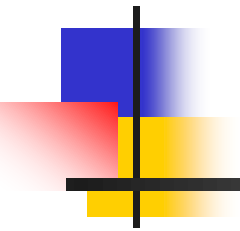


分布式系统

复制

第8章 复制

- 简介
- 系统模型和组通信
- 容错服务
- 高可用服务的实例研究：gossip体系结构、Bayou和Coda
- 复制数据上的事务
- 小结

简介

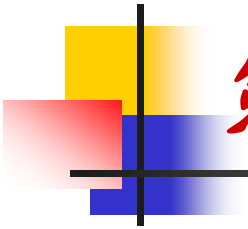
复制的概念

在多个计算机中进行数据副本的维护。

- **复制的动机：** 增强服务
  - 增强性能
    - 浏览器对Web资源的缓存
    - 数据在多个服务器之间的透明复制
  - 提高可用性
    - 服务器故障：$1-p^n$
    - 网络分区和断链操作：预先复制
  - 增强容错能力
    - 正确性：允许一定数量和类型的故障

简介

- 复制的基本要求
  - 复制透明性
    - 对客户屏蔽多个物理拷贝的存在
    - 客户仅对一个逻辑对象进行操作
  - 一致性
    - 在不同应用中有不同强度的一致性需求
    - 对复制对象集合的操作必须满足应用需求



第8章 复制

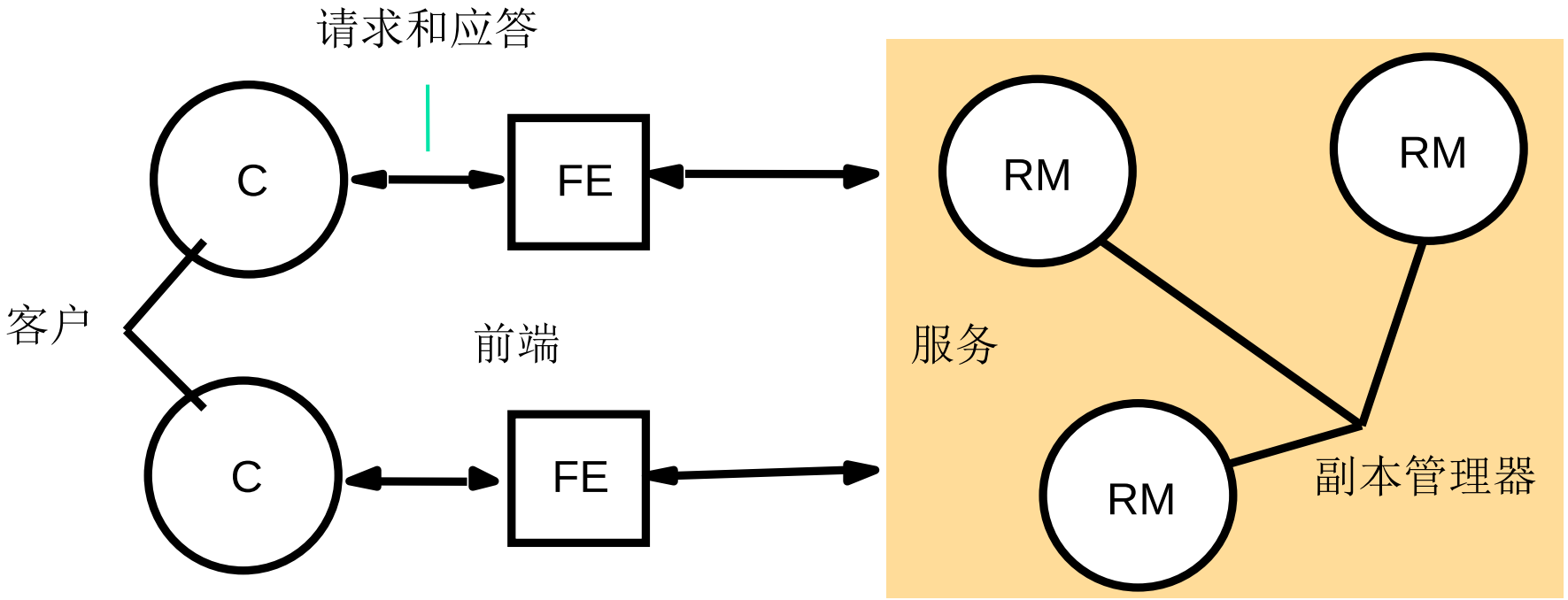
- 简介
- 系统模型和组通信
- 容错服务
- 高可用服务的实例研究：gossip体系结构、Bayou和Coda
- 复制数据上的事务
- 小结

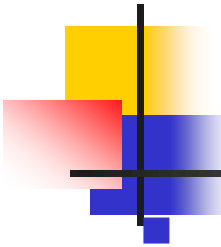
系统模型和组通信的作用

- 系统中的数据是由对象集合组成的
  - 一个对象可以是一个文件，或JAVA对象
- 每一个这样的逻辑对象是由若干称为副本的物理拷贝组成的集合实现的
  - 副本是物理对象
- 给定对象的副本未必完全相同
  - 至少不必有每个时间点上都要求一样

系统模型

基本模型





系统模型

基本模型组件

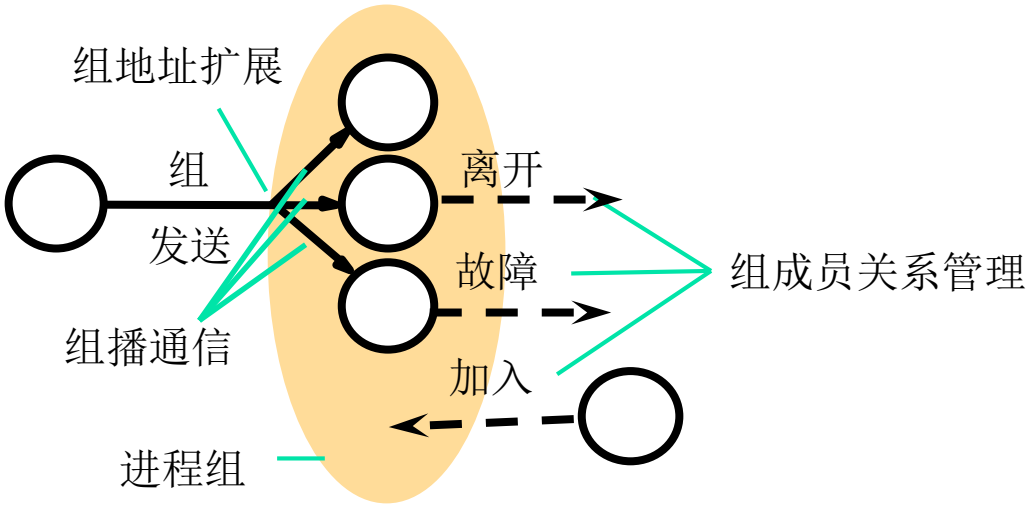
- 副本管理器
  - 接收前端请求
  - 对副本执行原子性操作
- 前端
  - 接收客户请求
  - 通过消息传递与多个副本管理器进行通信

系统模型

- 副本对象的操作
  - 请求：前端将请求发送至一个或多个副本管理器
  - 协调
    - 保证执行的一致性
    - 对不同请求进行排序
      - FIFO序、因果序、全序
  - 执行：包括临时请求的执行
  - 协定：就提交请求的影响达成一致
  - 响应：一个或多个副本管理器响应前端

组通信的作用

- 管理复制数据中组的作用
  - 在系统执行过程中，进程可以加入和离开组
  - 组成员关系服务——管理动态组成员关系
  - 组播通信的完整实现
- 进程组提供的服务



组通信的作用

- 组成员关系服务的作用
  - 为组成员关系变更提供接口
    - 创建或撤消进程组
    - 在组中加入或删除某个进程
  - 实现故障检测器
    - 检测组中成员，给进程添加“可疑”或“不可疑”标示
    - 删除被怀疑发生了故障或通信不可达的进程
  - 通告组成员关系变更
  - 负责组地址的扩展
    - 进程通过组标示符而非进程列表来组播消息

组通信的作用

- 视图传递
  - 组视图
    - 由进程标识符标识的当前组成员的列表
  - 传递一个视图
    - 组成员发生变化时，将新成员关系告知给应用
    - 组管理服务将一系列的视图传递给组中的每个进程
  - 传递视图和接收视图截然不同
    - 视图首先存放在保留队列上，直到所有组成员同意进行传递为止。

组通信

视图传递的基本要求

- 顺序
  - 如果一个进程p先后传递视图v(g)、v'(g)，那么不存在进程q(q≠p)，q在v(g)之前传递v'(g)。
- 完整性
  - 如果进程p传递视图v(g)，那么p∈v(g)。
- 非平凡性
  - 如果进程q加入到一个组中，且对进程p≠q可达，则q总是在p发送的视图中。
  - 如果组被分割并形成分区，且分区仍然存在，那么最终任何分区所传递的视图将不包含其他分区中的进程。

