7.分布式一致性与复制管理

1.一致性与复制

复制的理由：

- 提高可靠性：防止单点失败，数据校验
- 提高性能：并行性，可伸缩性

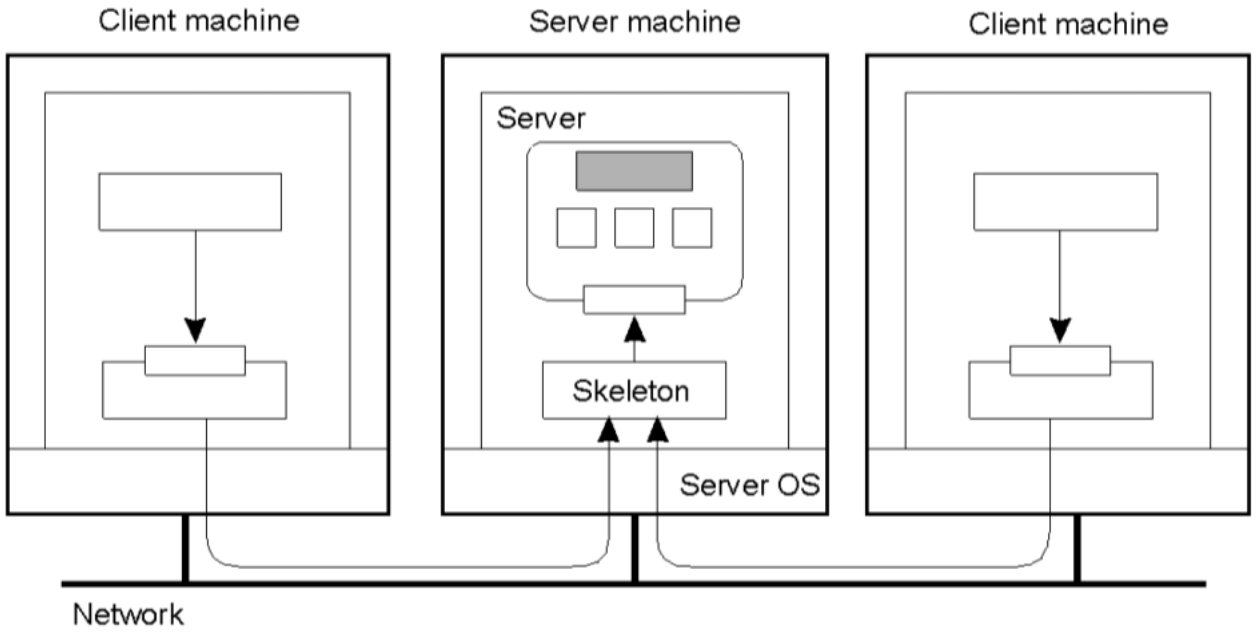
复制的代价

- 一致性维护：更新问题
- 例 1：Web 页的 Cache、镜像网站

1.1 对象复制问题

（1）单副本对象的同步控制

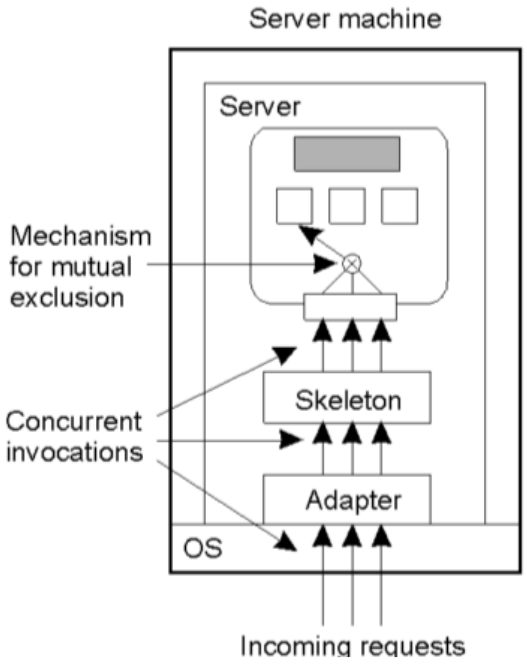
例：两个客户并发访问一个分布式远程控制对象

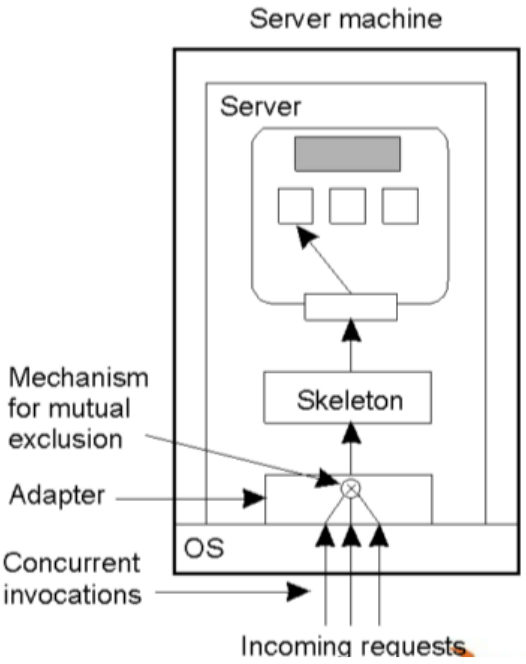


（2）单副本同步控制方法

由远程对象自己处理对它的并发调用。 如 Java，同时只允许执行一个方法线程

由对象适配器处理并发调用。如一个对象一个线程

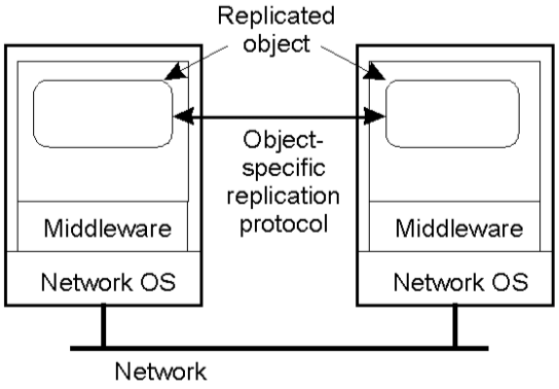


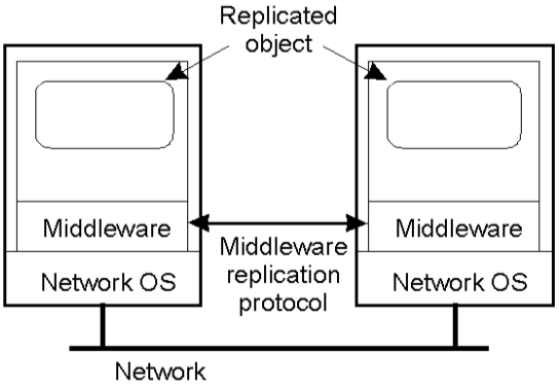


（3）多副本对象的同步控制方法

构造感知复制对象，由对象自己保证一致性。

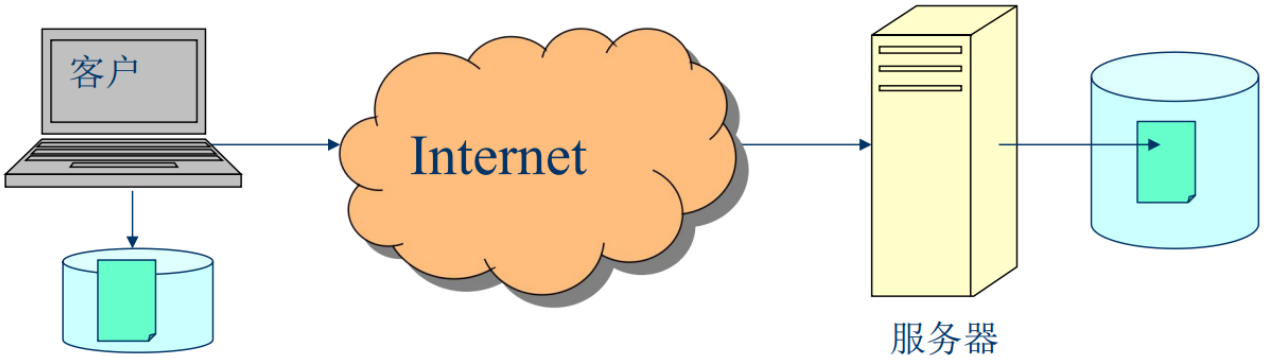
由分布式系统负责复制管理，由系统保证对副本访问的正确次序。例，CORBA 中，支持全序的、因果一致性的对象调用





1.2 支持伸缩性的复制技术

将数据的副本放置在处理它们的进程附近以减少访问时间，解决可伸缩性问题



复制策略

- 更新操作和访问操作的 trade-off
- 设进程 P 对数据 d 的访问 N 次/秒，d 的更新 M 次/秒
- 当 N<<M 时，访问/更新比非常低，由于一致性维护带来更大代价，因此，不应复制

一致性维护与可伸缩性问题

- 保证所有的副本都是相同的，→ 紧密一致性
- 当某个副本上执行更新操作时，需对所有副本进行全局同步 ，在大型系统上很难实施 → 可伸缩性问题

解决策略

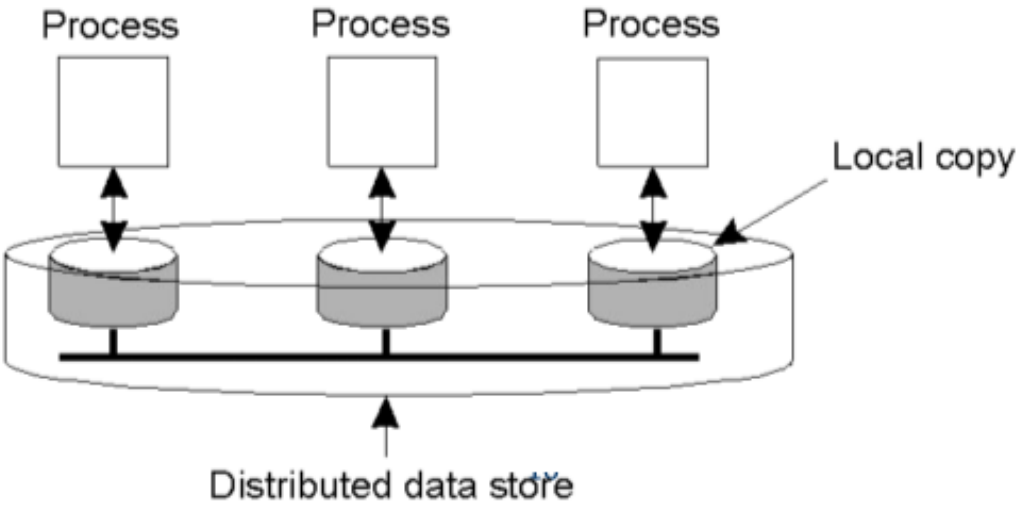
- 松驰一致性，所有副本不一定保持完全相同，尽量减少立即的全局同步

2.以数据为中心的一致性模型

2.1 分布式数据仓（data store）模型

物理上，分布的和复制的。例，分布式共享内存、数据库、文件

操作：每个进程可执行读操作，写操作。写操作在本地副本上进行，再传播给其他副本



2.2 一致性模型

数据相干性（coherency）

- 同一个数据在各个数据仓中的值保持一致
- 从单个数据的视角

一致性模型

- 多个进程与多个数据之间的操作，保持一致性
- 进程与数据仓之间的契约(contract)
- 如果进程遵守约定的规则，数据仓就能工作正常。
- 如果进程违反了这些规则，数据仓就不再保证操作的正确性

