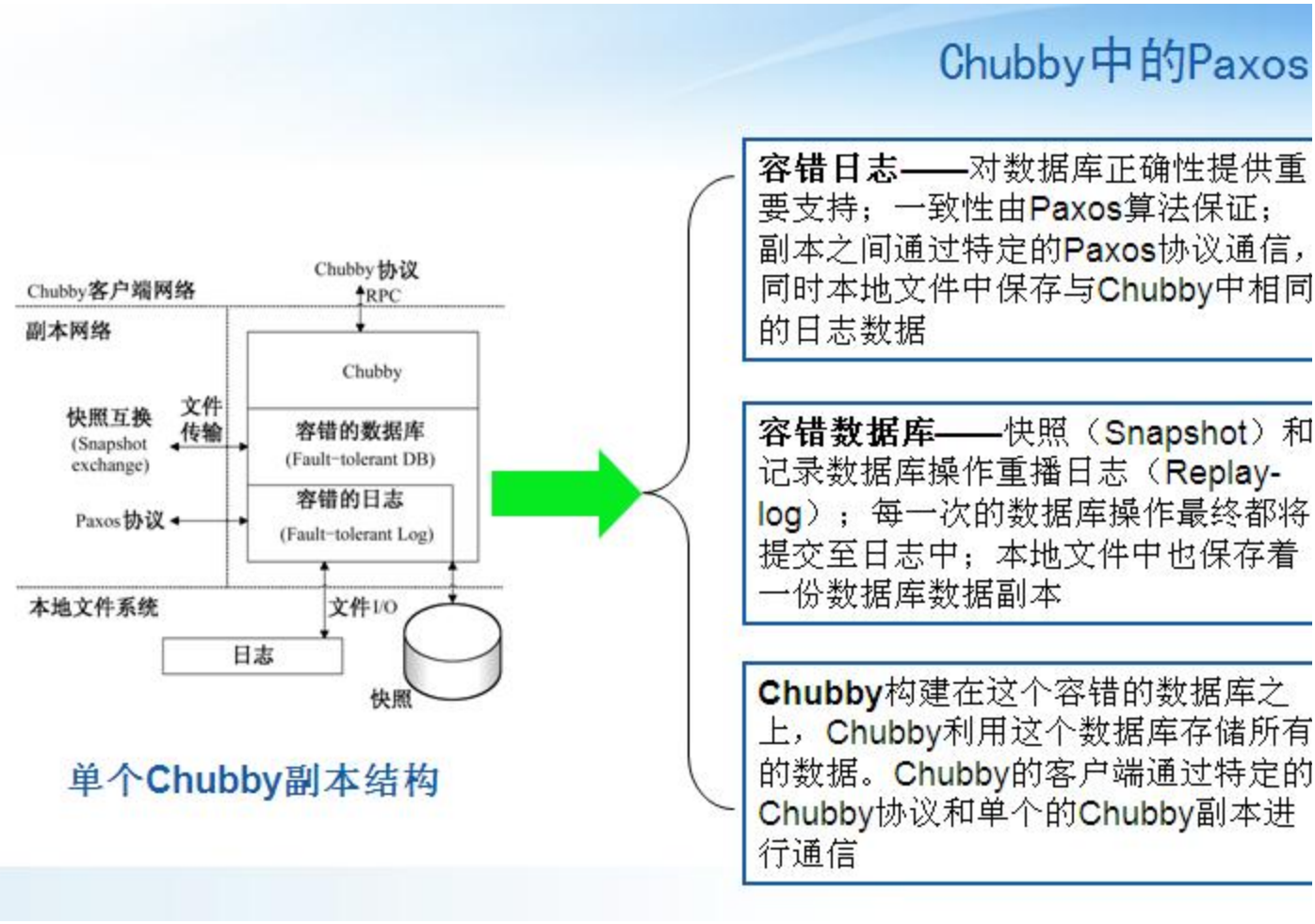
3、基于锁的开发接口容易被开发者接受。虽然在分布式系统中锁的使用会有很大的不同，但是和一致性算法相比，锁显然被更多的开发者所熟知

Paxos算法实现过程中需要一个“多数派”就某个值达成一致，本质上就是分布式系统中常见的quorum机制；为保证系统高可用性，需要若干台机器，但使用单独锁服务的话一台机器也能保证这种高可用性