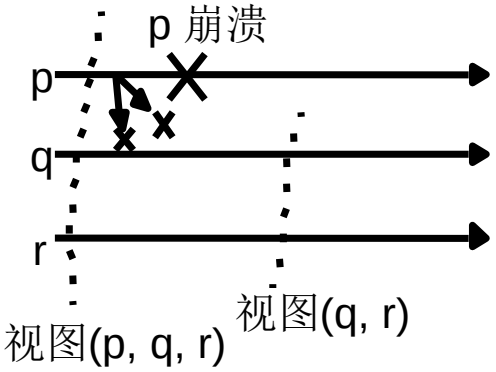
组通信

视图同步的组通信

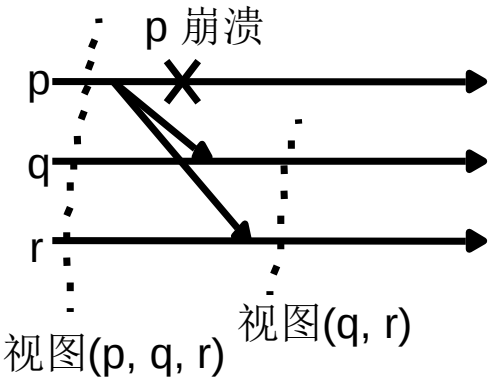
- 协定
  - 如果一个正确的进程在视图v(g)中传递了消息m，那么所有其它传递消息m的正确的进程都在视图v(g)中传递m。
- 完整性
  - 如果进程p传递了消息m，那么p不会再传递m。
- 有效性(封闭组)
  - 正确的进程总是传递它们发送的消息。

组通信中的视图

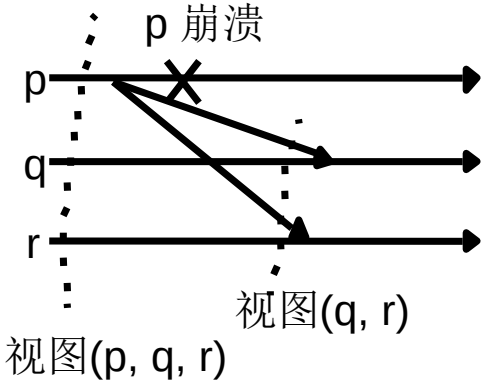
组通信中的视图同步示例



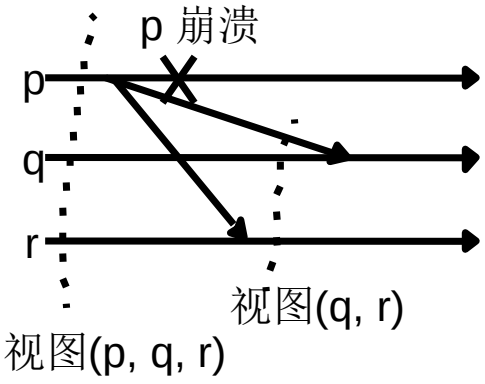
a 允许



b 允许



c 不允许（违反有效性）



d 不允许（违反协定）

第8章 复制

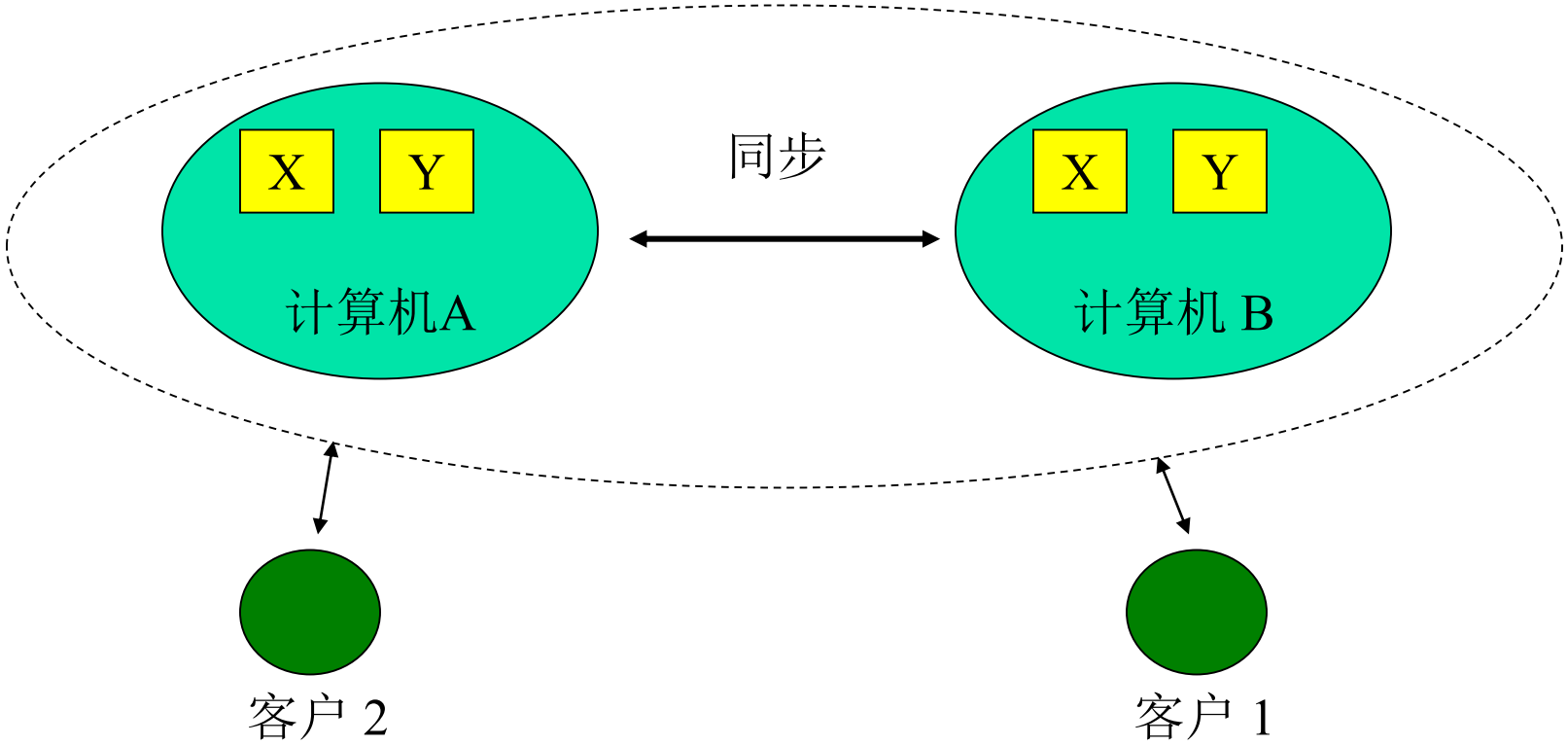
- 简介
- 系统模型和组通信
- 容错服务
- 高可用服务的实例研究：gossip体系结构、Bayou和Coda
- 复制数据上的事务
- 小结

容错服务

- 复制是提高系统容错能力的有效手段之一
  - 为用户提供一个单一的镜像
  - 副本之间需要保持严格的一致性
- 副本之间的不一致性将导致容错能力失效
  - 银行帐户示例

容错服务

1. 银行帐户x和y的两个副本管理器位于计算机A、B上
2. 客户在本地的副本管理器上读取和更新帐户



容错服务

客户1:

$setBalance_B(x,1)$

Server B failed…

$setBalance_A(y,2)$

客户2:

$getBalance_A(y)=2$

$getBalance_A(x)=0$

$setBalance_B(x,1)$没成功，由于B在把帐户x的更新传送至A前出现故障，所以产生了不一致现象

y=2发生在x = 1之后，读到y=2，x = 0

容错服务——线性化能力

- 交错序列操作
  - 两个客户的复制服务实现
  - 客户i的操作：$o_{i0}, o_{i1}, o_{i2}, \ldots$
  - 单个副本的交错序列：$o_{20}, o_{21}, o_{10}, o_{22}, o_{11}, o_{12}, \ldots$
- 线性化准则
  - 操作的交错执行序列符合对象的(单个)正确副本所遵循的规约。
  - 操作的交错执行次序和实际运行中的（请求）次序实时一致。
- 线性化能力是最严格的一致性需求，难以实现
  - 上例中，引起银行帐户客户执行的服务不是可线性化的

容错服务

顺序一致性

- 比线性化能力较弱的一致性条件
- 顺序一致性准则

  - 操作的交错序列符合对象的(单个)正确副本所遵循的规范。

  - 操作在交错执行中的次序和在每个客户程序中执行的次序一致。
- 每个可线性化服务都是顺序一致的，反之不成立。
- 示例

容错服务

客户1	客户2
$setBalance_B(x,1)$	
	$getBalance_A(y) \rightarrow 2$
	$getBalance_A(x) \rightarrow 0$
$setBalance_A(y,2)$	