2.3 连续一致性

连续一致性（continuous consistency）

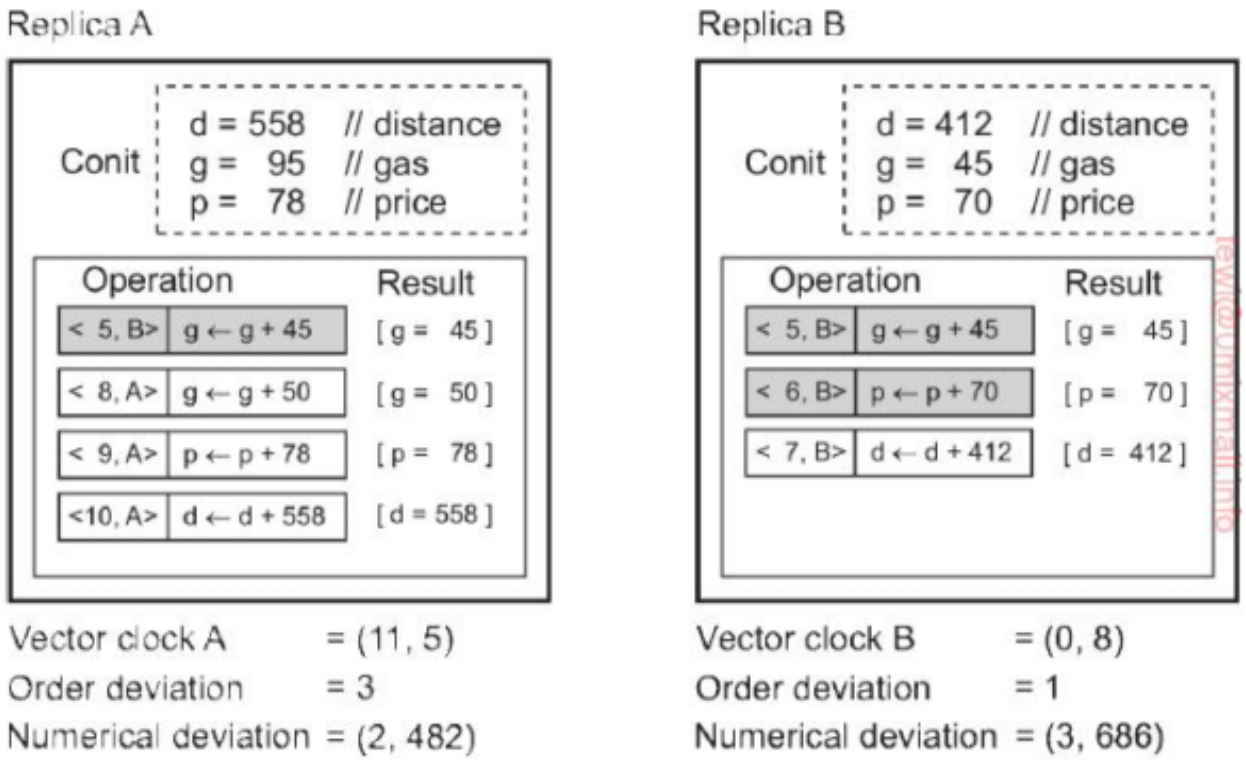
- （数值误差，次序误差，陈旧度）
- 数值误差：未传播的写操作的权重值
- 次序误差：临时写操作的个数
- 陈旧度：写操作传播的延迟(只要一个副本不太旧，就可以容忍它提供旧的数据。例如，天气报告）

一致性单元（conit）：受控的数据集

顺序偏差即在本地还没有提交的更新操作的个数

数值偏差（x,y）： x 表示其他副本已经做了， 但是本地副本还没看见的操作数量；Y 表示这些操作带来的数值变化

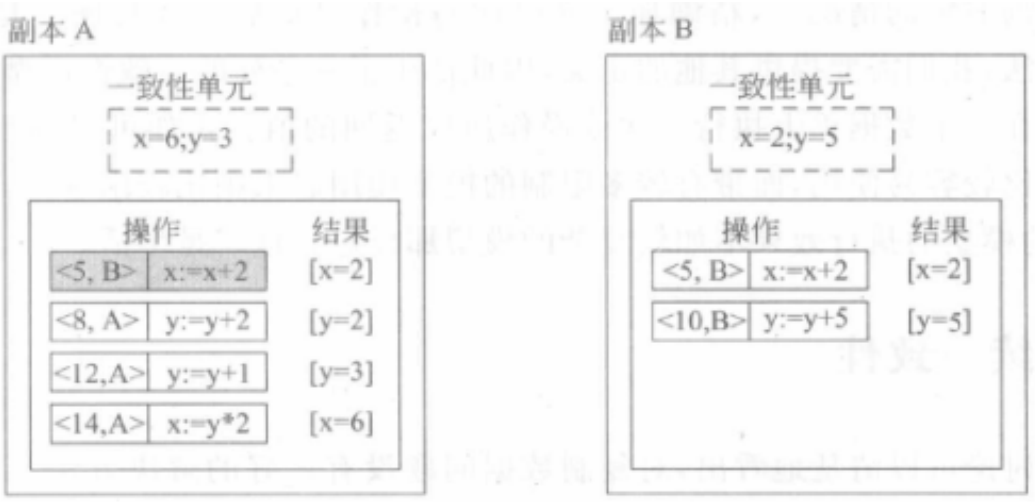
举例：conit(x, y)



<5, B>：5 为时间戳，B 为从 B 传过来

（1）示例

示例 1：如下图所示，为持续一致性的示意图。在副本 A 和副本 B 中有含数据项 x 和 y，这两个变量都假设初始化为 0。操作表格中阴影部分表示为持久化（永久性）的操作，不能回滚；其余操作表示暂时的更新操作。请采用向量时钟、顺序误差和数值误差三个指标描述两个副本之间的差异。（其中数值误差用各数据项的差分和表示）



(1) 向量时钟：

- 对于副本 A 而言，由于副本 A 和副本 B 中有含数据项 x 和 y，这两个变量都假设初始化为 0，副本 A 最后的操作是<14, A>，所以 A 的向量时钟：A = (15, 5)
- 对于副本 B 而言，由于副本 A 和副本 B 中有含数据项 x 和 y，这两个变量都假设初始化为 0，副本 A 最后的操作是<10, B>，所以 B 的向量时钟：B = (0, 11)

(2) 顺序偏差

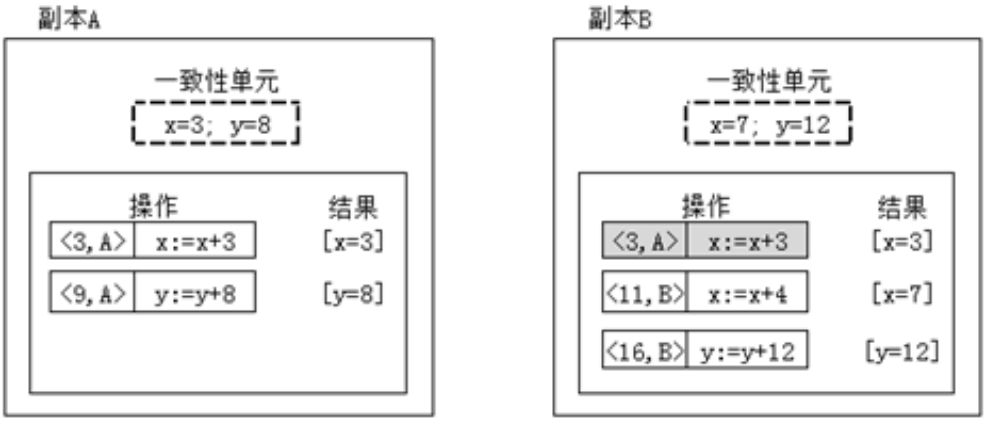
- 对于副本 A 而言，阴影部分表示为持久化（永久性）的操作，不能回滚，而 A 中不是阴影部分的有三个，即<8, A>, <12, A>, <14, A>，所以 A 中有三个暂存的操作，即顺序偏差为 3
- 对于副本 B 而言，阴影部分表示为持久化（永久性）的操作，不能回滚，而 B 中不是阴影部分的有二个，即<5, B>, <10, B>，所以 B 中有二个暂存的操作，即顺序偏差为 2

(3) 数值偏差

- 对于副本 A 而言，A 已经收到了 B 的<5, B>，但是还未看到<10, B>，所以数值偏差=1（即还有一个没有收到）；而此时 A 中已提交的值是（x, y）=(2, 0)，假设收到 B 的<10，B>之后，y=5；所以副本 A 的数值偏差=（1，5）
- 对于副本 B 而言，B 还未看到 A 中的<8, A>，<12, A>, <14, A>，所以数值偏差=3（即还有三个没有收到）；而此时 B 中已提交的值为（x，y）=（0，0），假设收到 A 的<8, A>，<12, A>, <14, A>之后，x=(1+2)*2，所以偏差的权重= 6；所以副本 B 的数值偏差=（3，6）

示例 2：如下图所示，为持续一致性的示意图。在副本 A 和副本 B 中有含数据项 x 和 y，这两个变量都假设初始化为 0。操作表格中阴影部分表示为持久化（永久性）

的操作，不能回滚；其余操作表示暂时的更新操作。请采用向量时钟、顺序误差和数值误差三个指标描述两个副本之间的差异。（其中数值误差用各数据项的差分和表示）



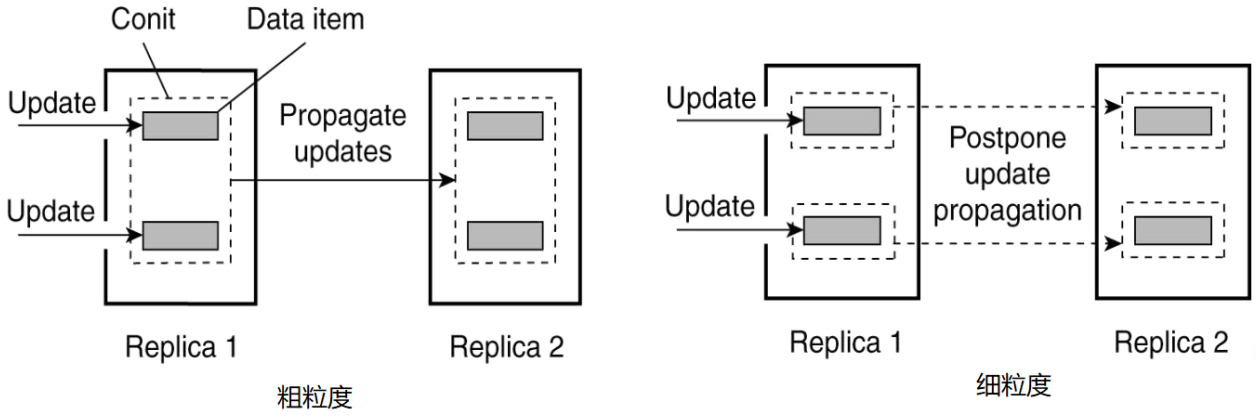
A：向量时钟：(10, 0)；顺序偏差：2 ；数值偏差：(2, 16)

B：向量时钟：(3, 17)；顺序偏差：2 ；数值偏差：(1, 8)