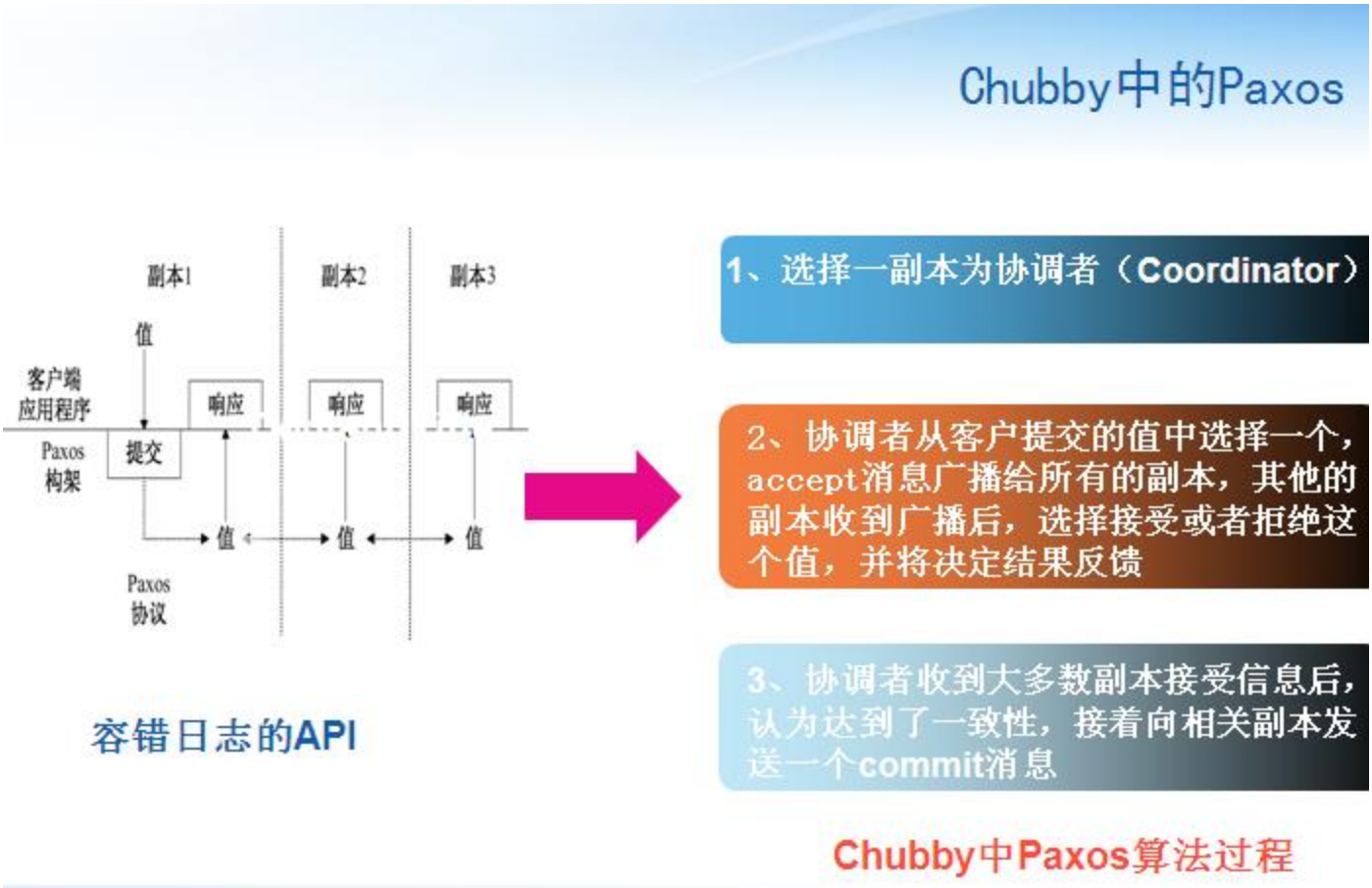
Chubby设计过程中一些细节问题值得关注：
在Chubby系统中采用了建议性的锁而没有采用强制性的锁。两者的根本区别在于用户访问某个被锁定的文件时，建议性的锁不会阻止访问，而强制性的锁则会阻止访问，实际上这是为了方便系统组件之间的信息交互
另外，Chubby还采用了粗粒度（Coarse-Grained）锁服务而没有采用细粒度（Fine-Grained）锁服务，两者的差异在于持有锁的时间，细粒度的锁持有时间很短



3、Chubby中的Paxos算法



(个人疑问：从上面来看，似乎上面给我们的启发是——我们无需在整个系统的每个环节保持数据和指令的一致性，只需其操作日志是一致，那么说明其操作一致？？)



øChubby设计者借鉴了Paxos的两种解决机制：给协调者指派序号或限制协调者可以选择的值

F指派序号的方法

–（1）在一个有n个副本系统中，为每个副本分配一个id，其中 0≤ir≤n-1。则副本的序号，其中k的初始值为0 ("则副本的序号，其中k的初始值为0 "这句话可能写得有点问题，这里没看懂)

