分布式系统

Distributed Systems

陈鹏飞 吴维刚

{chchenpf7, wuweig}@mail.sysu.edu.cn

主页：http://cse.sysu.edu.cn/



第七讲 — 一致性和复制

引言

CAP原则又称CAP定理，指的是在一个分布式系统中，Consistency（一致性）、Availability（可用性）、Partition tolerance（分区容错性），三者不可得兼

大纲

1. 背景知识
2. 以数据为中心的一致性模型
3. 以客户为中心的一致性模型
4. 复制管理
5. 一致性协议



1 背景知识

进行复制的原因

性能　　可用性

- **主要问题**

为了保证复制的一致性，我们通常需要确保所有冲突的操作无论在任何地方都要按照相同的顺序执行。

- **操作冲突：**

读写冲突（Read-write conflict），读操作和写操作并发执行；

写写冲突（Write-write conflict），两个并发的写操作；

- **问题**　　最终一致性

当发生冲突操作时，保证全局一致性是非常困难的，代价很高，一个折中的解决方案是：减弱一致性需求，避免全局一致性；



2

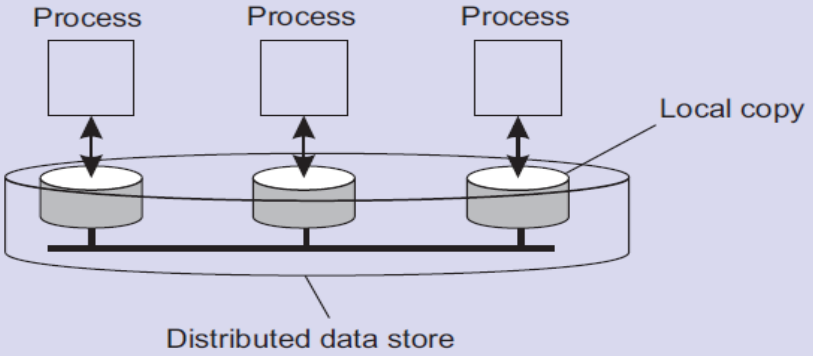
以数据为中心的一致性模型

以数据为中心的一致性模型

➢ 一致性模型

- 实质上是进程和数据存储之间的一个约定。即如果进程统一遵守某些规则，那么数据存储将正常运行。数据存储精确规约了当出现并发操作时读写操作的返回结果。

➢ 关键点



连续一致性

- **也可以定义为“一致性程度”**
  - 数值偏差：不同副本之间的数值可能不同；
  - 新旧偏差：不同副本之间的“新旧”不同；
  - 顺序偏差：不同副本更新操作的**顺序和数量**可能不同（即不同更新方法）；
- **Conit**
  - 一致性单元=> 一致性可度量的单元；

Conit

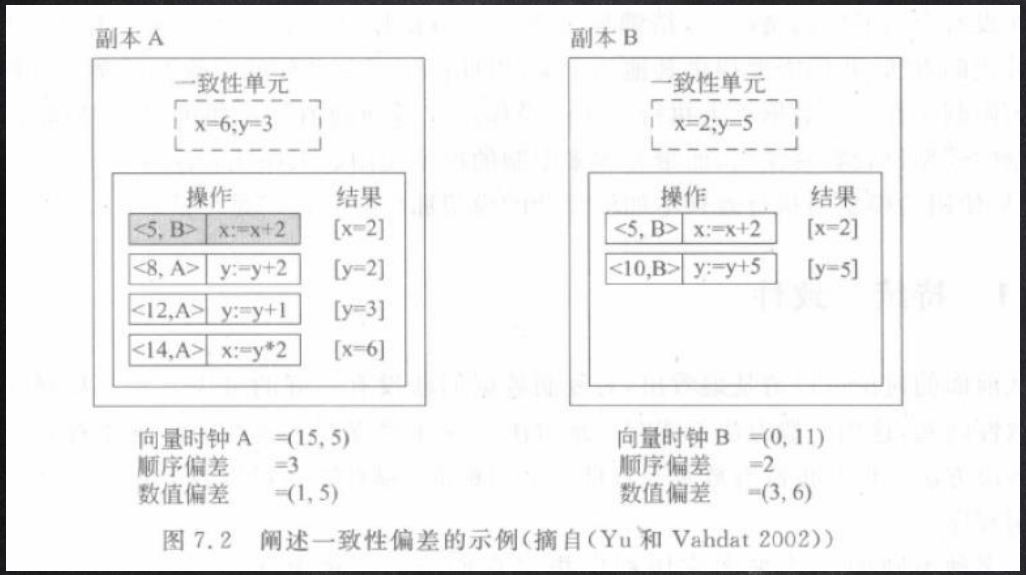


图 7.2 阐述一致性偏差的示例(摘自(Yu 和 Vahdat 2002))