（2）一致性单元的粒度选择

粗粒度：任意一个更新操作都导致更新传播；

细粒度：当一个数据更新时，另一个数据无需更新



2.4 严格一致性

规则：对数据项 x 的读操作返回的值为最近写入 x 的值

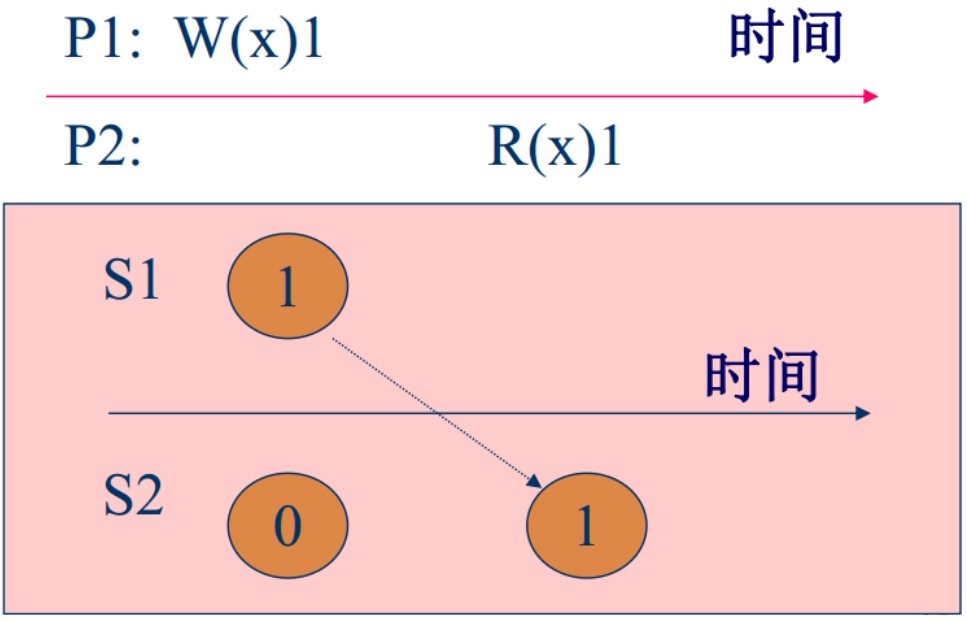
特点：绝对全局时间次序

不可实现性

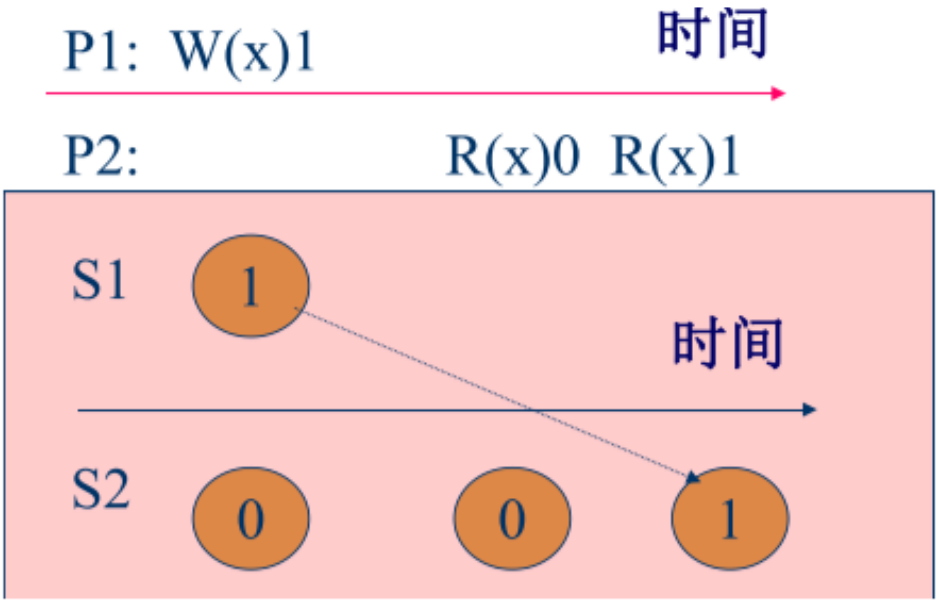
- 没有全局时钟

- 光速限制： $t1: W1(x) \rightarrow S2,\ t2: R2(x)$; 如 $t2 - t1 = 10^{-9}$ 秒， S1 和 S2 的距离为 3 米， 则需要 10 倍的光速进行传输

例：严格一致性：



例：非严格一致性：



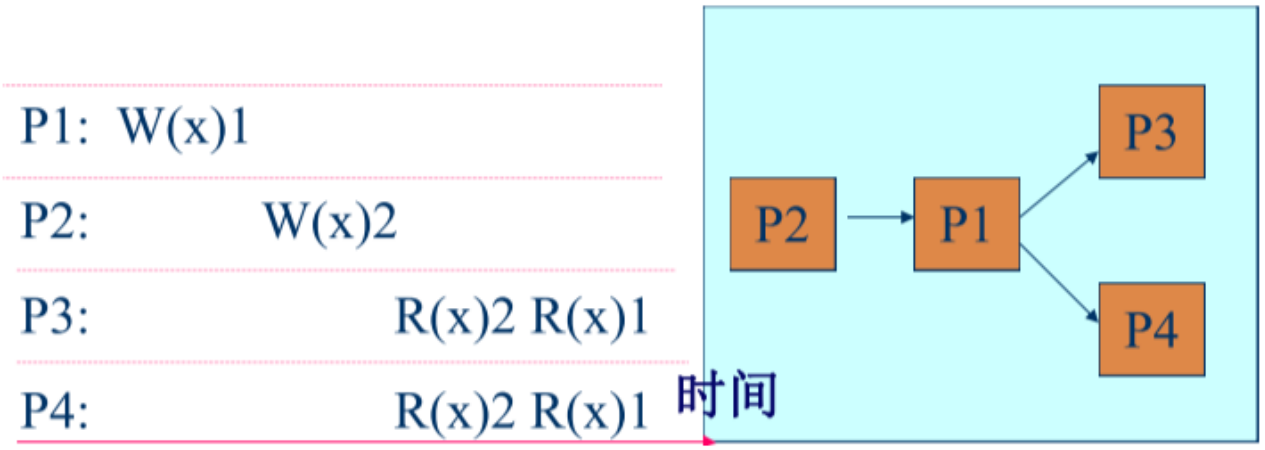
2.5 顺序一致性

规则：所有进程执行的结果，等同于它们的操作按某种顺序在数据仓上执行的结果。每个进程的操作都按照程序规定的顺序。

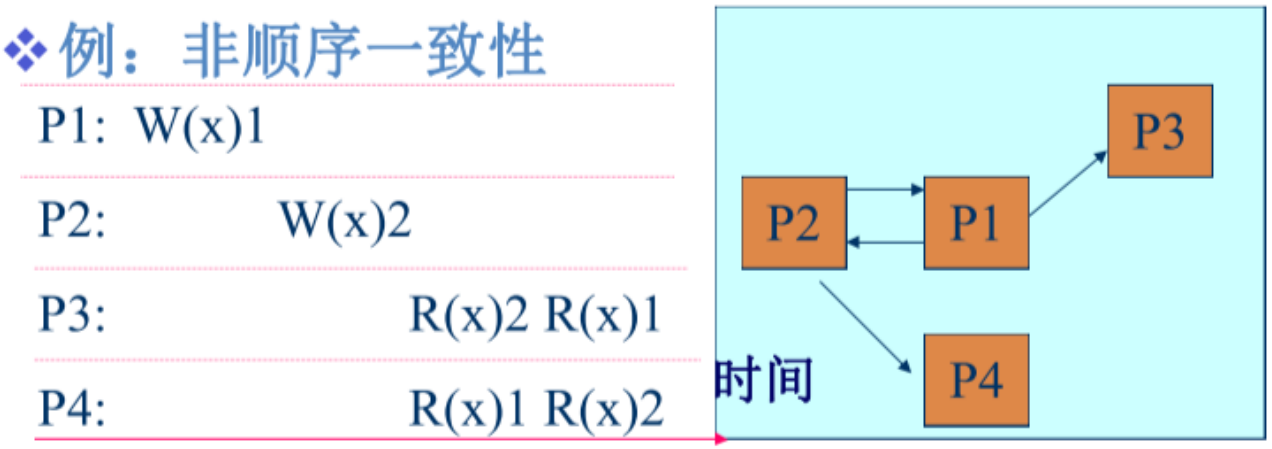
所有进程看到相同的内存访问操作次序，等价于数据库的可串行化（serializability）。

当每个数据项都保持顺序一致性时，他们的组合不一定符合这个要求

顺序一致性模型：



非顺序一致性：



(1) 示例

3 个并行执行的进程，90 种正确的执行顺序

P1	P2	P3
x = 1;	y = 1;	z = 1;
print (y, z);	print (x, z);	print (x, y);

x = 1;
print ((y, z);
y = 1;
print (x, z);
z = 1;
print (x, y);
 	
Prints: 001011
(a)
	x = 1;
	y = 1;
	print (x,z);
	print(y, z);
	z = 1;
	print (x, y);
	
	Prints: 101011
	(b)
		y = 1;
		z = 1;
		print (x, y);
		print (x, z);
		x = 1;
		print (y, z);
		
		Prints: 010111
		(c)
			y = 1;
			x = 1;
			z = 1;
			print (x, z);
			print (y, z);
			print (x, y);
			
			Prints: 111111
			(d)

签名： 把输出结果按照 P1， P2，P3 的顺序排序

（2）执行(Execution)

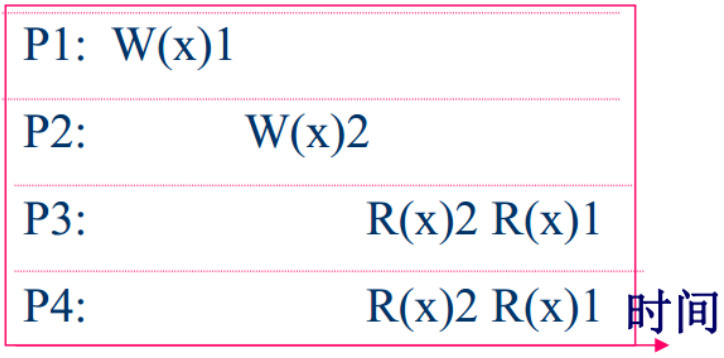
进程 Pi 在数据仓 S 上的读写操作序列， 记为 Ei

$E_1 = W_1(x)1;$

$E_2 = W_2(x)2;$

$E_3 = R_3(x)2, R_3(x)1$

$E_4 = R_4(x)2, R_4(x)1$



（3）历程(History)

合并 E1,E2,..,En 后的序列

就像在一个集中式数据仓上执行

合法历程：保持程序的操作次序，符合数据相干性

$$H= W_2(x)2, R_3(x)2, R_4(x)2, W_1(x)1, R_3(x)1, R_4(x)1$$

非法历程

$$H= W_2(x)2, R_3(x)2, R_4(x)1, R_4(x)2, W_1(x)1, R_3(x)1$$

性能问题：

- 设读操作时间为 r，写操作时间为 w， 包传输时间为 t
- 则 r+w⩾t。 如果减少 r 时间,则必然增加 w 时间

2.6 线性一致性

规则：具有顺序一致性，且如果 $ts_{op1}(x) < ts_{op2}(y)$ ， 则 $OP1(x) \rightarrow OP2(y)$

- 每个操作带有全局时钟戳
- 执行结果是顺序一致性的
- 每个进程的操作遵照时间戳顺序

2.7 因果一致性

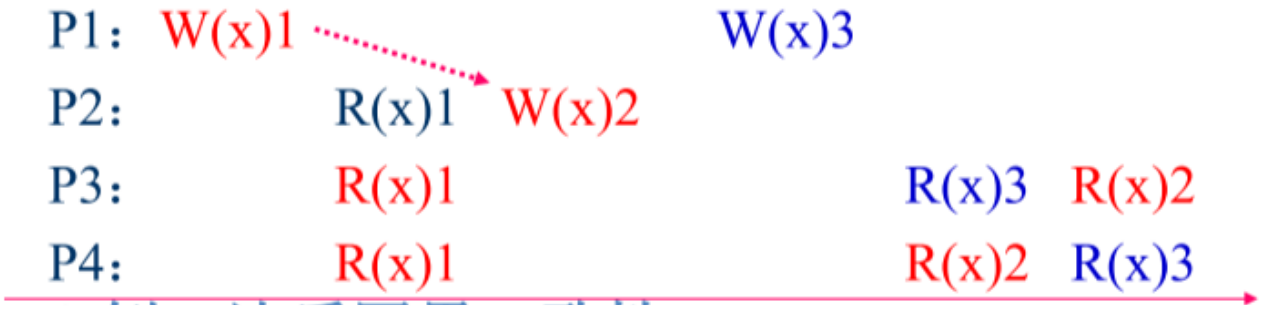
因果关系(Causality)：

- P1 写 x, P2 读 x, 然后写 y，则 W2(y)与 W1(x)具有潜在的因果关系。
- 否则，操作之间的关系为并发(Concurrent)关系
- 例：P1 写 x,P2 写 z，则 W1(x)与 W2(z)不具有潜在因果关系。