–（2）某个副本想成为协调者之后，它就根据规则生成一个比它以前的序号更大的序号（实际上就是提高k的值），并将这个序号通过propose消息广播给其他所有的副本

–（3）如果接受到广播的副本发现该序号比它以前见过的序号都大，则向发出广播的副本返回一个promise消息，并且承诺不再接受旧的协调者发送的消息。如果大多数副本都返回了promise消息，则新的协调者就产生了

F限制协调者可以选择的值

–Paxos强制新的协调者必须选择和前任相同的值

øChubby做了一个重要优化来提高系统效率—在选择某一个副本作为协调者之后就长期不变，此时协调者就被称为主服务器（Master）

F客户端的数据请求由主服务器完成，Chubby保证在一定时间内有且仅有一个主服务器，这个时间就称为主服务器租约期（Master Lease）

F客户端需要确定主服务器的位置，可向DNS发送一个主服务器定位请求，非主服务器的副本将对该请求做出回应

ø Chubby对于Paxos论文中未提及的一些技术细节进行了补充，所以Chubby的实现是基于Paxos，但其技术手段更加的丰富，更具有实践性

4、Chubby文件系统

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

øChubby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

ø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

ø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

øChubby的文件系统和UNIX类似

F例如在文件名“/ls/foo/wombat/pouch”中，ls代表lock service，这是所有Chubby文件系统的共有前缀；foo是某个单元的名称；/wombat/pouch则是foo这个单元上的文件目录或者文件名

ø Google对Chubby做了一些与UNIX不同的改变

F例如Chubby不支持内部文件的移动；不记录文件的最后访问时间；另外在Chubby中并没有符号连接（Symbolic Link，又叫软连接，类似于Windows系统中的快捷方式）和硬连接（HardLink，类似于别名）的概念

ø在具体实现时，文件系统由许多节点组成，分为永久型和临时型，每个节点就是一个文件或目录。节点中保存着包括ACL（Access Control List，访问控制列表）在内的多种系统元数据 o

Chubby文件系统

- 用户打开某个节点的同时会获取一个类似于UNIX中文件描述符（File Descriptor）的句柄，这个句柄由以下三个部分组成

Chubby文件系统

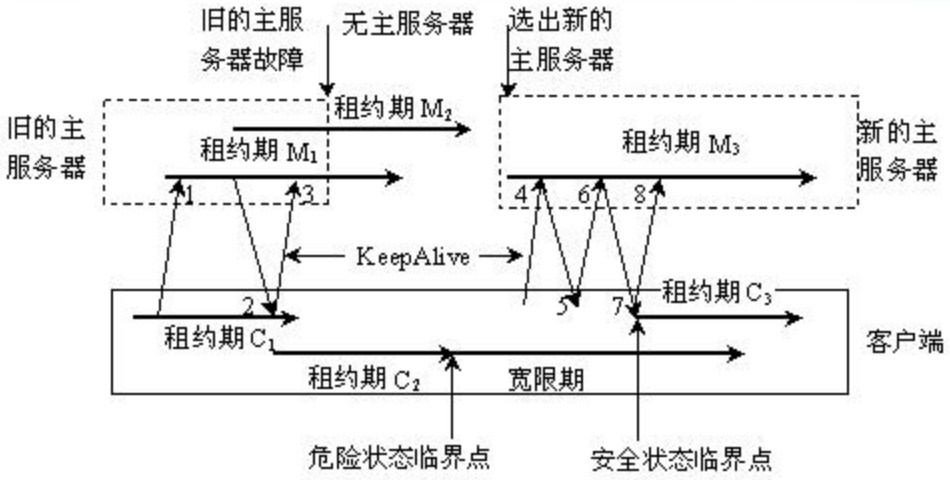
- 在实际执行中，为了避免所有的通信都使用序号带来系统开销增长，Chubby引入了sequencer的概念
- sequencer实际上就是一个序号，只能由锁的持有者在获取锁时向系统发出请求来获得。这样一来Chubby系统中只有涉及锁的操作才需要序号，其他一概不用。
- 在文件操作中，用户可以将句柄看做一个指向文件系统的指针