- 该Conit包含 x, y;
- 每一个副本包含一个向量时钟；
- A 接收 B的操作那么A会将该操作持久化（不允许回滚）；

Conit

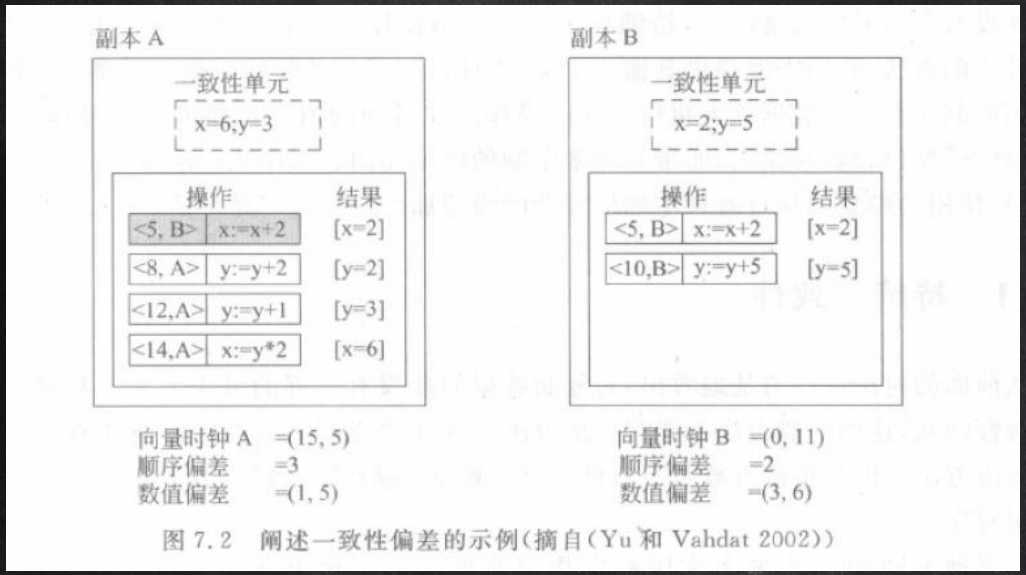


图 7.2 阐述一致性偏差的示例(摘自(Yu 和 Vahdat 2002))

- 数值偏差的构成（未接收到的更新次数，偏差权重）
- 偏差权重= A已提交的值（x,y），与还没有收到来自B的操作产生的结果之间的最大差分；

Conit的粒度

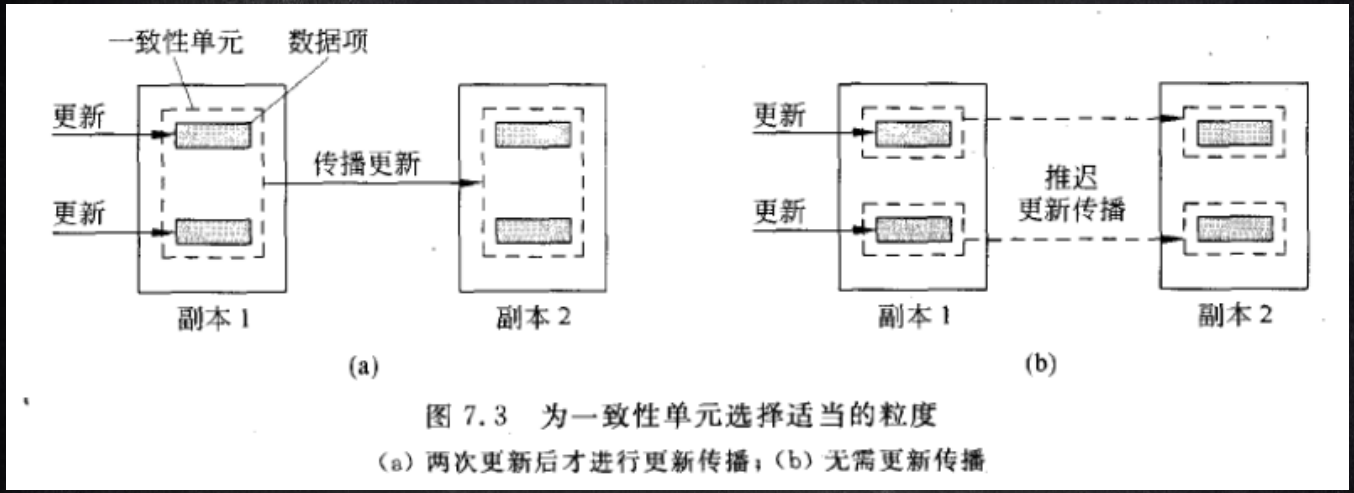


图 7.3 为一致性单元选择适当的粒度

(a) 两次更新后才进行更新传播；(b) 无需更新传播

需要在“粗粒度”和“细粒度”之间权衡

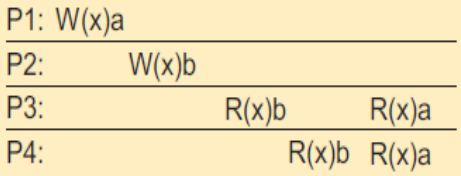
顺序一致性

- 定义

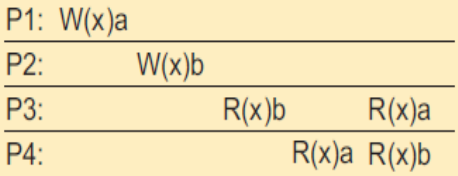
重要的以数据为中心的一致性模型。 任何执行结果都是相同的，就好像所有进程对数据存储的读写操作是按照某种序列顺序执行的，并且每个进程的操作按照程序所定制的顺序出现在这个序列中。