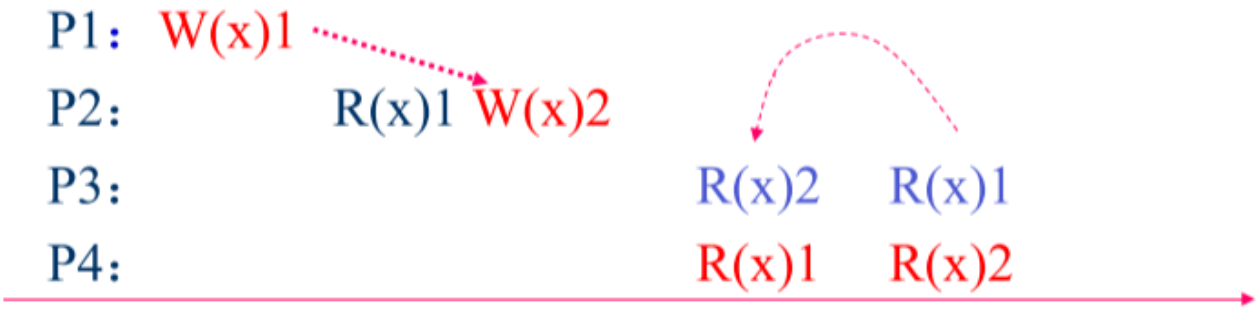
定义：

- 对于具有潜在因果关系的写操作，所有进程看到的执行顺序应相同。
- 并发写操作在不同主机上被看到的顺序可以不同。

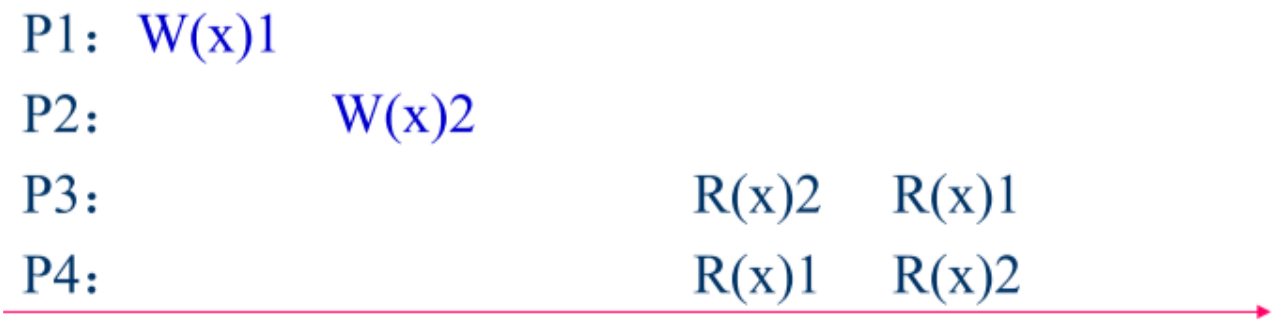
例子：因果一致性



例：违反因果一致性



例：符合因果一致性



实现技术：

- 操作依赖图
- 向量时钟 (vector clock)

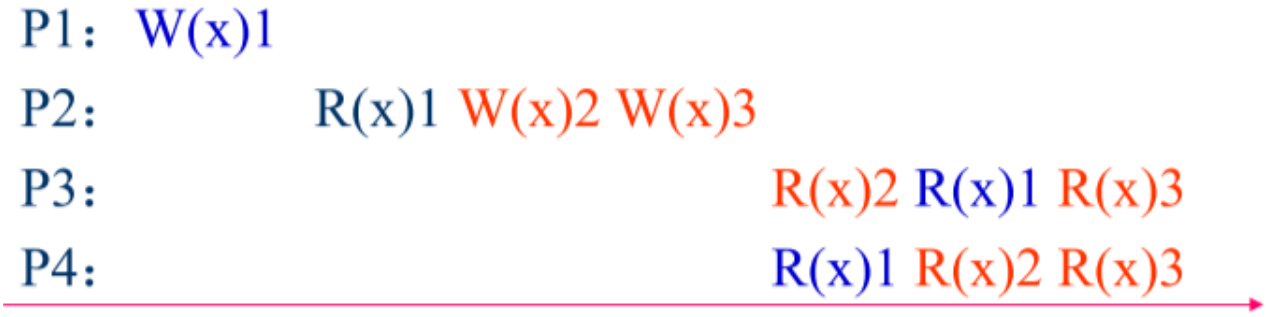
2.8 FIFO 一致性

规则：同一个进程的写操作的执行次序，其它进程看到的都相同。不同进程的写操作的执行次序，不同进程看到的可以是不同的。

也称作内存管道一致性(Pipelined RAM)：分布式共享内存系统

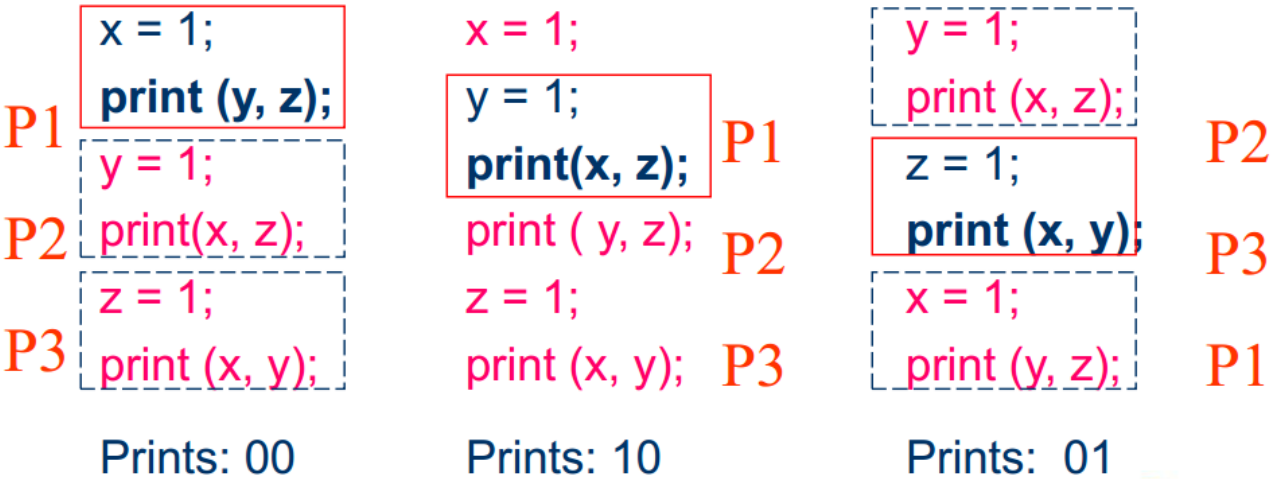
实现技术：写操作标签（进程 ID, 顺序号）

例：符合 FIFO 一致性，但不符合因果一致性



例： 符合 FIFO 一致性， 可输出“001001”，但不符合顺序一致性

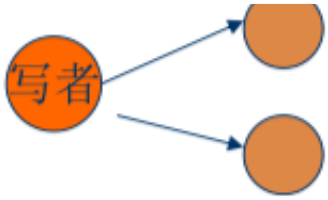
Process P1	Process P2	Process P3
x = 1;	y = 1;	z = 1;
print (y, z);	print (x, z);	print (x, y);



2.9 分组操作

同步分量：与一个数据区相关联

- Synchronize(S)
- 同步所有的数据局部拷贝
- 导出：



- 导入：

规则：

- 在一个进程对数据更新完毕之前，不允许另一个进程获取同步变量
- 一个进程进行互斥访问，其他不能访问，哪怕是非互斥模式
- 一个进程进行互斥访问，除非变量拥有者执行完操作，否则不能进行下一个操

作，哪怕是非互斥模式

2.10 释放一致性

被保护数据： 需要保持一致的共享数据

`Acquire(L)` 操作：进入临界区

- 导入数据：使被保护数据的局部拷贝与远程的最新版本一致

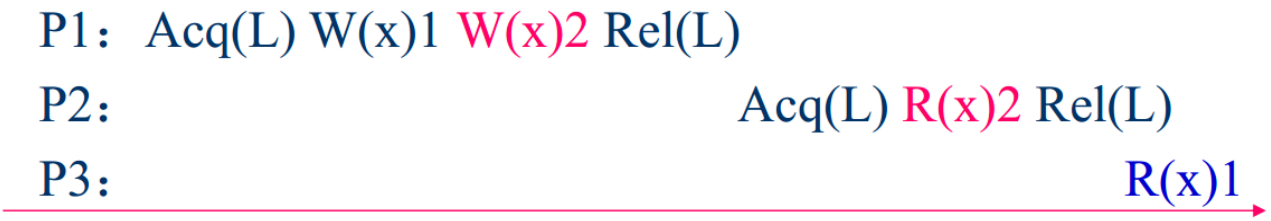
`Release(L)` 操作：退出临界区

- 导出数据：将被保护数据上的变化传播到其它的局部拷贝上

规则

- 在访问共享数据前，所有先前的 acquire 操作都必须完成。
- 在执行 release 前，先前的所有读写操作都必须完成。
- 对同步变量的访问必须满足 FIFO 一致性

例：符合释放一致性



及时释放一致性(EAGER release)

- 当执行了释放操作，执行此操作的处理机将所有修改的数据传给所有那些已经有其缓冲拷贝且可能需要它的处理机

滞后释放一致性（LAZY release ）

- 在执行释放时，不发送任何数据。
- 在执行获取操作时，处理机试图从拥有这些变量的机器上取得它们的最新值

2.11 入口项一致性

同步变量

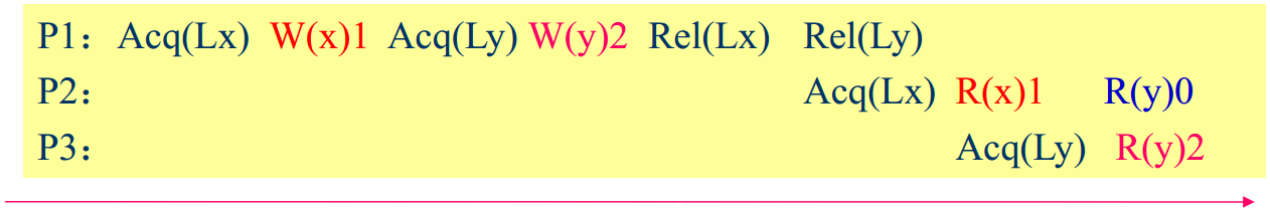
- 与某个共享数据项相关联 ； 不是与数据区中的所有保护型数据关联

- 拥有者(owner)：最后一个获取（acquire）它的进程。 其他进程必须从当前所有者手中取得拥有权。
- 非互斥方式（non-exclusive）：可以读，但不能写；个进程可以非互斥方式同时拥有一个同步变量

规则：

1. 在进程 P 获取同步变量 S 之前，有关的被保护的共享数据上的全部更新操作都必须完成；
2. 在进程 P 以互斥模式访问同步变量 S 之前，不允许其他进程同时拥有 S，即 使在非互斥模式下；
3. 在进程 P 以互斥模式获取同步变量 S 之后，任意其他进程都不能对 S 执行非互斥式访问，除非在 S 的拥有者 P 执行之后

例：入口项一致性



优点：

- 减少开销
- 增加并行性

2.12 总结

无同步操作

一致性	说明
严格	所有的共享访问事件都有绝对时间次序
顺序	所有进程都以相同的次序看到所有的共享访问事件，不按时间排序
因果	所有进程都以相同的次序看到所有因果联系事件
FIFO	所有进程见到的其他进程的写操作次序就是该进 程执行写操作的次序，但来自不同进程的写操作不必以相同的顺序看见