不满足线性化能力，但满足顺序一致性（不涉及绝对时间）能力

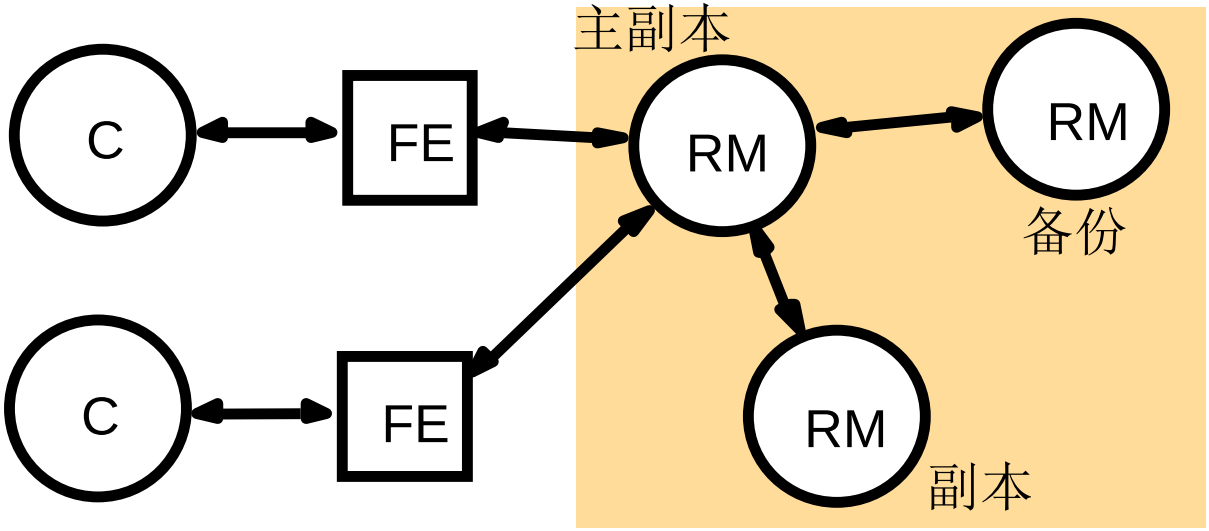
容错服务

被动(主备份)复制

- 一个主副本管理器＋多个次副本管理器

  - 若主副本管理器出现故障，则某个备份副本管理器将提升为主副本管理器。

- 模型



容错服务

- 被动复制时的事件次序
  - 请求
    - 前端将请求发送给主副本管理器
  - 协调
    - 主副本管理器按接收次序对请求排序
  - 执行
    - 主副本管理器执行请求并存储响应

容错服务

- 被动复制时的事件次序(续)
  - 协定
    - 若请求为更新操作，则主副本管理器向每个备份副本管理器发送更新后的状态、响应和唯一标识符。
    - 备份副本管理器返回确认。
  - 响应
    - 主副本管理器将响应发送给前端
    - 前端将响应发送给客户

容错服务

- 被动复制的线性化能力
  - 主副本管理器正确运行

    - 共享对象的操作顺序化⟶线性化能力
  - 主副本管理器发生故障时的线性化能力

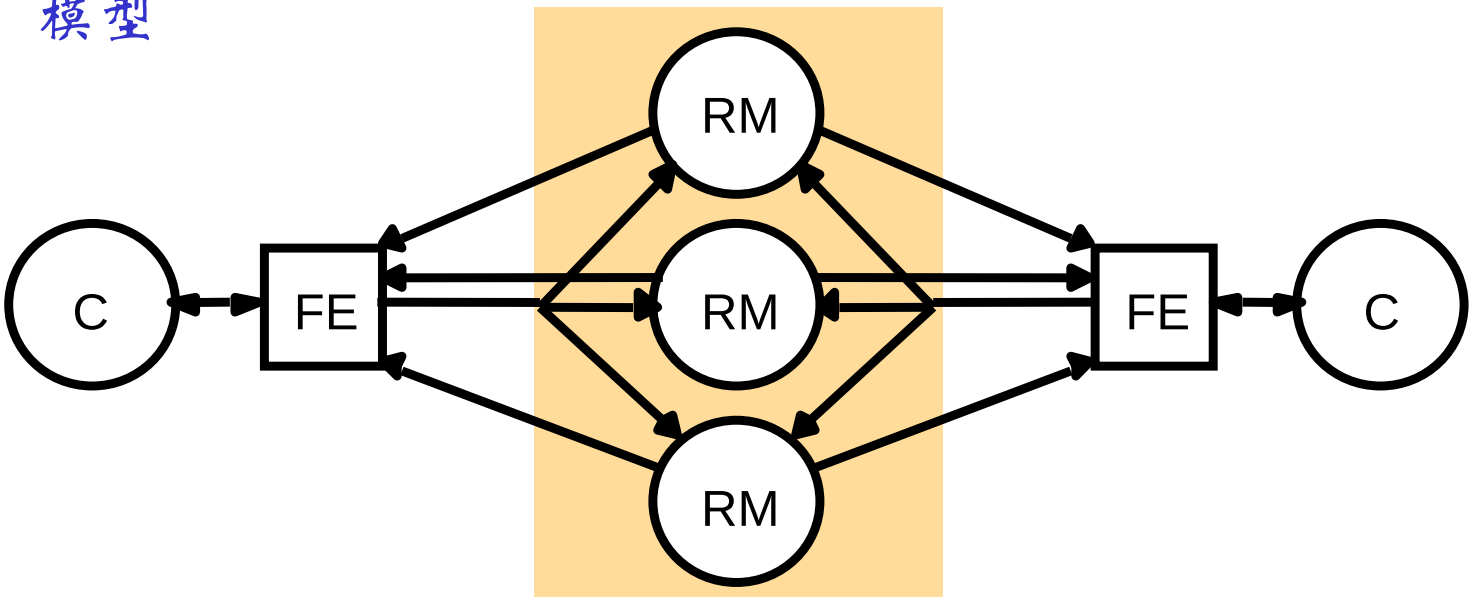
    - 被唯一地备份副本管理器代替

    - 剩余的备份服务器在哪些操作已被执行上达成一致。
  - 实现线性化能力的方法

    - 副本管理器组织为一个组，并且主副本管理器使用视图同步组通信发送更新到备份。

容错服务

- 主动复制
  - 副本管理器地位对等，前端组播消息至副本管理器组
  - 模型



容错服务

主动复制时的事件次序

- 请求
  - 前端使用全序、可靠的组播原语将请求组播到副本管理器组
- 协调
  - 组通信系统以同样的次序(全序)将请求传递到每个副本管理器
- 执行
  - 每个副本管理器以相同的方式执行请求
- 响应
  - 每个副本管理器将响应发送给前端
  - 前端将响应发送给客户

容错服务

- 主动复制的顺序一致性
  - 可靠性组播
    - 保证每个副本管理器处理同样的请求集合
  - 全序
    - 保证每个副本管理器以同样的顺序处理请求集合
  - FIFO顺序
    - 每个前端的请求以FIFO的顺序处理，与“程序的顺序”一致
- 主动复制不能保证线性化

  副本管理器处理请求的全序与客户发生这些请求的实时次序不一定相同。

第8章 复制

- 简介
- 系统模型和组通信
- 容错服务
- 高可用服务的实例研究：gossip体系结构、Bayou和Coda
- 复制数据上的事务
- 小结

高可用服务的实例研究

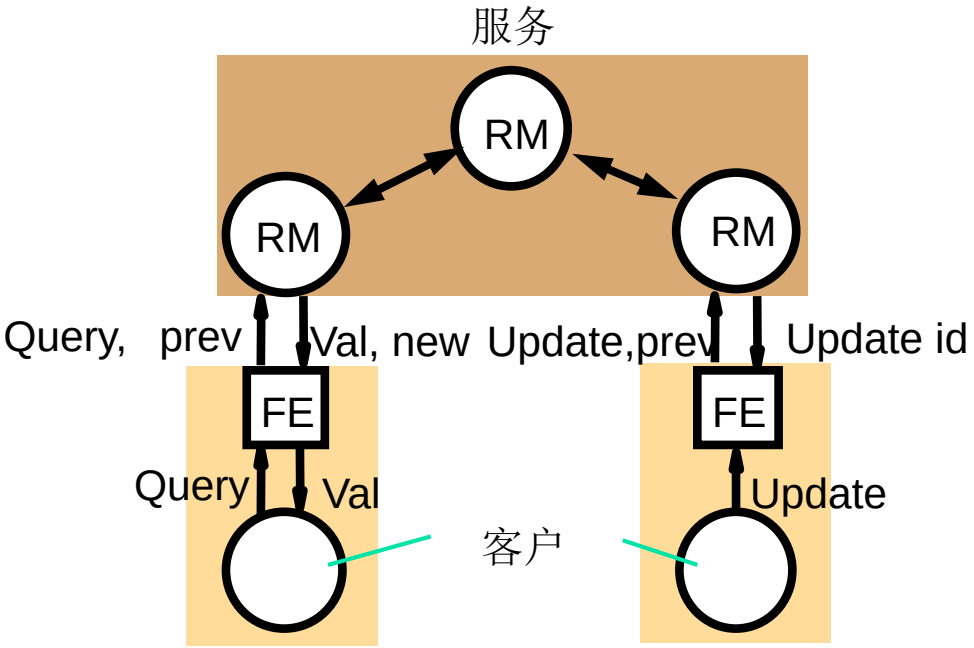
高可用性和容错能力

- 容错能力
  - 只要可能，所有正确的副本管理器都能够及时收到更新，并在将控制传递回客户以前达成一致。
- 高可用性
  - 采用较弱程度的一致性，提高共享数据的可用性。
  - 实例：gossip、Bayou和Coda

gossip体系结构

体系结构

- 前端可以选择任意副本管理器
- 提供两种基本操作：*查询*+*更新*
- 副本管理器定期通过gossip消息来传递客户的更新



gossip体系结构

体系结构(续)

- 系统的两个保证
  - 随着时间的推移，每个用户总能获得一致服务
    - 副本管理器提供的数据能反映迄今位置客户已经观测到的更新
  - 副本之间松弛的一致性
    - 所有副本管理器最终将收到所有更新
    - 两个客户可能会观察到不同的副本
    - 客户可能观察到过时数据

gossip体系结构

查询和更新操作流程

- 请求
  - 前端将请求发送至一个副本管理器
    - 查询：客户可能阻塞
    - 更新：无阻塞
    - 无响应：换另一个
- 对更新操作的响应
  - 副本管理器立即应答收到的更新请求
- 协调
  - 收到请求的副本管理器并不处理操作，直到它能根据所要求的次序约束处理请求为止。

gossip体系结构

查询和更新操作流程(续)