进程可以看到所有进程的写操作，但是只能看到自己的读操作

- 顺序一致的数据存储（a）和非顺序一致的数据存储（b）



(a)



(b)

因果一致性（Causal Consistency）

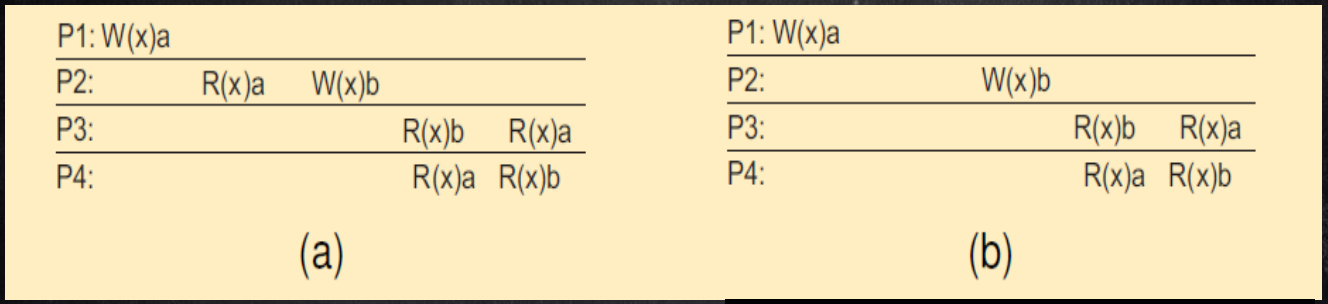
➢ 定义

一种弱化的顺序一致性模型。

所有的进程必须以相同的顺序看到具有潜在因果关系的写操作。

不同机器上可以以不同的顺序看到并发写操作。

➢ 因果一致的数据存储（a）和非因果一致的数据存储（b）



顺序一致性却不满足？

分组操作

- 定义

- 在一个进程对被保护的共享数据的所有更新操作执行完之前，不允许另一个进程执行对同步化变量的获取访问；
- 如果一个进程对某个同步化变量正进行互斥模式访问，那么其他进程就不能拥有该同步化变量，即使是非互斥模式也不行；
- 某个进程对某个同步化变量的互斥模式访问完成后，除非该变量的拥有者执行完操作，否则，任何其他进程对该变量的下一个非互斥模式访问也是不允许的；

分组操作

➢ 解释

- **条件1：**在一个进程对被保护的共享数据的所有更新操作执行完之前，不允许另一个进程执行对同步化变量的获取访问；

- **解释：**当一个进程获得拥有权后，这种拥有权直到所有被保护的数据都已更新为止、换句话说：对被保护数据的所有远程修改都是可见的。

分组操作

➢ 解释

- **条件2：**如果一个进程对某个同步化变量正进行互斥模式访问，那么其他进程就不能拥有该同步化变量，即使是非互斥模式也不行；

- **解释：**在更新一个共享数据项之前，进程必须以互斥模式进入临界区，以确保不会有其他进程试图同时更新该共享数据。

分组操作

➢ 解释

- **条件3**：某个进程对某个同步化变量的互斥模式访问完成后，除非该变量的拥有者执行完操作，否则，任何其他进程对该变量的下一个非互斥模式访问也是不允许的；
- **解释**：如果一个进程要以非互斥模式进入临界区，必须首先与该同步化变量的拥有者进行协商，确保临界区获得了被保护共享数据的最新副本。

分组操作

- 一个入口一致性的实例

P1:	L(x)	W(x)a	L(y)	W(y)b	U(x)	U(y)						
P2:							L(x)	R(x)a			R(y)	NIL
P3:									L(y)	R(y)b		