同步操作

一致性	说明
弱	当同步操作完成后,共享数据才保持一致
释放	当退出临界区后，共享数据才保持一致
入口项	当进入临界区时,和该临界区有关的 共享数据才保持一致

3.以客户为中心的一致性模型

分布式数据存储区

- 没有同时更新（无写-写冲突）或容易解决
- 大多数操作为读操作
- 例：Web 网页（服务器，代理缓存）

3.1 最终一致性（eventual consistency）

如果很长时间不发生更新操作，则所有的副本将逐渐变为一致的。

移动用户问题：

客户为中心的一致性（Client-centric）

- 保证对一个客户对数据存储的访问是一致的
- 考虑不同客户之间的并发访问

假设

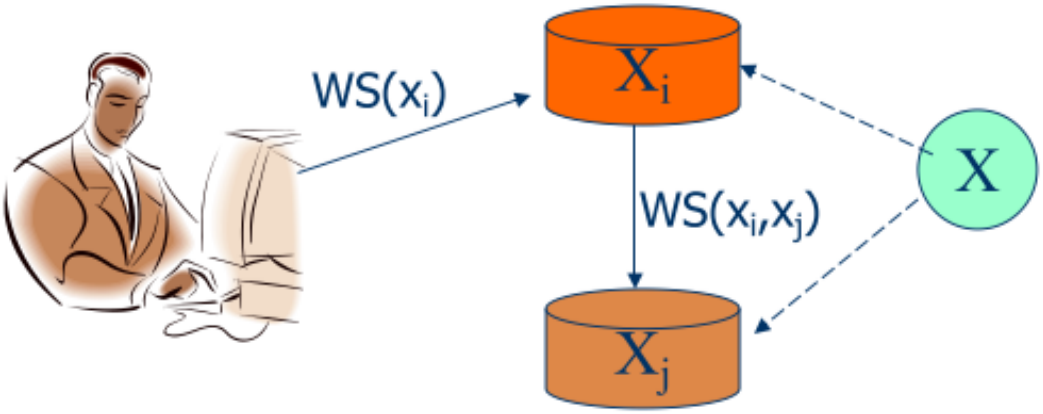
- 每个数据项 x 有一个拥有者，只有拥有者可以修改 x
- 客户的读写操作在本地副本上进行
- 更新最终将传播给其他副本上。

3.2 客户为中心的一致性

$x_i[t]$： 数据项 x 在局部场地 L_i 上，在时刻 t 的版本

$WS(x_i[t])$： 产生 $x_i[t]$ 的所有写操作的集合

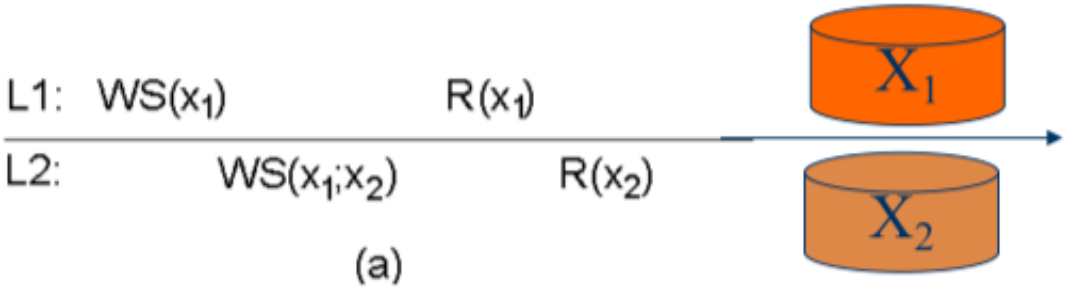
$WS(x_i[t1], x_j[t2])$： $WS(x_i[t])$ 中的写操作在 t2 时刻在 Lj 上执行



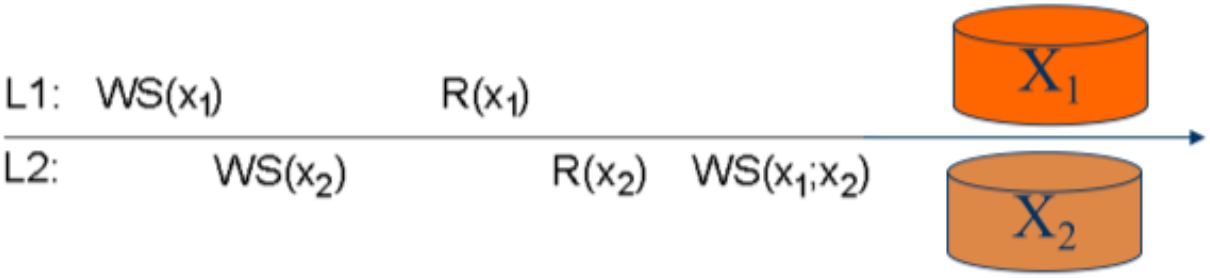
3.2 单调读一致性

当一个进程读了数据项 x 的值后，所有后续的对 x 的读操作，都将返回相同的值，或者更新的值

例：符合单调写一致性



例：不能保证单调写一致性

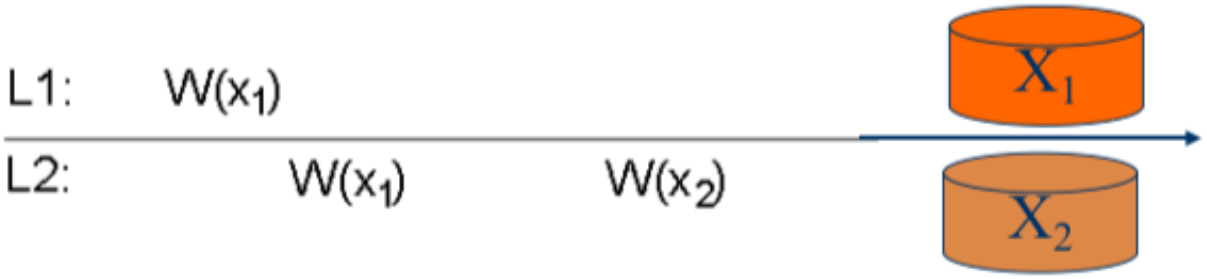


3.3 单调写一致性

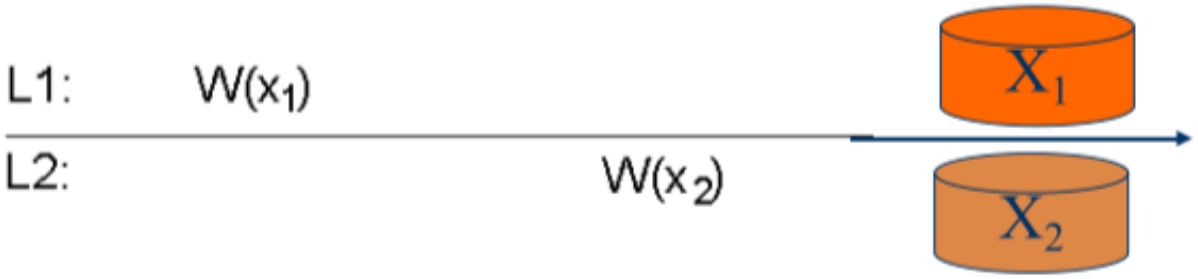
一个进程对数据项 x 的写操作，必须在该进程对 x 的所有后续写操作之前完成。

- 例：软件库更新（版本 1,...,n），增量写

例：符合单调写一致性



例：不能保证单调写一致性

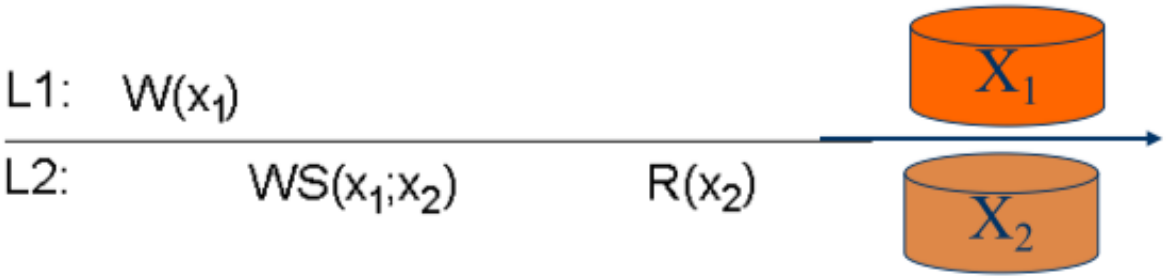


3.4 读自己写一致性

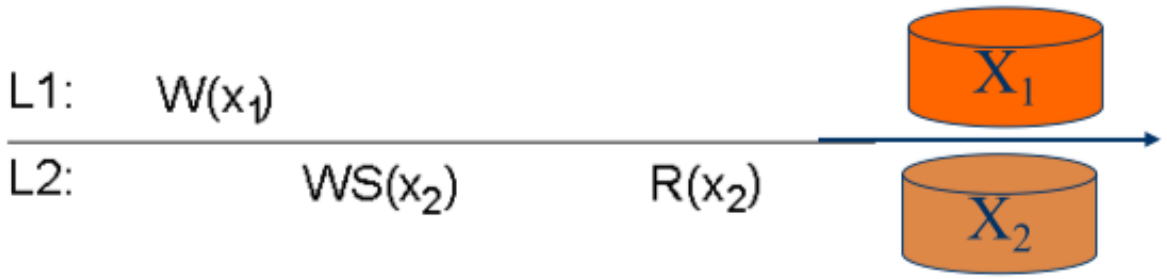
一个进程对数据项 x 的写操作结果，总能被该进程对 x 的后续读操作读到。

- 例 1：Web 网页更新、浏览
- 例 2：数字图书馆密码更新

例：符合读自己写一致性



例：不能保证自己写一致性

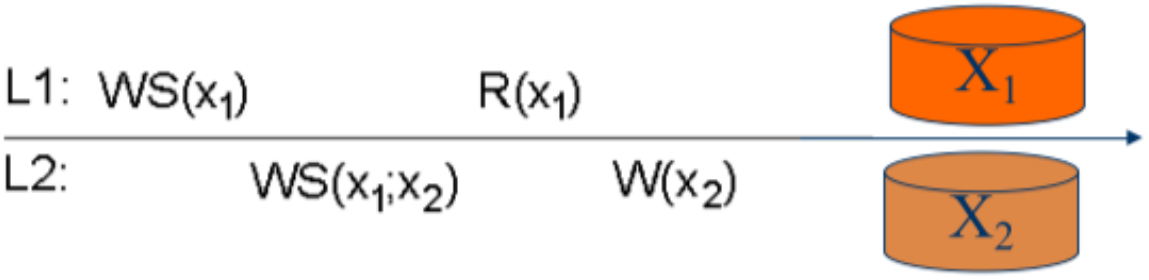


3.5 写跟随读一致性

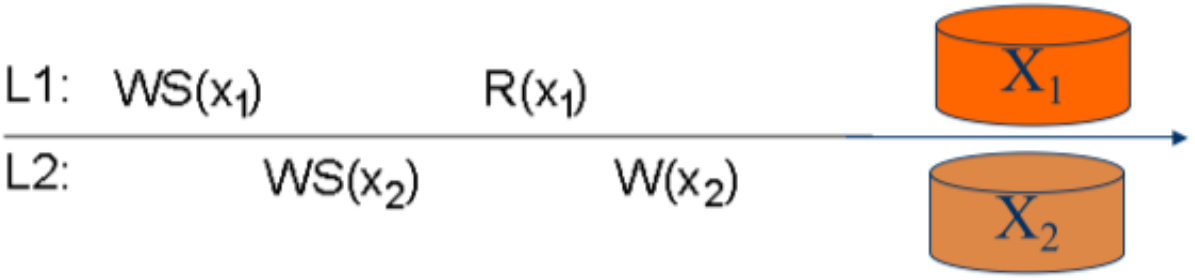
一个进程在对数据项 x 读操作之后对 x 的写操作， 必须在 x 已读出的相同值或者更近的值之上进 行

- 例：BBS 跟帖（读文章 A,写文章 B）

例：符合写跟随读一致性



例：不能保证写跟随读一致性



4.复制管理

两个子问题

1. 副本服务器位置选择：从多个可行的位置中挑选，放置一台服务器
2. 内容副本放置：从多个已经存在的副本服务器中挑选一台，放置一份内容的副本

4.1 副本服务器的放置策略

副本服务器的放置问题

- 从 N 个位置中选出 K 个最优位置

（1）基于距离的方法

- 最优：所有客户与所有位置的距离最短
- 距离：延时、带宽等指标

（2）基于自治系统的方法

- 不考虑客户的位置，假设客户在网络上均匀分布
- 自治系统（autonomous system, AS）.所有节点运行相 同的路由协议，且由单个组织管理。
- 在含有最大链接数量的网络接口的路由器上放置副本服务器