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

常用句柄函数及其作用

5、通信协议

Chubby客户端与服务器端的通信过程



⇨从左到右的水平方向表示时间在增加，斜向上的箭头表示一次KeepAlive请求，斜向下的箭头则是主服务器的一次回应

⇨ M1、M2、M3表示不同的主服务器租约期；C1、C2、C3则是客户端对主服务器租约期时长做出的一个估计

⇨ KeepAlive是周期发送的一种信息，它主要有两方面的功能：延迟租约的有效期和携带事件信息告诉用户更新

故障处理

客户端租约过期

Ø客户端向主服务器发出一个KeepAlive请求（上图1）

Ø如果有需要通知的事件时则主服务器会立刻做出回应，否则等到客户端的租约期C1快结束的时候才做出回应（图2），并更新主服务器租约期为M2

Ø客户端接到回应后认为该主服务器仍处于活跃状态，于是将租约期更新为C2并立刻发出新的KeepAlive请求（图3）

Ø宽限期内，客户端不会立刻断开其与服务器端的联系，而是不断地做探询，当它接到客户端的第一个KeepAlive请求（图4）时会拒绝（图5）

Ø客户端在主服务器拒绝后使用新纪元号来发送KeepAlive请求（图6）

Ø新的主服务器接受这个请求并立刻做出回应（图7）

如果客户端接收到这个回应的时间仍处于宽限期内，系统会恢复到安全状态，租约期更新为C3。如果在宽限期未接到主服务器的相关回应，客户端终止当前的会话

主服务器出错

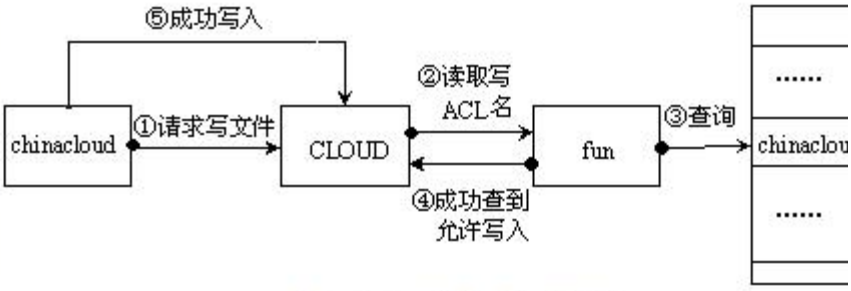
正常情况下旧的主服务器出现故障后系统会很快地选举出新的主服务器，新选举需要经历以下九个步骤：

（1）产生一个新的纪元号以便今后客户端通信时使用，这能保证当前的主服务器不必处理针对旧的主服务器的请求
（2）只处理主服务器位置相关的信息，不处理会话相关的信息
（3）构建处理会话和锁所需的内部数据结构
（4）允许客户端发送KeepAlive请求，不处理其他会话相关的信息
（5）向每个会话发送一个故障事件，促使所有的客户端清空缓存
（6）等待直到所有的会话都收到故障事件或会话终止
（7）开始允许执行所有的操作
（8）如果客户端使用了旧的句柄则需要为其重新构建新的句柄
（9）一定时间段后（1分钟），删除没有被打开过的临时文件夹
——如果这一过程在宽限期内顺利完成，则用户不会感觉到任何故障的发生，也就是说新旧主服务器的替换对于用户来说是透明的，用户感觉到的仅仅是一个延迟
Ø系统实现时，Chubby还使用了一致性客户端缓存（Consistent Client-Side Caching）技术，这样做的目的是减少通信压力，降低通信频率
F在客户端保存一个和单元上数据一致的本地缓存，需要时客户可以直接从缓存中取出数据而不用再和主服务器通信
F当某个文件数据或者元数据需要修改时，主服务器首先将这个修改阻塞；然后通过查询主服务器自身维护的一个缓存表，向对修改的数据进行了缓存的所有客户端发送一个无效标志（Invalidation）
F客户端收到这个无效标志后会返回一个确认（Acknowledge），主服务器在收到所有的确认后才解除阻塞并完成这次修改
——这个过程的执行效率非常高，仅仅需要发送一次无效标志即可，因为对于没有返回确认的节点，主服务器直接认为其是未缓存

6、正确性与性能
一致性
Ø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。实际执行过程中，Chubby使用Paxos算法来解决
Ø主服务器产生后客户端的所有读写操作都是由主服务器来完成的
a读操作很简单，客户直接从主服务器上读取所需数据即可
a写操作就会涉及数据一致性的问题；为了保证客户的写操作能够同步到所有的服务器上，系统再次利用了Paxos算法

安全性

- Chubby用ACL形式安全保障措施。系统有三种ACL名：写ACL名（Write ACL Name）、读ACL名（Read ACL Name）和变更ACL名（Change ACL Name）
- 只要不被覆写，子节点都是直接继承父节点的ACL名
- ACL同样被保存在文件中，它是节点元数据的一部分，用户在进行相关操作时首先需要通过ACL来获取相应的授权



Chubby的ACL机制

用户chinacloud提出向文件CLOUD中写入内容请求。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

性能优化

Ø为满足系统高可扩展性，Chubby目前已经采取了一些措施：比如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；除此之外，Google的工程师们还考虑使用代理（Proxy）和分区（Partition）技术

a代理可以减少主服务器处理KeepAlive以及读请求带来的服务器负载，但是它并不能减少写操作带来的通信量

a 使用分区技术的话可以将一个单元的命名空间（NameSpace）划分成 N 份。除了少量的跨分区通信外，大部分的分区都可以独自地处理服务请求。通过分区可以减少各个分区上的读写通信量，但不能减少 KeepAlive 请求的通信量