- 观察

入口一致性需要（显式或者隐式）加锁或者解锁数据；

- 问题

如何让这种一致性对于开发人员更加透明？

中间件



3 以客户为中心的一致性模型

背景

- 以数据为中心的一致性模型

  读写并重的数据存储；

- 以客户为中心的一致性模型

  读多写少，提供弱一致性即最终一致性；（举例）

- 最终一致性

  如果在很长一段时间内没有更新操作，那么所有的副本将逐渐地成为一致的。

移动用户的一致性

➢ 样例

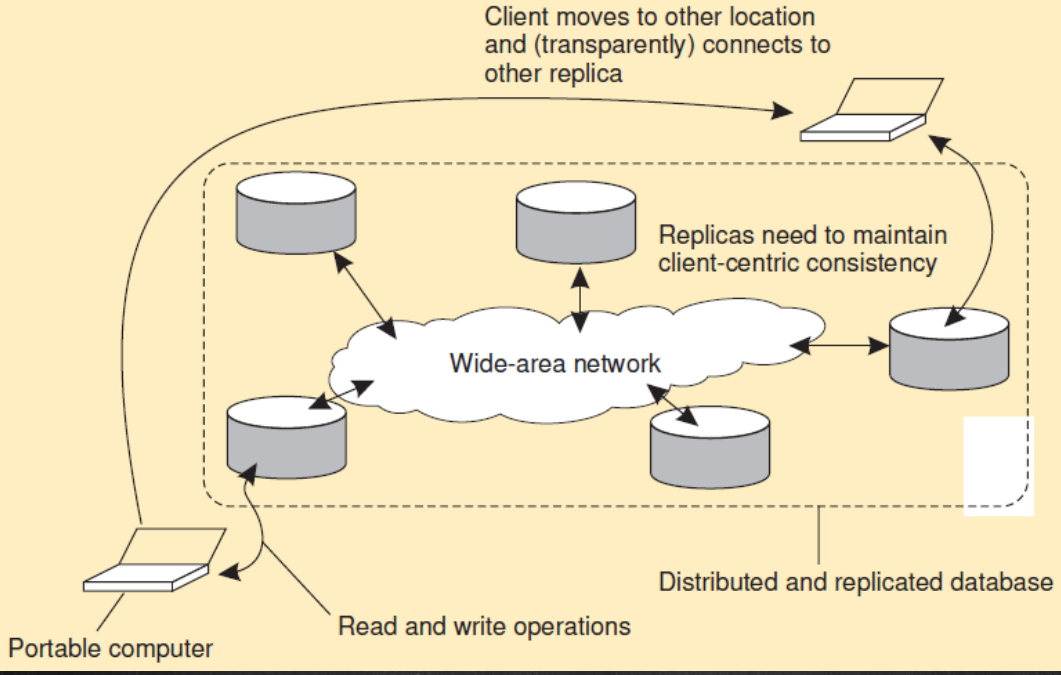
考虑一个可以通过笔记本访问的分布式数据库，假定你的笔记本可以作为前端访问数据库；

- 在地点 A，你访问数据库并进行读写操作；
- 在地点 B 你继续原来的工作，但是除非你访问位于A上的服务器，否则会检测到不一致：
  - 你在A处的更新还没有传播到B；
  - 你可能访问到比A处更新的数据；
  - 你在B处的更新可能最终于A处的更新发生冲突；

基本架构

以客户为中心的一致性

The principle of a mobile user accessing different replicas of a distributed database



单调读

➢ 定义

如果一个进程读取数据项 x 的值，那么该进程对 x 执行的任何后续操作将总是得到第一次读取的那个值或更新的值；

➢ 单调一致性

单调一致性的数据存储（a）和不提供单调一致性的数据存储(b)

L1:	$W_1(x_1)$	$R_1(x_1)$
L2:	$W_2(x_1;x_2)$	$R_1(x_2)$

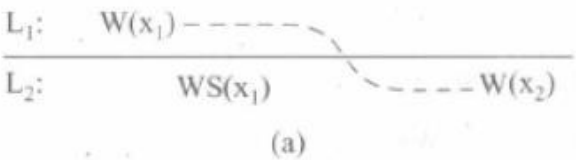
L1:	$W_1(x_1)$	$R_1(x_1)$
L2:	$W_2(x_1\|x_2)$	$R_1(x_2)$

- $W_1(x_2)$ is the write operation by process P_1 that leads to version x_2 of x
- $W_1(x_i;x_j)$ indicates P_1 produces version x_j based on a previous version x_i.
- $W_1(x_i|x_j)$ indicates P_1 produces version x_j concurrently to version x_i.

单调写

- 定义

一个进程对数据项 x 执行的写操作必须在该进程对 x 执行任何后续写操作之前完成；



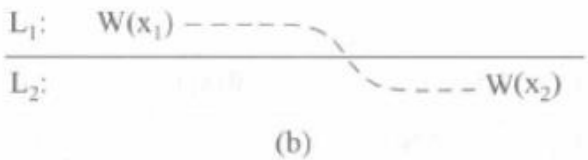


图 7.13 单一进程 P 对同一数据存储的两个不同本地副本所执行的写操作

(a) 提供单调写一致性的数据存储；(b) 不提供单调写一致性的数据存储

单调写

- **举例-1**

  - 在服务器 S2上更新程序，并且保证程序编译、链接所依赖的组件已经在S2 存在；

- **举例-2**

  - 代码版本管理程序， 在任何地方按照正确的顺序维护副本文件的版本；

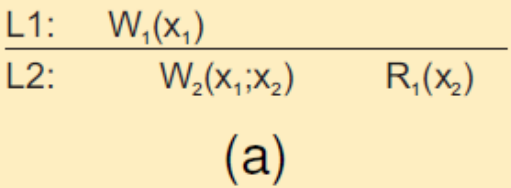
读写一致性（Read your writes）

- 定义

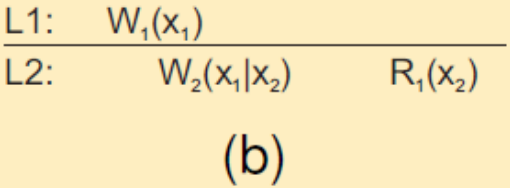
一个进程对数据项 **x** 执行一次写操作的结果总是会被该进程对 **x** 执行的后续读操作看见。