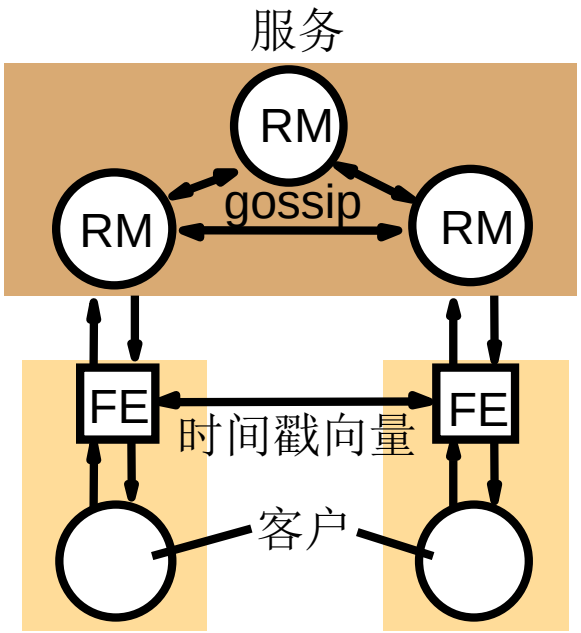
- 执行
  - 副本管理器执行请求
- 对查询操作的响应
  - 副本管理器对查询请求作出应答
- 协定
  - 副本管理器通过交换gossip消息进行相互更新
    - gossip消息的交换是偶尔的
    - 发现消息丢失后，才和特定的副本管理器交换消息

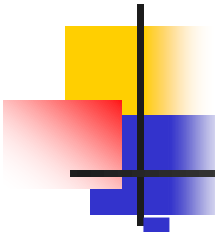
gossip体系结构

- 前端的版本时间戳
  - 客户交换数据

    -通过访问相同的gossip服务

    - 直接通信



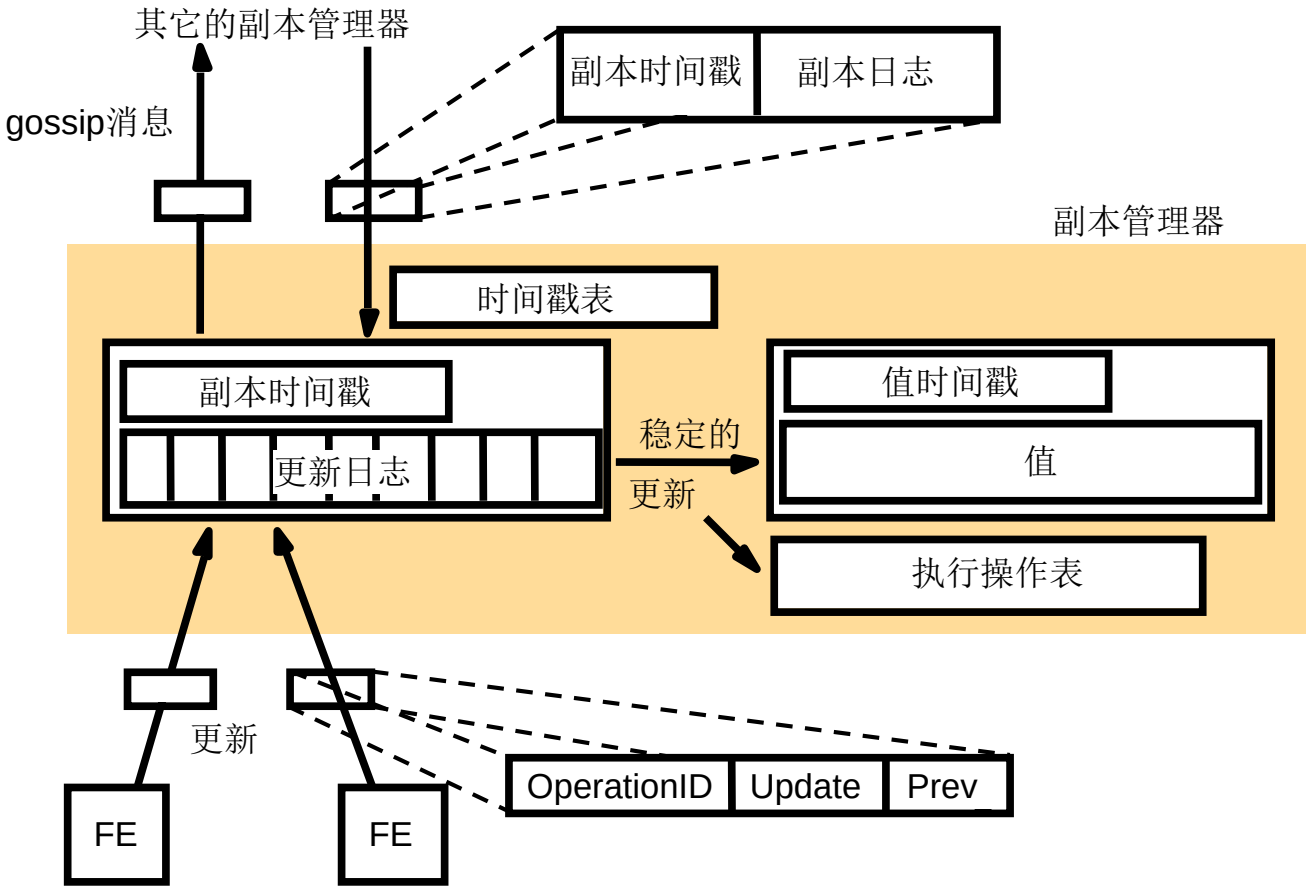


gossip体系结构

- 前端的版本时间戳(续)
  - 每个前端维护一个向量时间戳
    - 每个副本管理器有一条对应的记录
    - 更新或查询信息中包含时间戳
    - 合并操作返回的时间戳与前端时间戳
  - 向量时间戳的作用
    - 反映前端访问的最新数据值

gossip体系结构

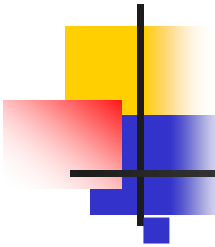
副本管理器状态



gossip体系结构

副本管理器状态(续)

- 值
  - 副本管理器维护的应用状态的值
- 值的时间戳
  - 更新的向量时间戳
- 更新日志
  - 记录更新操作
- 副本的时间戳
  - 已经被副本服务器接收到的更新



gossip体系结构

副本管理器状态(续)

- 已执行操作表
  - 记录已经执行的更新的唯一标识符，防止重复执行
- 时间戳表
  - 确定何时一个更新已经应用于所有的副本管理器

gossip体系结构

查询操作

- 副本管理器收到查询
  - q.pre≤valueTS
    - 立即响应
    - 返回消息中的时间戳为valueTS
  - 否则
    - 将消息保存到保留队列
- 前端收到查询响应
  - 合并时间戳：frontEndTS:=merge(frontEndTS,new)

gossip体系结构

- 按因果次序处理更新
  - 前端发送更新请求：(u.op, u.prev, u.id)
  - 副本管理器i接收请求
    - 丢弃：操作已经处理过
    - 否则，将更新记录日志
      - ts =u.prev, ts[i]=ts[i]+1
      - logRecord= <i, ts, u.op, u.prev, u.id>
    - 副本管理器将ts返回给前端
      - frontEndTS=merge(frontEndTS, ts)

gossip体系结构

- 按因果次序处理更新(续)
  - 更新请求u的稳定性条件

    u.prev≤valueTS

    说明更新值依赖的所有更新已经执行了
  - 副本管理器的更新操作

    - value := apply(value, r.u.op)
    - valueTS := merge(valueTS, r.ts)
    - executed := executed ∪ {r.u.id}

gossip体系结构

- gossip消息交换
  - 副本管理器更新自身状态
  - 通过时间戳表估计丢失的更新
- gossip消息格式
  - 日志m.log
  - 副本时间戳m.ts

gossip体系结构

- 收到gossip消息后执行的操作
  - 日志合并
    - 若r.ts ≤replicsTS，则丢弃
    - 将记录加入到日志，合并时间戳

      replicaTS := merge(replicaTS, m.ts)
  - 执行稳定的更新

    - 根据向量时间戳的偏序“≤”对更新进行排序，并依次执行更新。
  - 从日志和已执行操作表中删除冗余记录

    - 若tableTS[i][c] ≥ r.ts[c]，则丢弃r。

    表明任何一个副本管理器都已收到的更新

gossip体系结构

- 更新传播
  - gossip体系结构未规定具体的更新传播策略
  - 如何选择合适的gossip消息的发送频率？
    - 分钟、小时或天？——*由具体应用需求决定*
  - 如何选择合适的合作者(副本管理器)
    - 随机策略