存在问题：

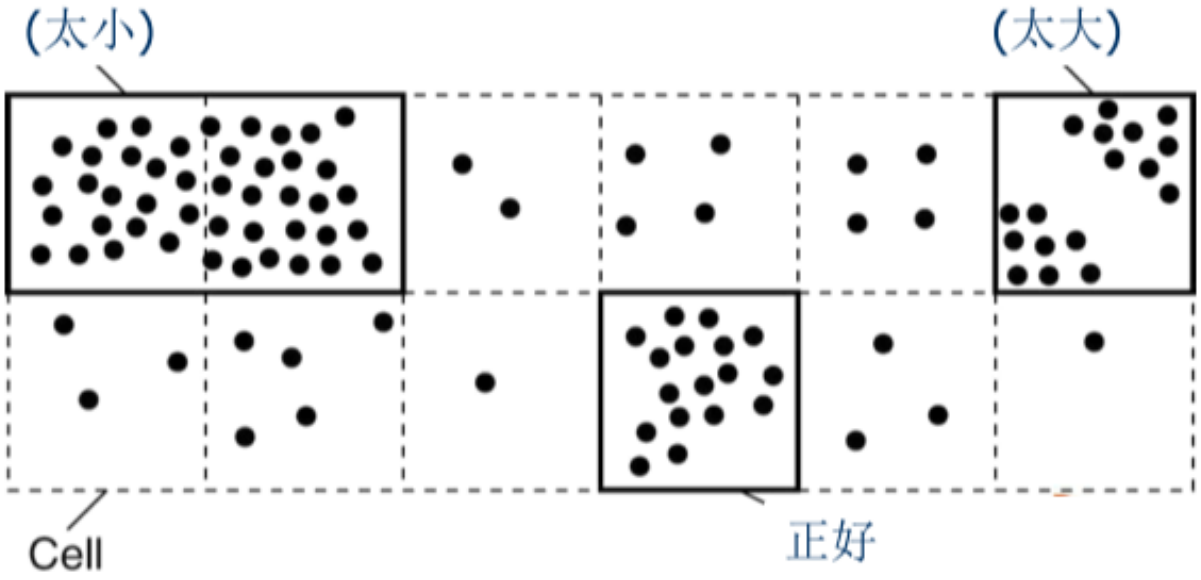
- 计算复杂度高 $> O(N^2)$
- 而在瞬时拥塞（flash crowds）情况下，要求快速布置副本服务器

（3）基于单元的方法

- 将 m 维空间划分成多个相同大小的单元
- 选择 K 个密度最大的单元放置副本服务器
- 计算复杂度： $O(N * max\{log(N), K\})$

选择适当的单元大小

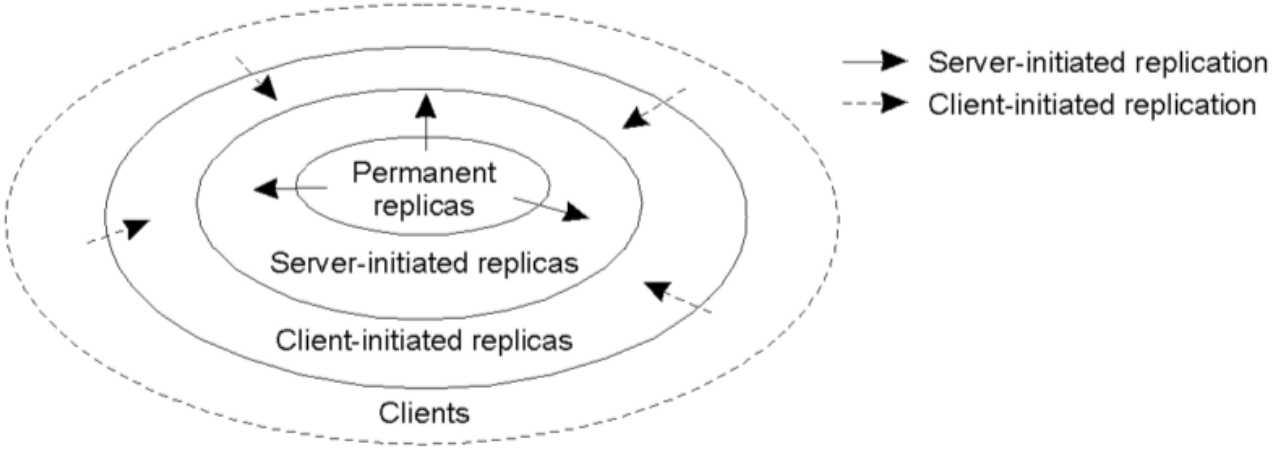
- 太大：一个单元格中集群太多，而副本服务器太少；
- 太小：一个集群被分成多个单元格，导致副本服务器太多



4.2 内容复制与放置

分布式数据仓的设计

- 复制副本的类型：永久型、服务器发起型、客户发起型



（1）永久性副本

构成分布式数据仓的初始集合

- 静态的、固定的
- 副本数量较少

例 1：Web 场地的分布类型

- 局域网：轮回策略型服务器
- Internet：镜像服务器

例 2：分布式数据库

- 工作站集群（COW）
- 联邦数据库

（2）服务器发起型副本

推送式缓存（push cache）

- 由服务器，动态地设置新的副本

作用：

- 当负载发生变化时，如突然增加。
- 减少服务器负担
- 减少客户的通信开销

问题?

- 在何时、何地发起复制

实现方法

- 访问计数： $cnt(S,F)$ ， S 为服务器，F 为文 件
- 复制阈值： $rep(S,F)$
- 删除阈值： $del(S,F)$
- 距离： $dist(S,C)$, C 为客户。该信息来自路由数据库

复制副本

- 复制副本条件： $cnt(S,F) > rep(S,F)$

删除副本

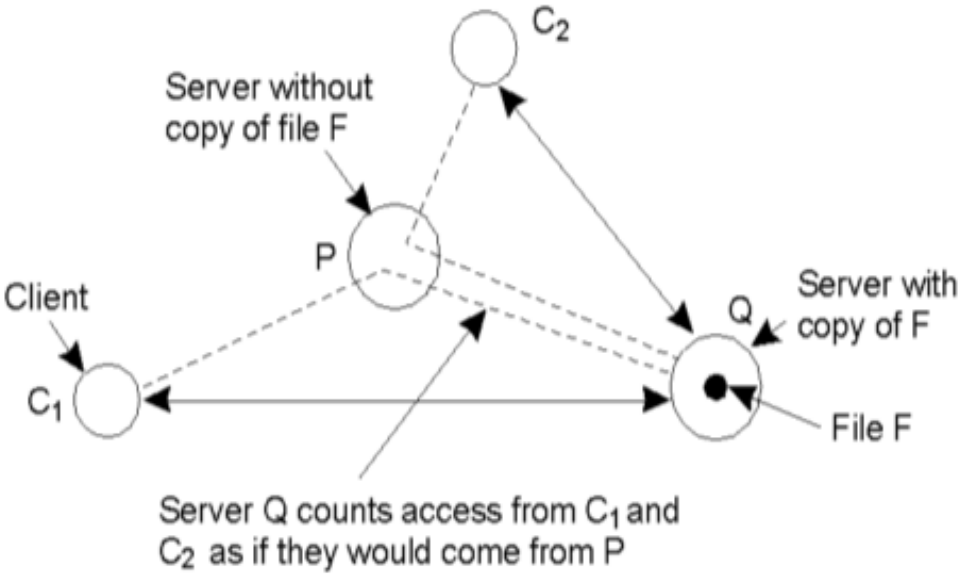
- 删除副本条件： $cnt(S,F) < del(S,F)$

迁移副本

- 迁移副本条件： $del(S,F) < cnt(S,F) < rep(S,F)$

举例：计数来自不同客户的请求，将文件复 制到离客户附近的服务器上。

如果 $cntQ(P,F) > cnt(Q,F)/2$ ，将 F 从 Q 迁移到 P



（3）客户端发起副本

客户缓存(cache)： 客户端的本地存储

缓存命中率(cache hit)：

- 请求的数据可在缓存中取出的概率
- 提高命中率：缓存可由多个客户共 享

客户缓存的设置场地

1. 与客户相同的机器
2. 局域网上多个客户共享的机器上
3. 广域网上的代理服务器上

4.3 内容分发

当客户执行一个更新操作后，该操作将传播到所有副本

传播状态与传播操作策略

1. 传播更新通告
2. 传输数据拷贝
3. 传播更新操作

（1）更新传播

通告无效协议（invalidation）

- 只传输被修改的数据的位置信息
- 数据量少，占用很少网络带宽
- 适用于读/写比非常低的情况

数据传输(data shipping)

- 传输被修改的数据
- 数据量多，占用较多网络带宽
- 适用于读/写比非常高的情况

操作传输(operation shipping)

- 主动复制技术 – 每个副本有一个进程主动地进行更新
- 占用最少网络带宽
- 要求有较高处理能力

（2）推送式与拉取式协议

推送式协议：基于服务器的协议

- 不需要请求，就将更新传播给副本
- 可保持高度的一致性，通常用于永久性副本和服务器副本之间
- 优点：适用于读/写比非常高的情况

拉取式协议：基于客户的协议

- 由客户请求服务器发送更新
- 优点：适用于读/写比非常低的情况
- 缺点：当 cache miss 时，响应时间长

（3）推送式协议与拉取式协议的比较

项目	推式协议	拉式协议
服务器的状态	客户端副本和 Cache 的清单	无

发送的消息	更新（以及稍后读取数据）	轮询和更新
客户端响应时间	立即（或读取被更新数据时间）	读取被更新数据时间

（4）推拉混合式方法

- 基于租期的更新传播方法
- 租期（lease）： 服务器承诺在租期时间内向客户传播更新(推送式)。
- 当租期过期后， 客户申请新的租期， 或自己取修改数据（拉取式）

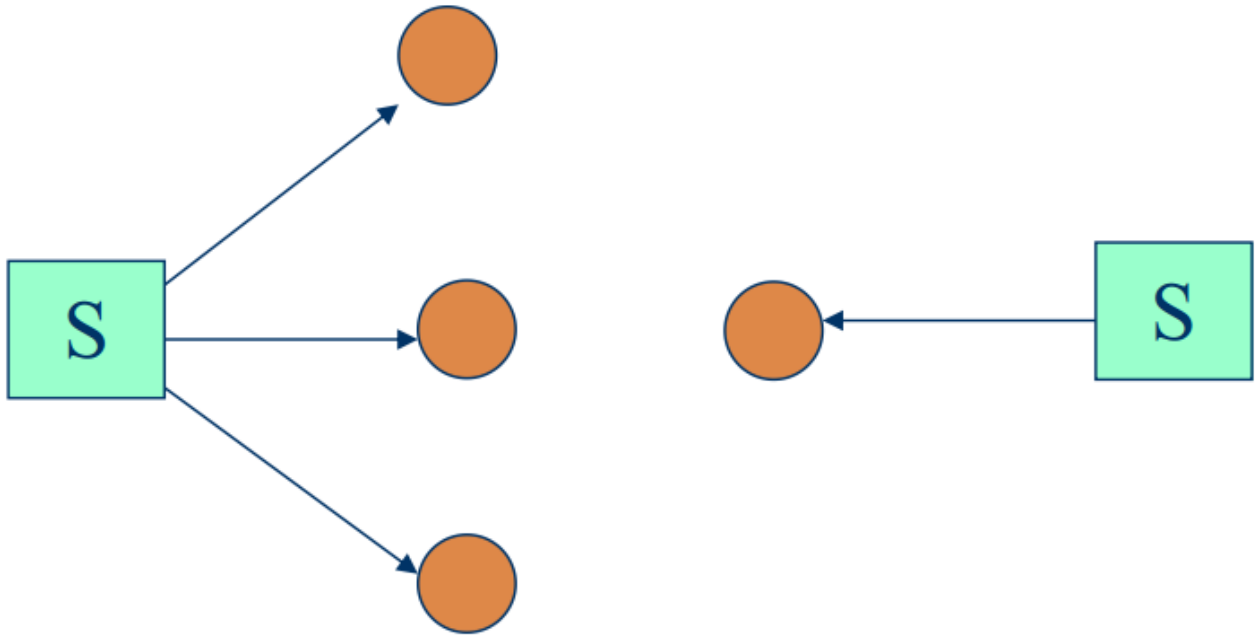
设置租期的准则

- 基于年龄的租期： 对年龄大(不经常修改)的数据， 租期长。
  - 年龄： 数据项最后一次修改后的延续时间。
  - 假设长时间未被修改的数据，今后也不打可能修改
- 基于刷新频率的租期： 对经常被刷新的缓存中(经常使用的)数据， 租期长。
- 基于状态空间开销的租期： 当服务器负载小时， 租期长。