也就是说，一个写操作总是在同一进程执行的后续读操作之前完成，而不管这个后续读操作发生在什么位置

- 保证读写一致性的数据存储（a）和非读写一致性的数据存储



(a)



(b)

读写一致性例子

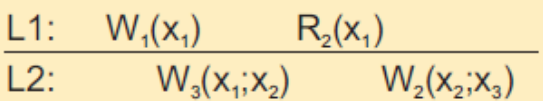
- 更新Web页面，并且保证Web浏览器能够展示最新的版本的数据，而不是缓存的内容；

写读一致性（Writes follow reads）

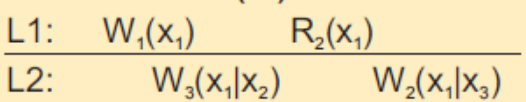
- 定义

同一个进程对数据项 **x** 执行的读操作之后的写操作，保证发生在与 **x** 读取值相同或比之更新的值上。即更新是作为前一个读操作的结果传播的。 进程对数据项 **x** 所执行的任何后续的写操作都会在 **x** 的副本上执行，而该副本是用该进程最近读取的值更新的。

- 写读一致性的数据存储（**a**）和非写读一致性的存储（**b**）



(a)



(b)



4 复制管理

副本管理

- **背景**

对于任何支持副本的分布式系统来说，关键的问题是决定何处、何时、由谁来负责副本以及以何种机制来保持副本的一致性。 副本放置问题：副本服务器的放置问题和内容放置问题。

副本服务器放置

➢ 本质

从 N 个可能的位置中找出 K 个最佳的位置；

- 假定已放置了 k 个服务器，需要从 N-k个服务器中选择一个最佳的服务器，与所有的客户端之间的距离最小。 计算复杂度过高；
- 选择第 K 大的自治系统 ，然后在含有最大数量的连接的路由器上放置一台服务器，一次类推。计算复杂度高；
- 假定在一个 d 维的几何空间中放置服务器， 节点之间的距离反映了延迟。 把整个空间划分为多个单元，选择 K 个密度最大的单元放置副本服务器。 计算复杂度比较低；

内容放置

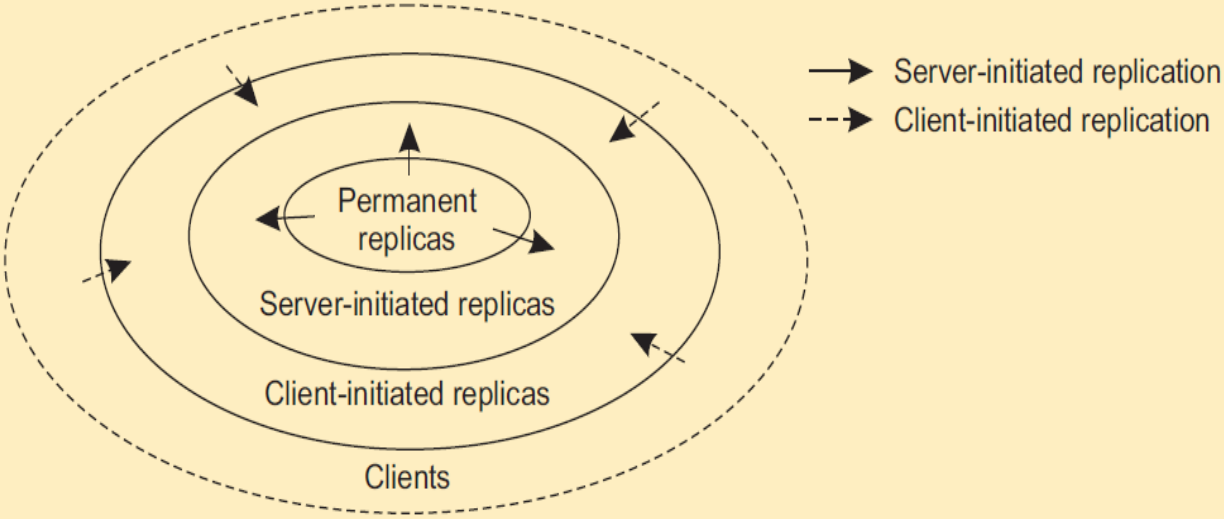
- **区分三种不同的进程**

一个进程能够维护对象或者数据的副本：

- 永久副本：进程/机器持久存储副本数据；
- 服务器启动的副本：进程可以动态的持有副本数据，该副本是在初始化数据存储的所有者时创建的；
- 客户端启动的副本：当客户端初始化时创建的副本；