      使用加权概率来选择更合适的合作者
    - 确定策略

      使用副本管理器状态的函数来选择合作者
    - 拓扑策略

      网格、环、树

gossip体系结构

讨论

- 目标

  保证服务的高可用性，代价为遵守松弛的一致性。

- 缺点

  - 不适合接近实时的更新复制

    由更新传播的惰性特点决定

  - 可伸缩性差

    若采用因果序更新，R各副本管理器收集G个更新，则交换的消息数量为：$2+(R-1)/G$

Bayou系统和操作变换方法

Bayou系统简介

- 与gossisp体系结构和基于时间戳的反熵协议类似
- 提供的一致性保证弱于顺序一致性
- 能够进行冲突检测和冲突解决

  - 依靠行业准则等解决冲突

  - 操作变换：一个或多个相冲突的操作被取消或改变以解决冲突的过程。

  - 例子，行政主管和秘书同时预约，其中行政主管为离线更新——行政主管上线后，Bayou系统检测到冲突，然后批准行政主管的预约而取消秘书的预约。

Bayou系统和操作变换方法

临时更新和提交的更新

- 临时的更新

  更新首次应用于数据库时，被标记为临时的。

- 提交的更新

  Bayou将临时的更新以规范次序放置，并添加提交标识。

- 数据库副本状态

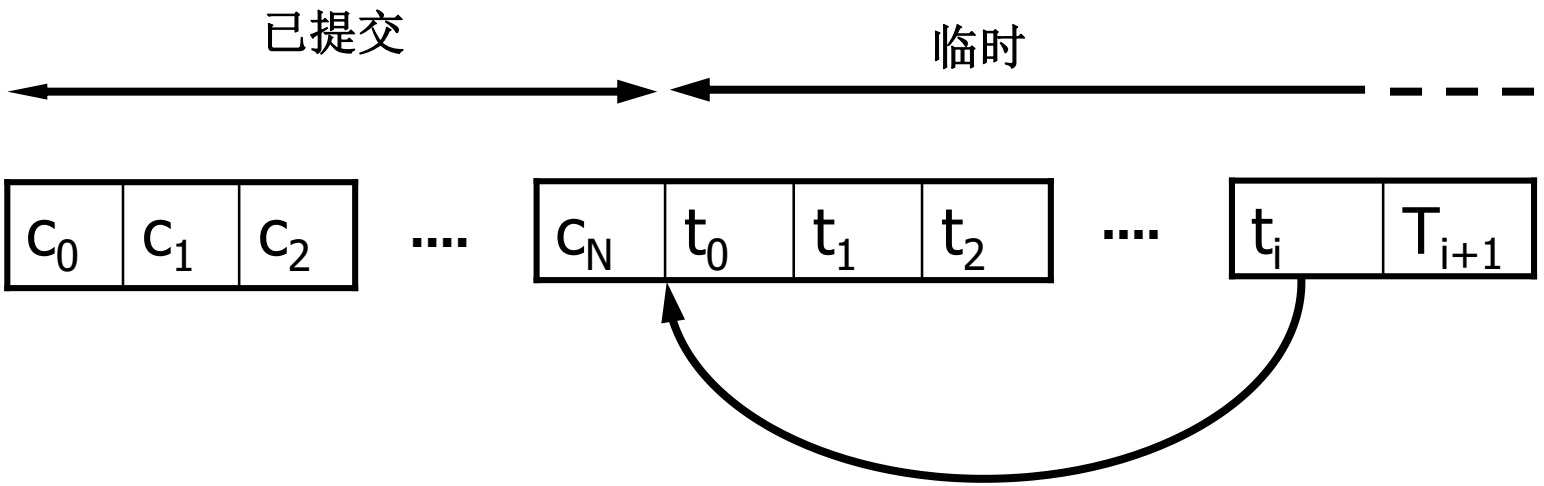
  提交的更新序列＋临时的更新序列

- 更新重排序

  新更新到达或某个临时更新被修改为提交的更新

Bayou系统和操作变换方法

临时更新和提交的更新示例



临时更新t_i成为下一个提交更新，

并被插入到最新提交更新c_N之后

Bayou系统和操作变换方法

- 依赖检查和合并过程
  - 依赖检查
    - 一个更新执行时是否会产生冲突
    - 例子：写－写冲突、读－写冲突检测
  - 合并过程
    - 改变将要执行的操作，避免冲突，并获得相似效果
    - 无法合并→系统报错
- Bayou系统讨论
  - 复制对于应用而言是不透明的
  - 利用应用语义知识来提高数据的可用性。程序员需要依赖检查和合并，复杂度高

Coda文件系统

- AFS的主要缺陷

  只读卷的复制

- Coda目标

  稳定的数据可用性

- Coda对AFS的扩展
  - 读-写副本
  - 采用文件卷复制技术——提高吞吐率和容错性
  - 在客户计算机上缓存文件副本——断链处理

Coda文件系统

Coda体系结构

- Venus/Vice进程
  - Venuc：前端和副本管理器的混合体
  - Vice：副本管理器
- 卷存储组(VSG)

  持有一个文件卷副本的服务器集合
- 可用的卷存储组(AVSG)

  打开一个文件的客户能访问的VSG的某个子集

Coda文件系统

Coda体系结构(续)

- 文件访问过程
  - 当前AVSG中的任何一个服务器提供文件的缓存拷贝给客户计算机
  - 对每个副本进行更新分布
- 关闭文件
  - 修改过的拷贝并行广播到AVSG中的所有服务器
  - 允许用户在网络分区的情况下更新数据
  - 网络分区修复后，更新其它服务器

    副本冲突检测、解决

Coda文件系统

Coda体系结构(续...)

- 断链操作
  - AVSG为空时，客户缓存文件
  - 手工干预解决冲突
- 设计原则
  - 服务器上的拷贝比客户计算机缓存中的拷贝更可靠

Coda文件系统

复制策略

- 乐观策略
  - 在网络分区和断链操作期间，仍然可以进行文件修改
- 实现
  - Coda版本向量(CVV)
    - 作为时间戳添加在每个版本的文件上
    - 服务器上文件修改次数的估计
  - 以CVV为依据，检测潜在冲突
  - 手工干预解决冲突

Coda文件系统

复制策略(续)

- 冲突检测

  - 若一个战点的CVV大于或等于所有其它战点相应的CVV，则不存在冲突。→*自动更新*

  - 若对于两个CVV而言，$V_1 \geq V_2$与$V_2 \geq V_1$均不成立，则存在一个冲突。 →*手工干预*

Coda文件系统

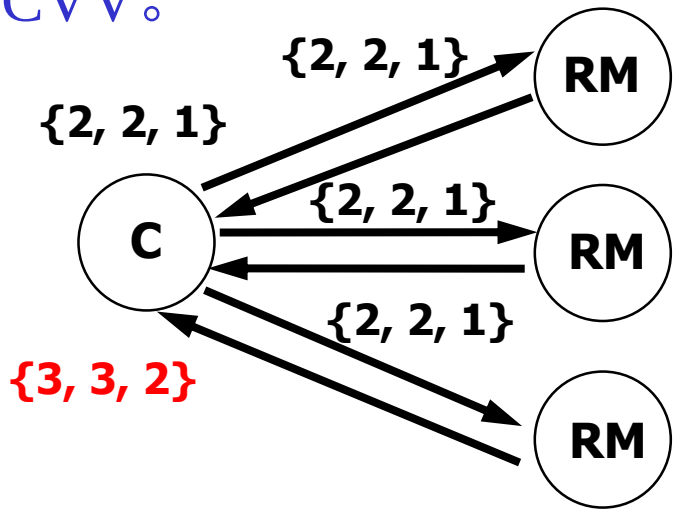
复制策略(续.)

- 文件修改

  - Venus进程发送更新消息

  - AVSG中的每个服务器更新文件，并返回确认。

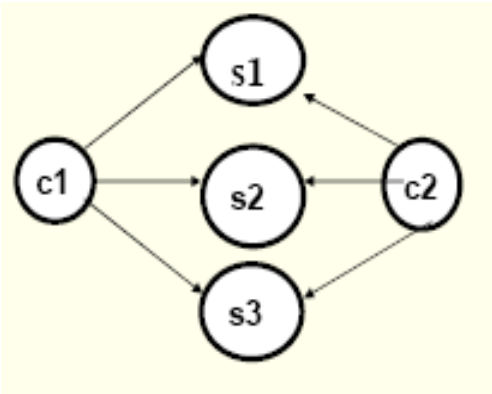
  - Venus计算新的CVV，增加相应服务器的修改记数，并分发新的CVV。

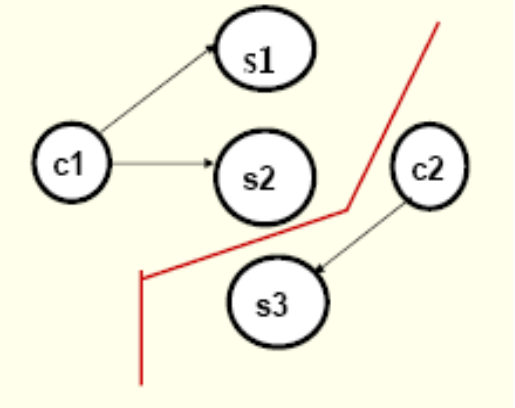


Coda文件系统

构建CVV示例一

- 文件F在三个服务器S_1、S_2和S_3上有副本
  - VSG = {S_1, S_2, S_3}
  - F在被C_1修改
  - C_1: {S_1, S_2}； C_2: {S_3}