4.4 单播方式与多播方式

多播 ： 一对多通信；适用于从服务器到客户的推送式协议

单播：一对一通信；适用于从客户到服务器的抽拉式协议



5.一致性协议

一致性协议： 对一致性模型的实现方法的描述

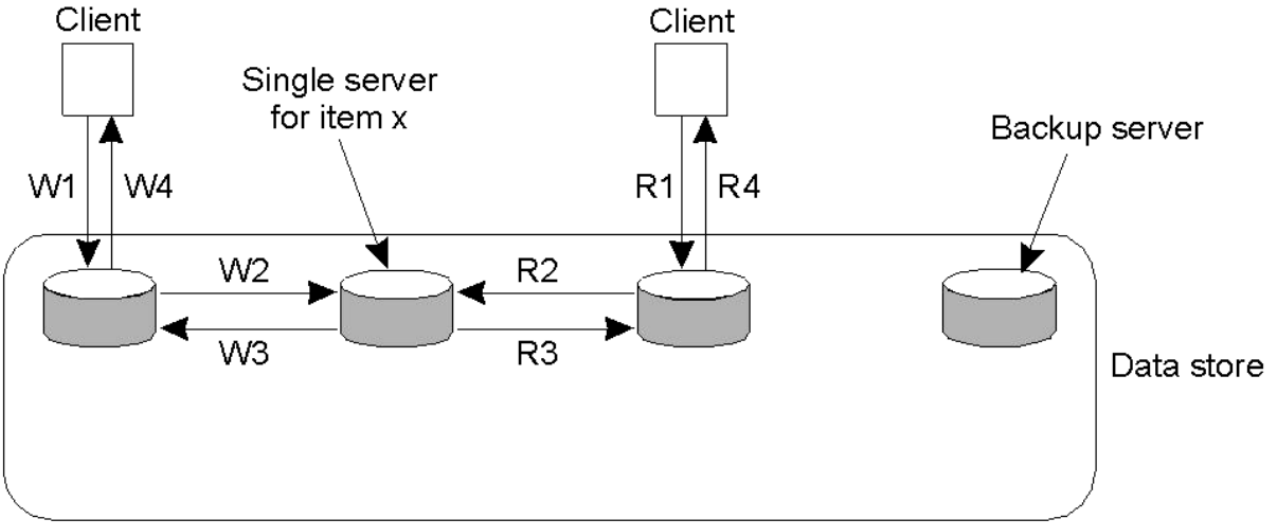
基于主副本的协议

- 主副本： 在数据的所有复制副本中， 写操作必须先在主副本上进行。
- 实现了顺序一致性
- 远程写协议： 所有写操作由远程服务器执行
- 本地写协议： 将主副本读到执行写操作的本地上执行

5.1 远程写协议

（1）单个主副本服务器

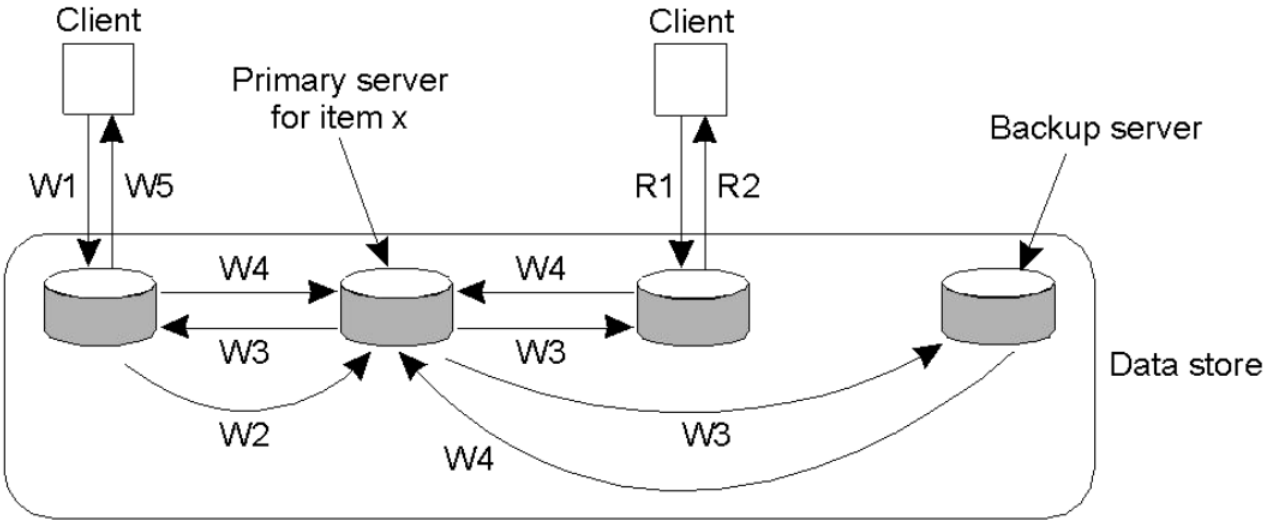
读操作在本地执行，写操作在远地执行



W1. Write request
W2. Forward request to server for x
W3. Acknowledge write completed
W4. Acknowledge write completed

R1. Read request
R2. Forward request to server for x
R3. Return response
R4. Return response

（2）带有备份的主副本服务器



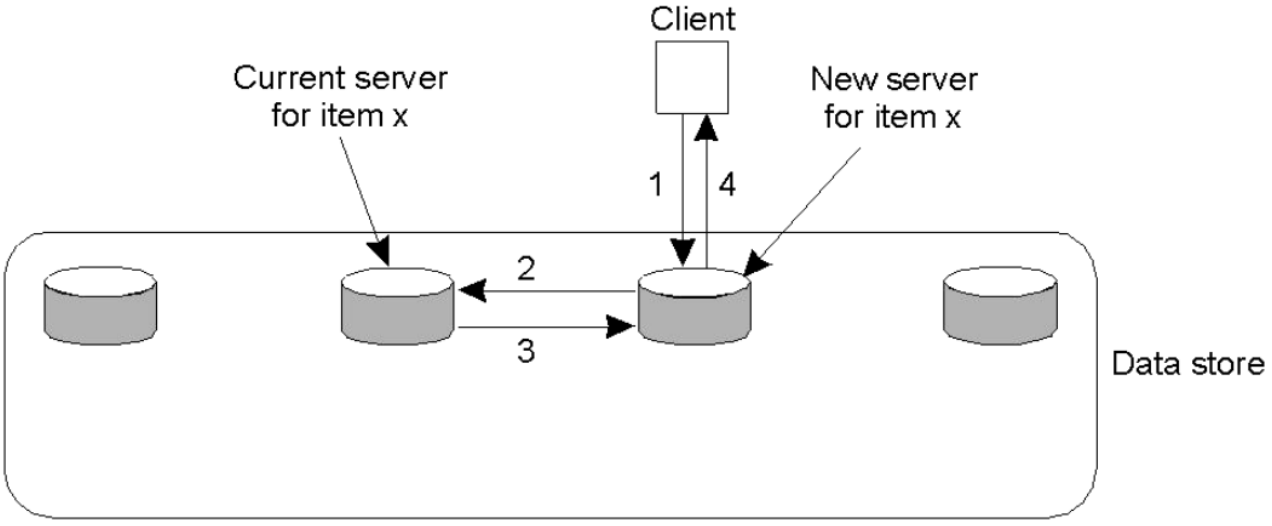
W1. Write request
W2. Forward request to primary
W3. Tell backups to update
W4. Acknowledge update
W5. Acknowledge write completed

R1. Read request
R2. Response to read

5.2 本地写协议

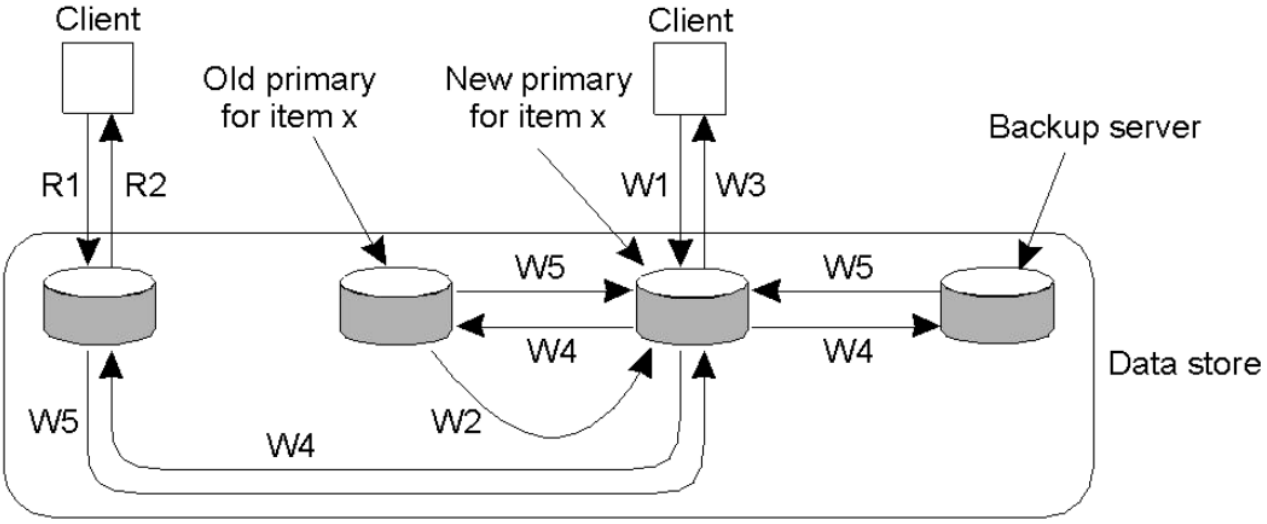
(1) 单个主副本服务器

主副本迁移到当前写操作的场地上



1. Read or write request
2. Forward request to current server for x
3. Move item x to client's server
4. Return result of operation on client's server