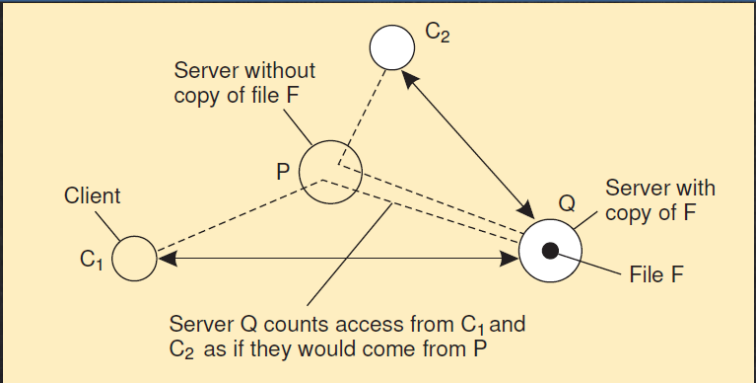
内容放置

- 不同类型的副本逻辑地组织成三个同心环



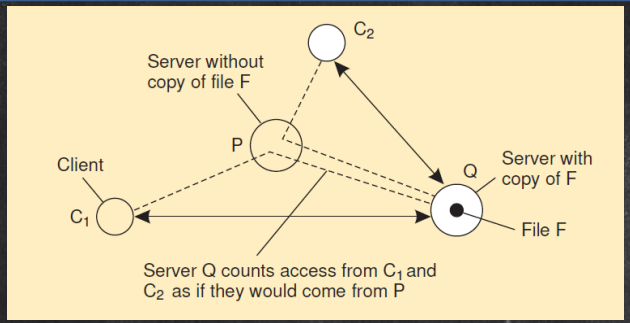
服务器初始化的副本

- **动态复制主要考虑的问题**
  - 复制可能是为了减轻一台服务器的负载而进行的；
  - 一台服务器上制定的文件可能被转移或复制到对这些文件提出很多请求的客户附近的服务器。
- **对来自不同客户端的请求计数**



服务器初始化的副本

- 对来自不同客户端的请求计数



一个进程能够维护对象或者数据的副本：

- 记录每个文件的访问次数，并且将其合并为来自最靠近客户端服务器的请求；
- 如果请求数量低于阈值D => 删除文件；
- 如果请求数量超过阈值R=>复制文件；
- 如果请求数量在 D 和 R之间 => 移动文件；

内容分发

➢ 状态与操作

一个重要的设计问题是要实际传播哪些信息：

- 只传播更新的通知（常用于缓存）；
- 将数据从一个副本传送到另一个副本（被动复制）；
- 把更新操作传播到其他副本（主动复制）；

➢ 注意

没有哪一个方法是最佳的选择，高度依赖于可用的网络带宽和副本上的读写比率；

内容分发：客户/服务器系统

➢ **在多客户端、单服务器系统中，基于Push和基于Pull的协议的比较**

- 基于Push的更新：服务器初始化的方法，不需要其他副本请求更新，这些更新就被传播到那些副本那里；
- 基于Pull的更新：客户端初始化的方法，客户端请求的更新；

Issue	Push-based	Pull-based
1:	List of client caches	None
2:	Update (and possibly fetch update)	Poll and update
3:	Immediate (or fetch-update time)	Fetch-update time

1: State at server
2: Messages to be exchanged
3: Response time at the client

内容分发

➢ 观察

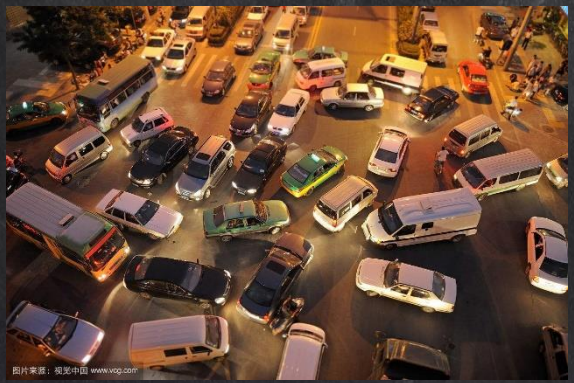
利用租用（lease）的方式在Pull和Push之间动态切换。租用是服务器所作的承诺，在指定的时间内把更新推给客户。租用到期改为pull方式。

➢ 租用失效的时间依赖于系统的行为（自适应租用）

- Age-based leases: An object that hasn't changed for a long time, will not change in the near future, so provide a long-lasting lease
- Renewal-frequency based leases: The more often a client requests a specific object, the longer the expiration time for that client (for that object) will be
- State-based leases: The more loaded a server is, the shorter the expiration times become

5

一致性协议



一致性协议

- 定义

  一致性协议描述了特定的一致性模型的实现。

- 持续一致性： 限定数值偏差

- Every server S_i has a log, denoted as L_i.
- Consider a data item x and let $val(W)$ denote the numerical change in its value after a write operation W. Assume that

$$\forall W : val(W) > 0$$

- W is initially forwarded to one of the N replicas, denoted as $origin(W)$. $TW[i,j]$ are the writes executed by server S_i that originated from S_j:

$$TW[i,j] = \sum\{val(W) | origin(W) = S_j \ \& \ W \in L_i\}$$

持续一致性： 限定数值偏差

Note

Actual value $v(t)$ of x:

$$v(t) = v_{init} + \sum_{k=1}^{N} TW[k,k]$$

value v_i of x at server S_i:

$$v_i = v_{init} + \sum_{k=1}^{N} TW[i,k]$$

持续一致性： 限定数值偏差

Problem

We need to ensure that $v(t) - v_i < \delta_i$ for every server S_i.