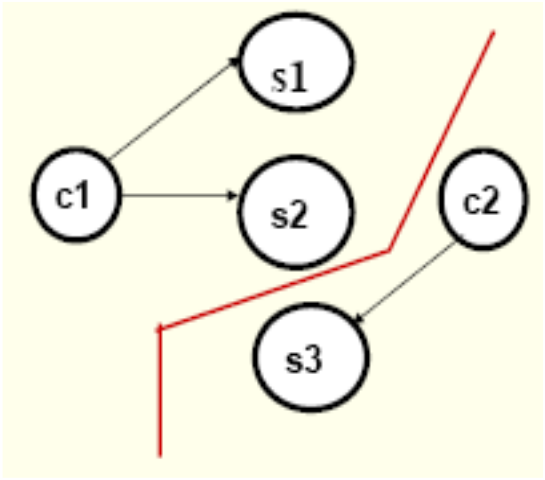




Coda文件系统

构建CVV示例一(续)

- 最初，F的CVV在3个服务器上是相同的，比如[1, 1, 1]
- C_1修改F，然后关闭
  - S_1和S_2上的CVV更新为：[2, 2, 1]



Coda文件系统

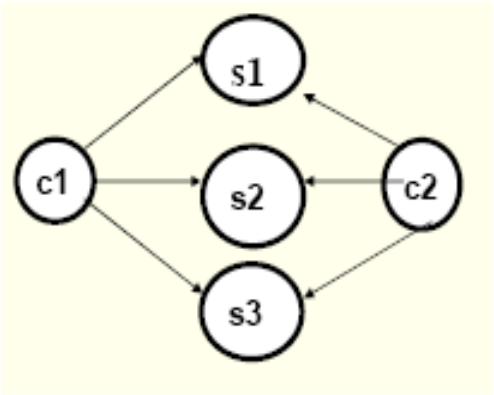
构建CVV示例一(续.)

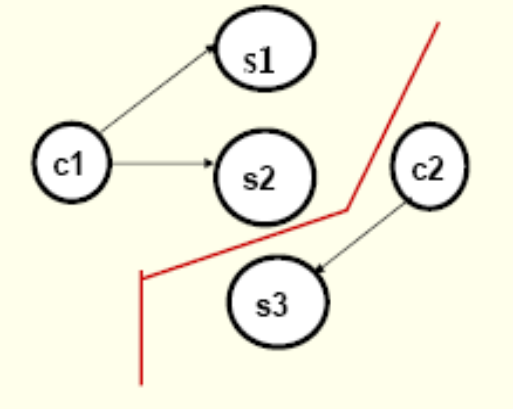
- 网络故障修复后
  - S_1和S_2上的CVV为[2, 2, 1]，S_3上的CVV为[1, 1, 1]
  - S_1或S_2上的文件版本替代S_3上的文件版本

Coda文件系统

构建CVV示例二

- 文件F在三个服务器S_1、S_2和S_3上有副本
  - VSG = {S_1, S_2, S_3}
  - F在同一时间被C_1和C_2修改
  - C_1: {S_1, S_2}； C_2: {S_3}

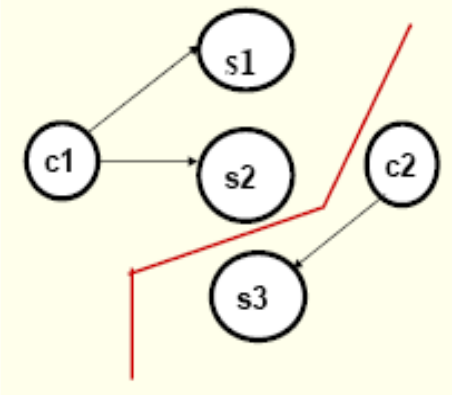




Coda文件系统

构建CVV示例二(续)

- 最初，F的CVV在3个服务器上是相同的，比如[1, 1, 1]
- C_1修改F，然后关闭
  - S_1和S_2上的CVV更新为：[2, 2, 1]
- 同时，C_2上的两个进程分别修改F，然后关闭
  - S_3上的CVV更新为：[1, 1, 3]



Coda文件系统

构建CVV示例二(续.)

- 网络故障修复后
  - S_1和S_2上的CVV为[2, 2, 1]，S_3上的CVV为[1, 1, 3]
  - 存在冲突，需手工干预

第8章 复制

- 简介
- 系统模型和组通信
- 容错服务
- 高可用服务的实例研究：gossip体系结构、Bayou和Coda
- 复制数据上的事务
- 小结

复制数据上的事务

- **单拷贝串行化**

  作用于复制对象的事务和它们在某一对象集上执行具有一样的效果。

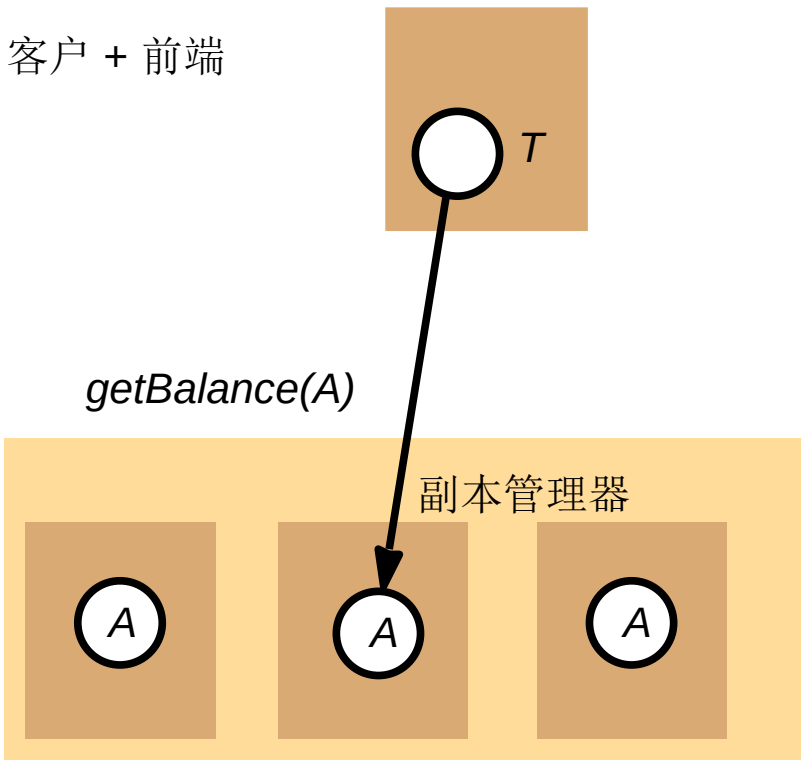
- **体系结构**
  - 一个客户请求能否寻址到某个副本管理器
  - 为了成功完成一个操作需要多少副本管理器
  - 如何放弃提交
- **三种复制方案**
  - 带验证的可用拷贝
  - 法定数共识 （意味着一个法定组有足够的成员）
  - 虚拟分区

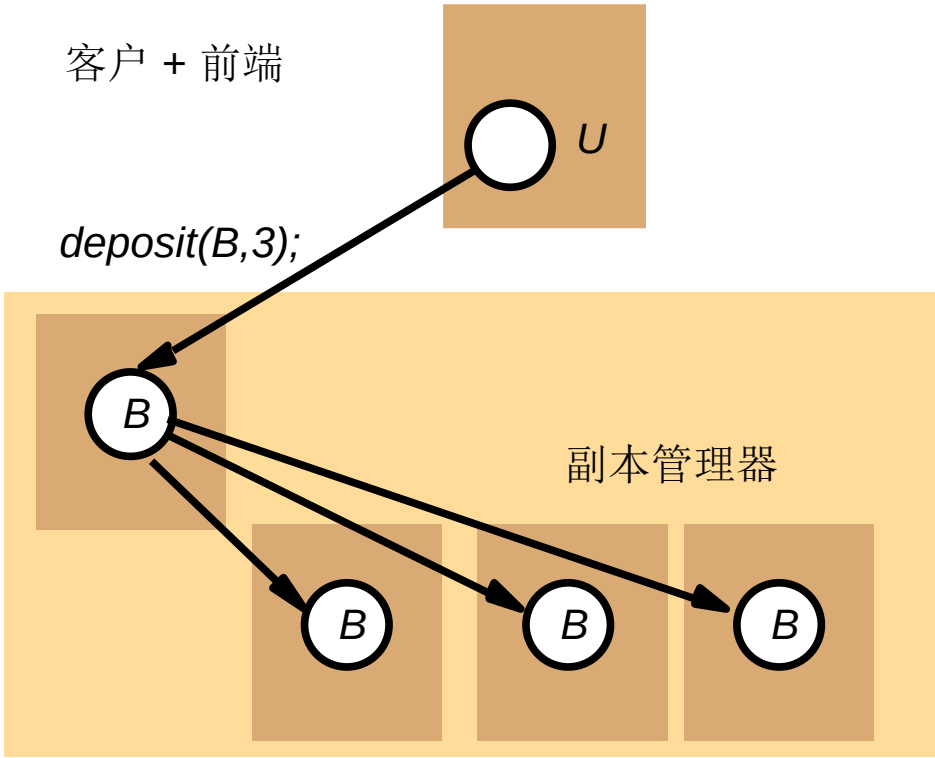
复制事务的体系结构

- 主拷贝
  - 所有前端和一个“主”副本管理器通信来执行某个操作
  -“主”副本管理器负责更新备份
- 副本管理器的协调
  - 读一个/写所有方案
  - 法定数共识方案
- 更新传播
  - 惰性方法
  - 及时方法

复制事务的体系结构

- 复制数据上的事务示例：读一个/写所有方案





复制事务的体系结构

- 两阶段提交协议
  - 两层嵌套的两阶段提交协议
    - 第一阶段
      - 协调者发送 “canCommit”给参与者
      - 参与者传递给其它副本管理器，并收集应答
      - 参与者回答协调者
    - 第二阶段
      - 协调者发送 “doCommit”或“doAbort”给参与者
      - 参与者传递给其它副本管理器

复制事务的体系结构

- 读一个/写所有
  - 简单的复制方案
  - 单拷贝串行化的实现
    - 写锁

      在操作影响的每个对象上加写锁
    - 读锁

      在操作影响的对象上加读锁
  - 可能发生死锁

数据上的事务复制方法

- 可用拷贝复制
  - 读一个/写所有方案不现实

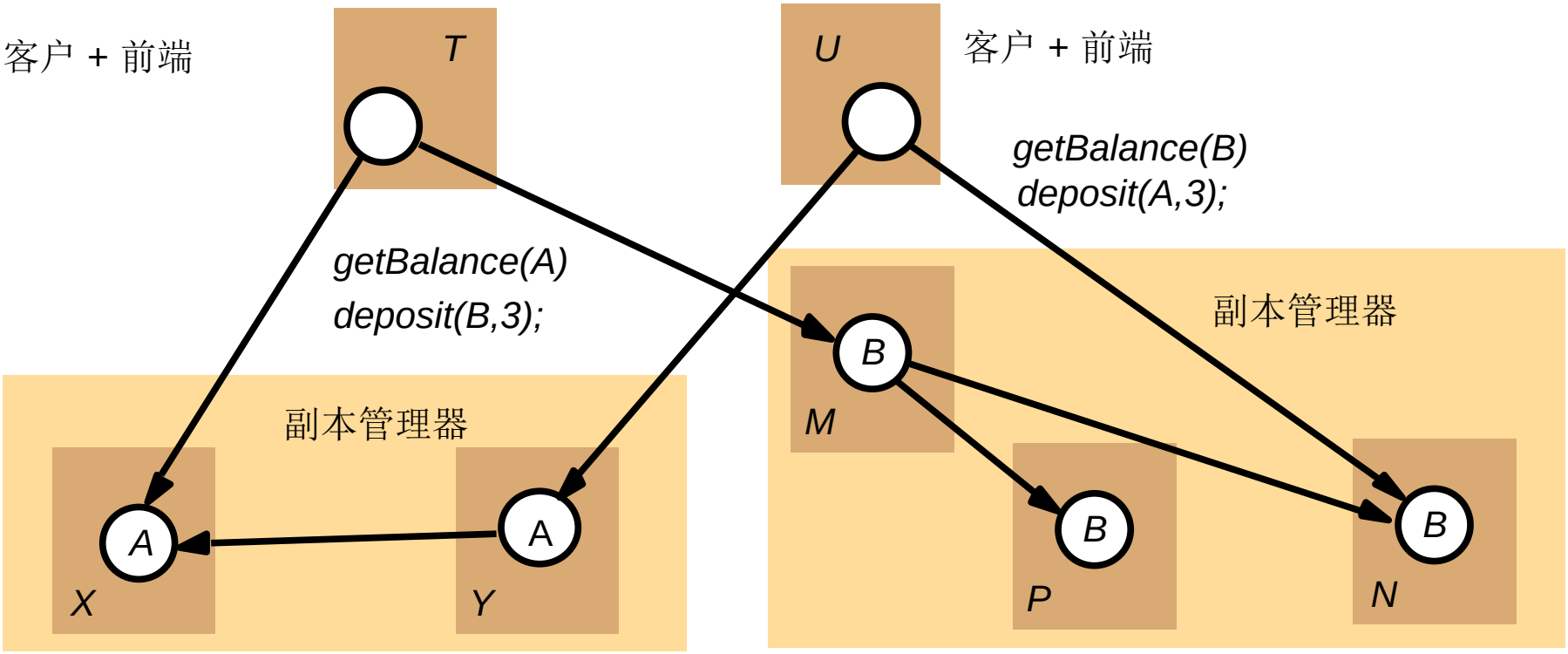
    某些副本管理器有时不可用
  - 可用拷贝复制方案

    - 允许某些副本管理器暂时不可用

    - read/write

      - read请求：由接收到请求的任意副本管理器执行
      - write请求：由接收到请求的副本管理器和组中可用的副本管理器执行
  - 示例

可用拷贝复制


客户 + 前端
T
U
客户 + 前端
getBalance(B)
deposit(A,3);
getBalance(A)
deposit(B,3);
副本管理器
副本管理器
B
M
A
A
B
B
X
Y
P
N

可用拷贝复制

- 副本管理器故障
  - 管理器崩溃导致数据的不一致性

    副本管理器在事务执行期间崩溃
  - 示例(参见上页图例)

    - X在T执行getBalance后发生故障
    - N在U完成getBalance后发生故障
    - N和U发生故障在T和U执行deposit前

后果：副本管理器**X**上对**A**的并发控制并不会阻止事务**U**在副本管理器**Y**上更新**A**；

副本管理器**N**上对**B**的并发控制并不会阻止事务**T**在副本管理器 **M**和**P**上更新**B**。

- *对象拷贝的本地并发控制不能保证单拷贝串行化*

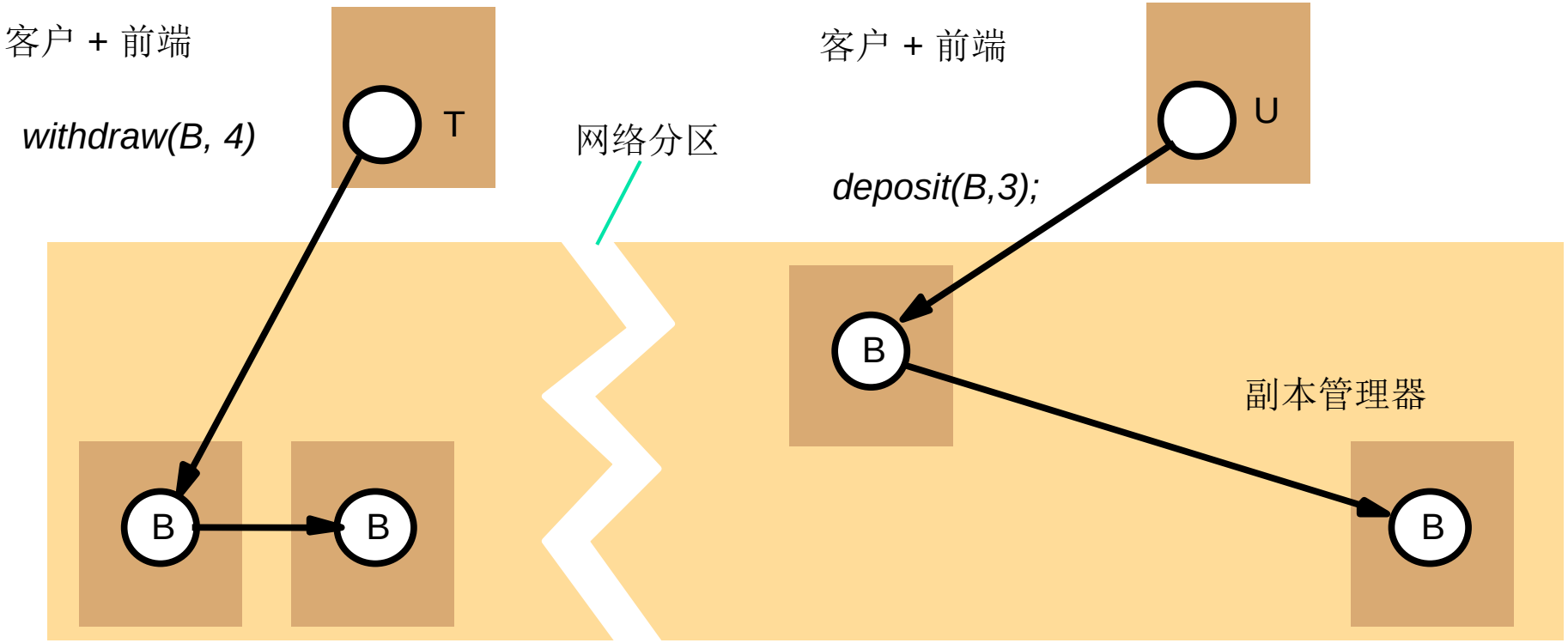
可用拷贝复制

- 本地验证
  - 额外的并发控制

    确保任何故障或恢复事件不会在事务的执行过程中发生
  - 示例
    - 事务T：
      - N出现故障→T在X读对象A
      - T在M和P上写对象B →T提交→X出现故障
    - 事务U：
      - X出现故障→U在N读对象B
      - U在Y上写对象A →U提交→N出现故障
    - 发现冲突：若T首先验证，则提交T，放弃U；反之依然。