Approach

Let every server S_k maintain a view $TW_k[i,j]$ of what it believes is the value of $TW[i,j]$. This information can be gossiped when an update is propagated.

Note

$$0 \leq TW_k[i,j] \leq TW[i,j] \leq TW[j,j]$$

持续一致性： 限定数值偏差

➢ 核心思想

当服务器Sk知道Si与提交给Sk的更新操作步调不一致时，
它把写操作从日志中转发给Si。
转发操作可以把Sk的试图TWk[i，k]往TW[i，k]靠近。
当应用程序提交一个新的写操作时，Sk会把其视图往TW[i，k]推，从而：
TW[i，k] - TWk[i，k] > δi/(N-1)
但是这样的方法能够确保：TW[i，k] - TWk[i，k] <= δi

持续一致性：限定复制的新旧程度偏差

➢ 核心思想

- 让服务器 S_k 保持实时向量时钟 RVC_k，其中RVC_k[i] = T(i) 为到时间 T(i)时， S_k看到了已提交给S_i的所有写操作；

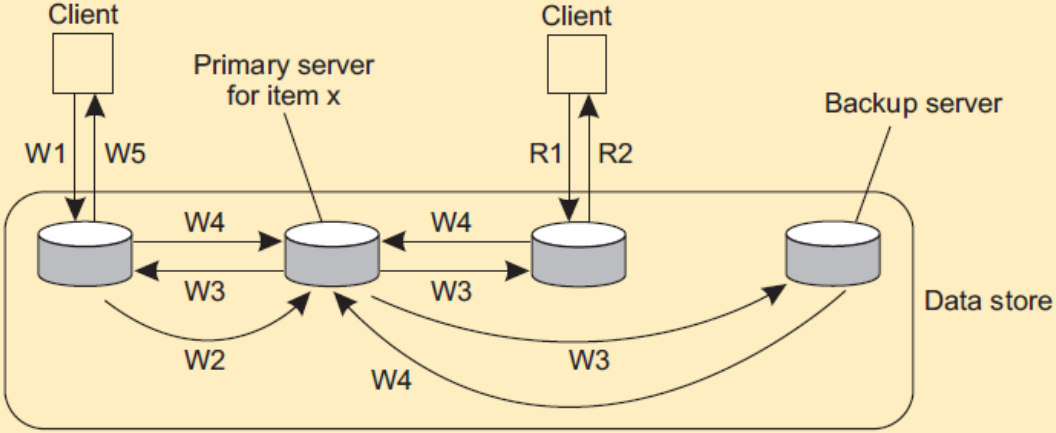
- 只要服务器 S_k 通知 T[k]-RVC_k[i] 将超出指定的界限，那么它就开始拉入来自S_i的时间戳晚于RVC_k[i]的写操作；

持续一致性：限定顺序偏差

- 暂时写操作的本地队列；
- 当本地队列的长度超过制定的最大长度时，服务器不再接受任何新提交的写操作，而是按照相应的顺序提交写操作；

基于主备份的协议

- 远程写协议



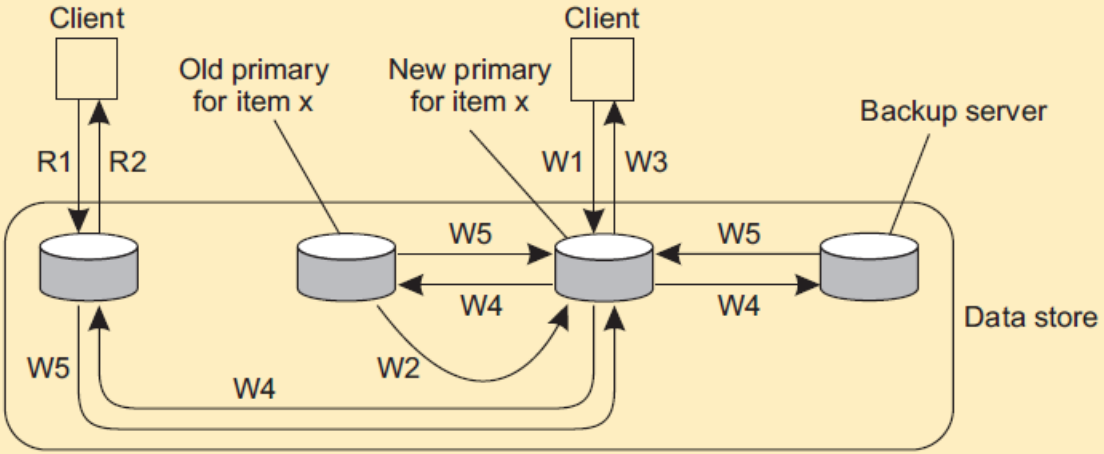
W1. Write request
W2. Forward request to primary
W3. Tell backups to update
W4. Acknowledge update
W5. Acknowledge write completed