数据上的事务复制方法

- 网络分区
  - 将副本管理器分成多个子组



网络分区

- 假设

  网络分区最终将被修复

- 处理要求

  不会产生不一致数据

- 处理方法

  - 乐观方法

    允许在所有的分区进行更新，在分区修复后解决不一致问题。

  - 悲观方法

    阻止在分区时任何不一致的产生，在分区修复后进行更新对象的拷贝。

数据上的事务复制方法

- 法定数共识方法
  - 阻止不一致产生的方法
    - 冲突操作只能在一个分区中执行
    - 法定数

      若干个副本管理器的子组，它的大小使它能够执行某个操作。

法定数共识方法

- Gifford算法
  - 选票

    一个对使用特定拷贝的需求度的权重
  - 法定数方案

    - 读法定数

      read操作前必须获得一个有R个选票的读法定数

    - 写法定数

      write操作前必须获得一个有w个选票的读法定数

    - 满足W > 总选票的一半
    - R + W > 组选票总数

    - 分区出现时，不可能在不同的分区中进行同一拷贝上冲突操作。

法定数共识方法

- 副本管理器组的配置能力
  - 提供不同的性能或可靠性
    - 降低W(或R)，增加write (或read)操作的性能
    - 增加W(或R)，增加write (或read)操作的可靠性
  - 弱代表性
    - 客户机上的文件拷贝
    - Vote = 0
    - 一旦某个读法定数构造成功，若文件的本地拷贝是最新的，则可以在该拷贝上执行read操作

法定数共识方法

- Gifford的例子
  - 例1
    - 高读写率
    - 复制用来提高系统的性能而非可靠性
  - 例2
    - 中读写率
    - 读可以在本地副本上执行
    - 写必须访问本地副本管理器和一个远程副本管理器
  - 例3
    - 非常高的读写率
    - 读一个/写所有

Gifford的例子

		例 1	例 2	例 3
延迟 (ms)	副本 1	75	75	75
	副本2	65	100	750
	副本3	65	750	750
选票配置	副本1	1	2	1
	副本2	0	1	1
	副本3	0	1	1
法定数大小	R	1	2	1
	W	1	3	3
文件包得到的性能				
读	延迟	65	75	75
	阻塞概率	0.01	0.0002	0.000001
写	延迟	75	100	750
	阻塞概率	0.01	0.0101	0.03

网络分区情况下的事务复制方法

- 虚拟分区算法
  - 法定数共识＋可用拷贝

    -法定数共识：出现分区时能正确工作

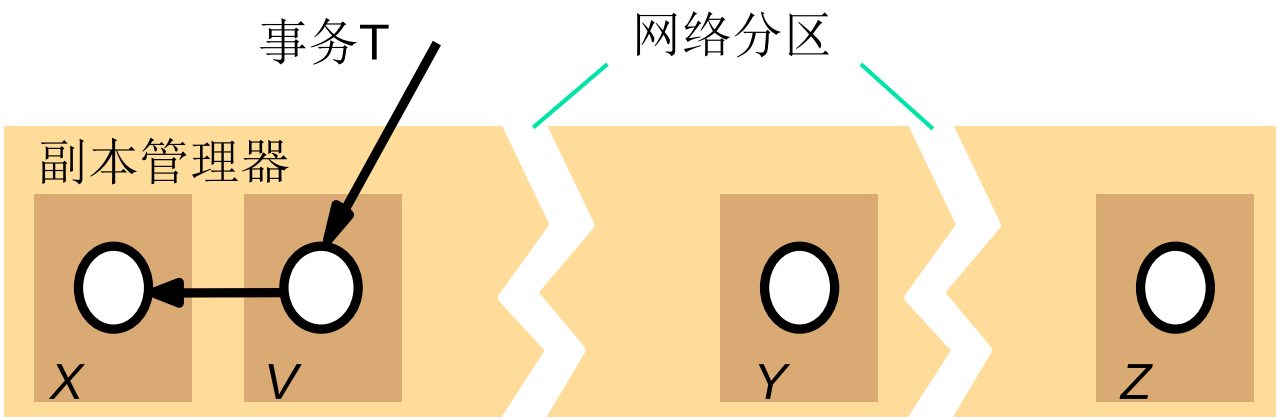
    - 可用拷贝：read操作的代价低
  - 虚拟分区

    - 分区包含足够的副本管理器以满足法定数需求

    - 在虚拟分区内执行可用拷贝算法

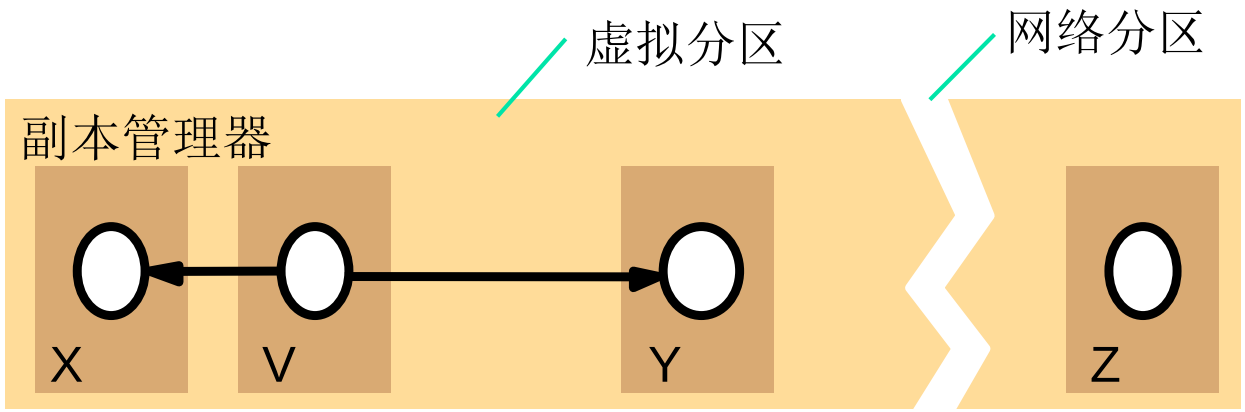
虚拟分区算法示例

- 4个副本管理器V、X、Y和Z，R = 2，W = 3
- 最初，所有副本管理器相互连接，使用可用拷贝算法
- 事务T：V的read＋4个服务器的write
- 网络在read执行后发生分区



虚拟分区算法示例(续)

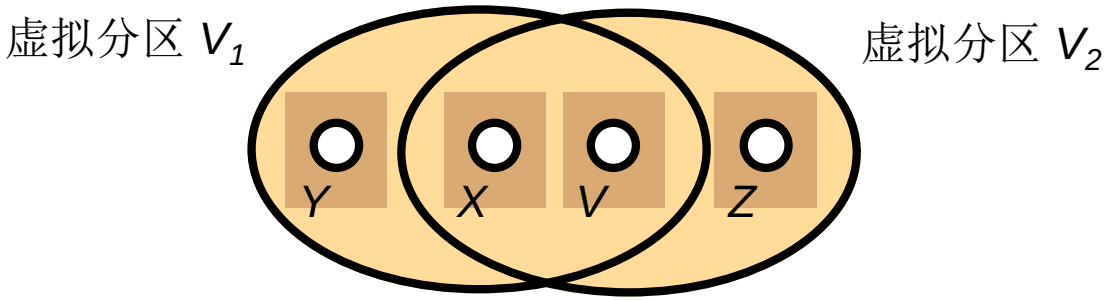
- 创建虚拟分区
  - V不断试图连接Y和Z，直至它们中的一个或全部回应为止
  - V、X和Y组成了一个虚拟分区
  - 在虚拟分区执行可用拷贝算法



虚拟分区的实现

- 重叠的虚拟分区
  - Y和Z同时创建虚拟分区
  - 冲突

在虚拟分区V_1中的事务在Y上设置的读锁不会和另一个虚拟分区事务的写操作设置的写锁相冲突，从而违背单拷贝串行化。



虚拟分区的实现(续)

- 协议
  - 采用逻辑时间戳
  - 算法

创建一个虚拟分区

阶段 1:

- 发起者发送一个Join请求给每个潜在成员。Join的参数是一个用于新虚拟分区的逻辑时间戳。
- 当某个副本管理器收到Join请求后，它比较请求的时间戳和自己当前虚拟分区的时间戳。
  - 如果请求中的逻辑时间戳大，那么它同意加入并回复Yes;
  - 如果请求中的逻辑时间戳小，那么它拒绝加入并回复 No.

阶段 2:

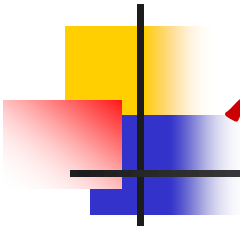
- 如果发起者收到了足够的Yes应答，从而获得读和写法定数，那么它通过发送一个 Confirmation消息给同意加入的站点来创建一个新的虚拟分区。该虚拟分区的创建时间戳和实际成员列表以参数形式发送。
- 收到Confirmation消息的副本管理器加入新虚拟分区，并记录它的创建时间戳和实际成员列表。

第8章 复制

- 简介
- 系统模型和组通信
- 容错服务
- 高可用服务的实例研究：gossip体系结构、Bayou和Coda
- 复制数据上的事务
- 小结

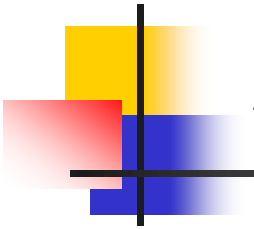
小结

- 分布式系统的复制
  - 增强性能、提高可用性和容错能力
- 组通信
  - 组成员关系服务——视图传递
  - 视图同步的组通信
- 容错服务
  - 线性化能力和顺序一致性
  - 主备份复制
  - 主动复制



小结

- 高可用服务
  - gossip
  - coda
- 复制数据上的事务
  - 读一个/写所有
  - 可用拷贝复制
  - 法定数共识方法
  - 虚拟分区



思考题

- P484
- 18.1 18.2 18.9
- 18.14