R1. Read request
R2. Response to read

传统的分布式数据库和文件系统需要较高的容错性，副本位于LAN

基于主备份的协议

- 本地写协议



W1. Write request
W2. Move item x to new primary
W3. Acknowledge write completed
W4. Tell backups to update
W5. Acknowledge update

R1. Read request
R2. Response to read

主要应用于离线模式下的移动计算机，在断线之前传递相关文件；

复制的写协议

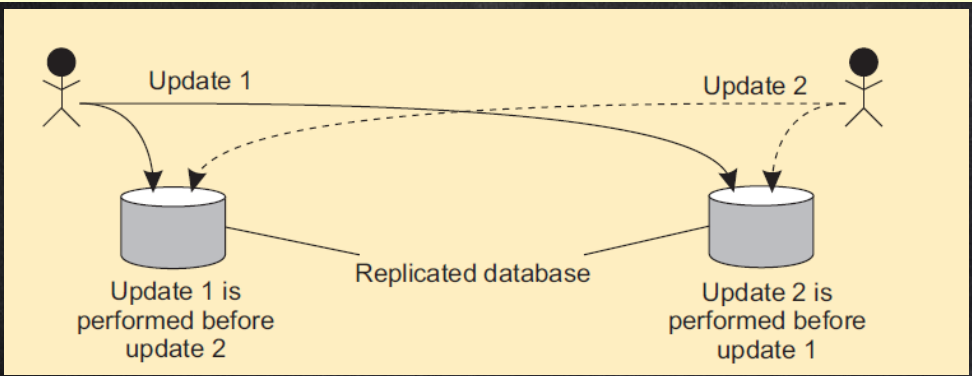
基于全副本主动复制的协议

更新操作发给每一个副本

每个副本各自进行更新

如何保证顺序一致？

需要专门的控制协议，如全序多播



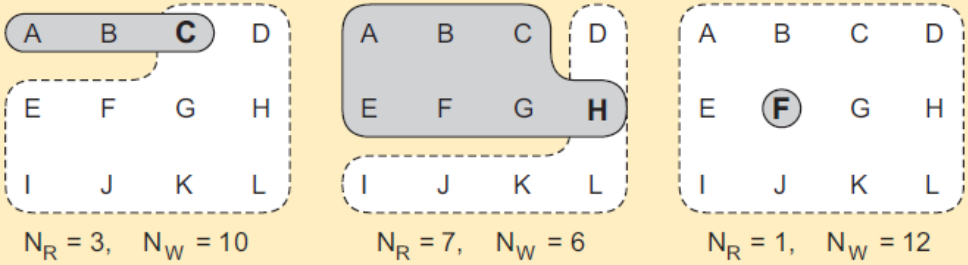
复制的写协议

(1) $N_R + N_W > N$;
(2) $N_w > N/2$。

Quorum-based protocols 基于团体的协议

Ensure that each operation is carried out in such a way that a majority vote is established: distinguish read quorum and write quorum

Three examples of the voting algorithm. (a) A correct choice of read and write set. (b) A choice that may lead to write-write conflicts. (c) A correct choice, known as ROWA (read one, write all)



客户为中心的一致性协议

客户检查自己的更新是否完成

每个写操作被唯一标识；
每个客户维护两个集合：
读操作集{客户的读操作相关的写操作}；
写操作集{客户的写操作}

单调读/写：
相关的读/写操作集与读/写请求一起发送服务器；
执行读操作前检查是否所有写已经在本地执行；
没有的话：联系其他服务器进行更新，或转发读请求出去。

读写一致（写后读）：读之前检查写操作集。
写读一致（读后写）：写之前检查读操作集，并将读操作集加入写操作集。

小结

- 一致性是分布式系统，特别是数据复制的关键问题
- 复制管理：副本、缓存；放置、内容
- 一致性模型：对一致性的定义和约定
  - 数据为中心的一致性（读写并重系统）
    - 连续（持续）一致性：数值偏差、陈旧偏差、更新顺序偏差
    - 操作一致性：顺序一致性、因果一致性、入口一致性（分组）
  - 用户为中心的一致性（读为主）
    - 单调读、单调写、读写一致（写后读）、写读一致（读后写）
- 一致性协议：一致性模型的实现机制、方法
  - 数据为中心：
    - 持续一致性协议：更新扩散过程限制偏差
    - 操作一致性协议：主备份写、全复制写、多数（Quorum）写
  - 用户为中心：
    - 读写集检查

谢谢！