(2) 带有备份的主副本服务器



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

5.3 复制式写协议

写操作可在多个副本上执行

主动复制协议：将各更新操作发给各个副本上的进程

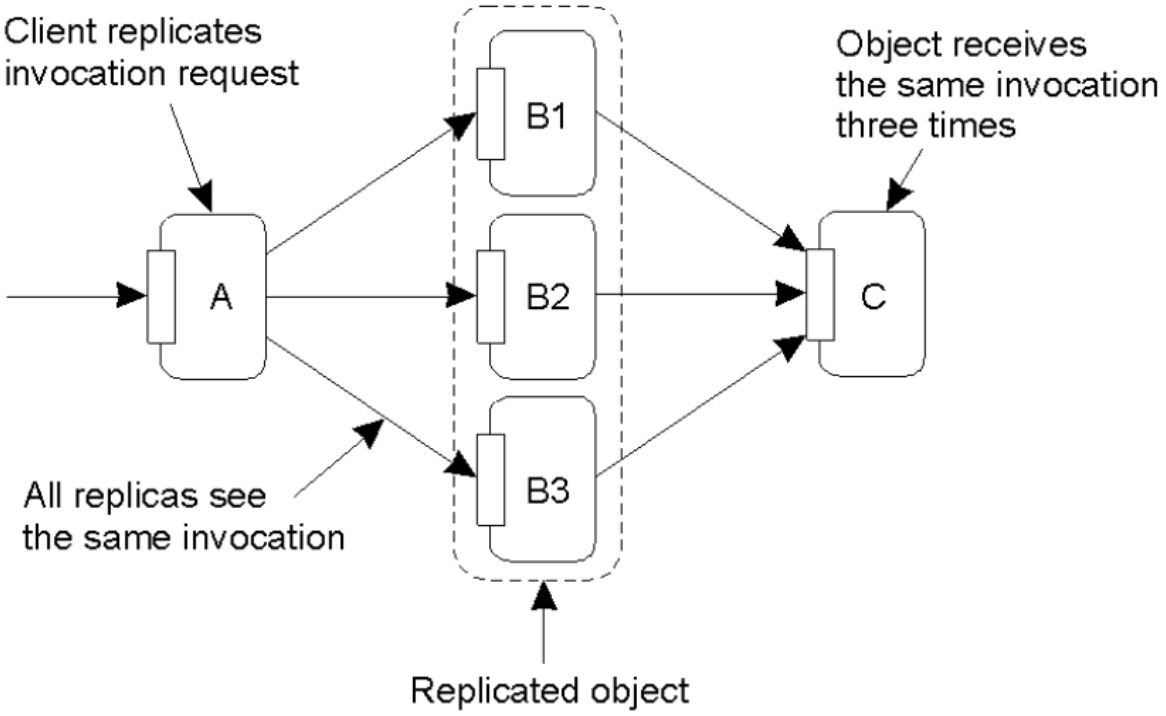
问题 1： 更新顺序问题

解决方案：

- 全序多播机制： 如 Lamport 时间戳向量
- 顺序管理器（sequencer） ： 集中式协调器， 负责为每个操作赋予唯一的顺序号， 转发给各个副本

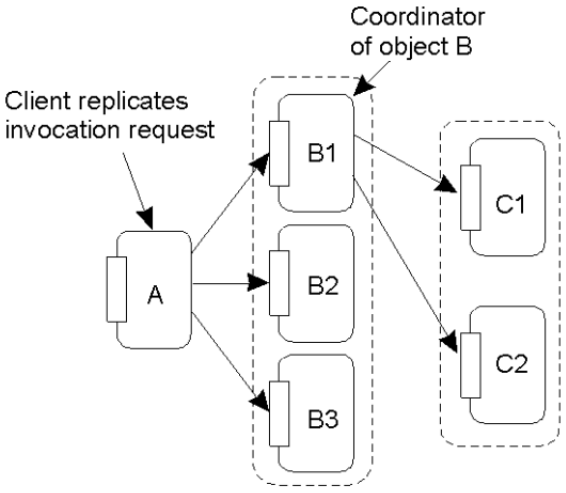
问题 2：重复调用问题

- 对象副本的重复调用
- 服务器的重复调用

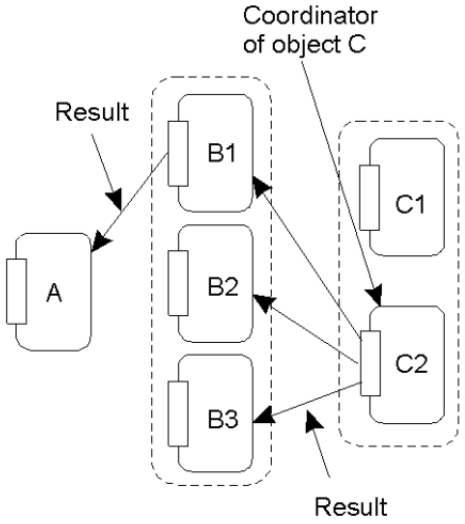


解决方案： 设置了解副本的通信层

- 基于发送者模式： 由协调者决定发送调用/响应



(a)对复制对象的调用



(b)复制对象返回的应答

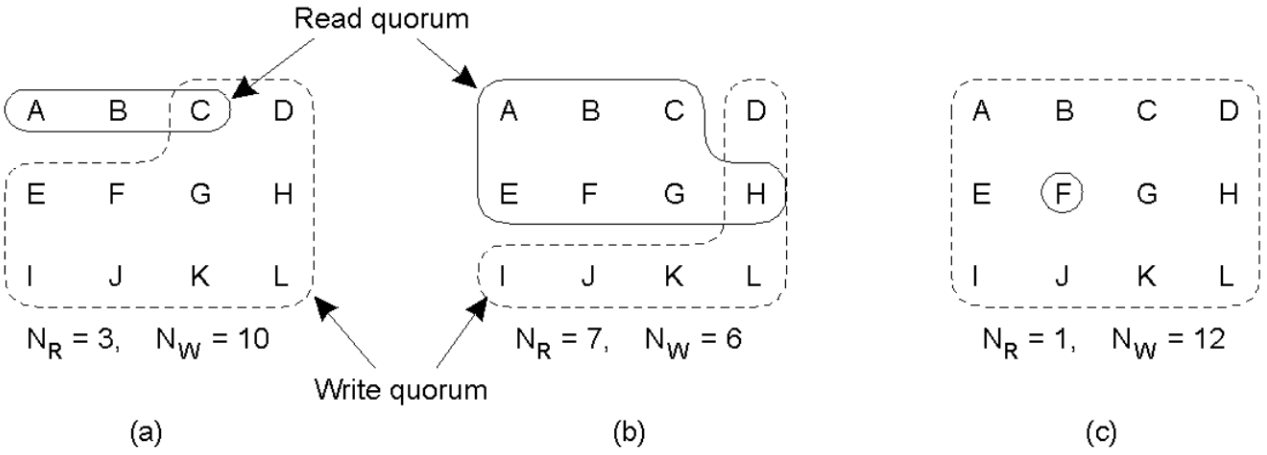
5.4 基于合法数的协议

基于多数表决的复制写协议

- 与主副本协议的区别： 多个副本同时执行写操作

基本算法：

- 设有 N 个副本
- 设置读合法数 N_R， 写合法数 N_w
- 要求： $N_R + N_w > N$； $N_w > N/2$



正确读写合法数　　不正确写合法数　　ROWA协议

5.5 以客户为中心一致性的实现技术

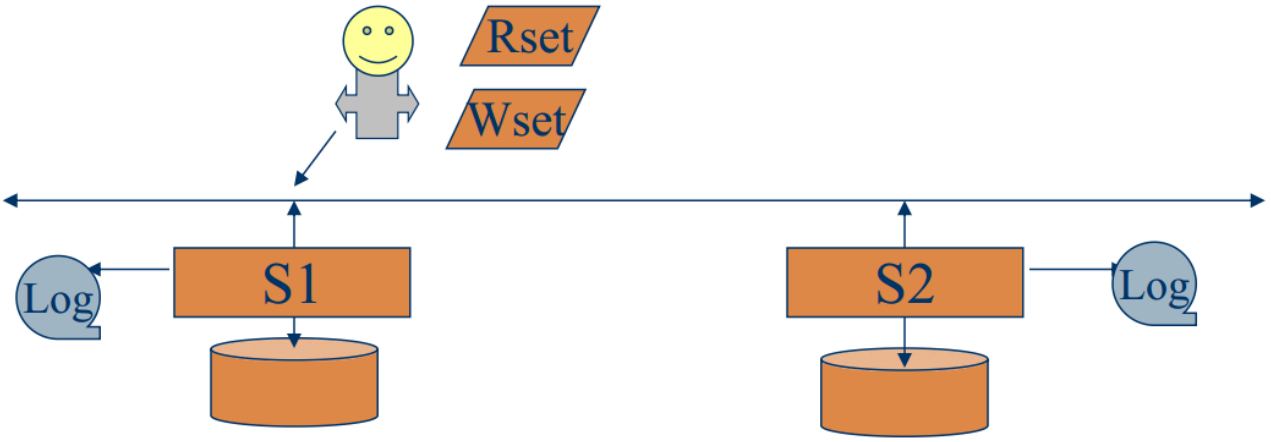
（1）基本方法

写操作： 全局唯一标识符。

- 服务器 ID：执行位置
- 序号：执行顺序

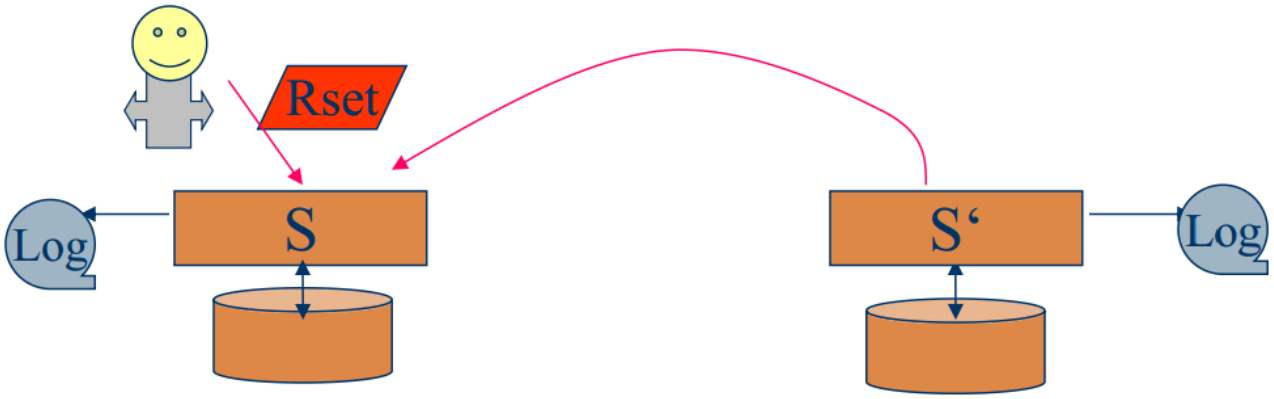
读集合 Rset(c)： 与客户 c 读操作相关的写操作标识符

写集合 Wset(c)： 客户 c 执行的写操作的标识符



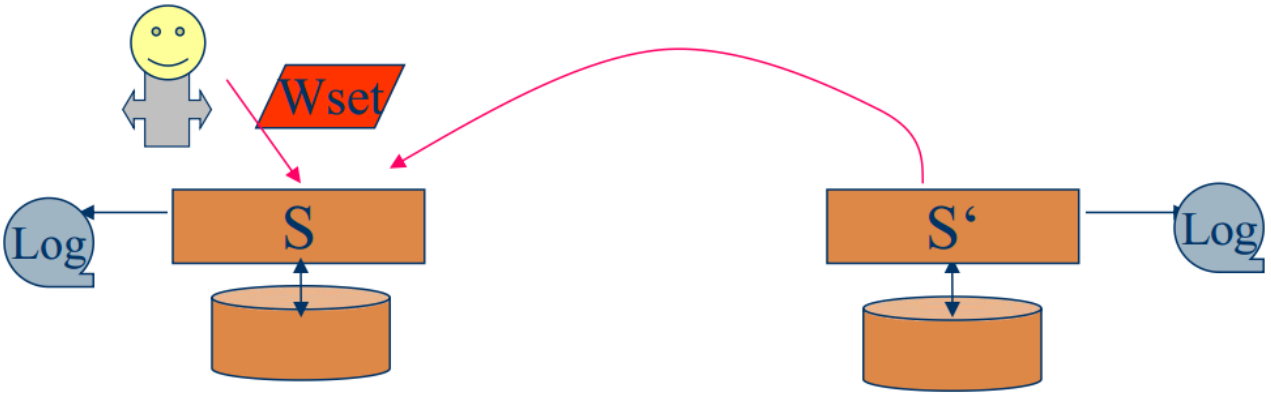
（2）实现技术：单调读一致性

- 当客户 c 执行读操作 r；
- c 的服务器 s 检查 c 的读集合 Rset(c)， 是否所有写操作已在 c 的本地更新；
- 如果 c 的本地没有更新， 则请求更新；
- 执行读操作 r；
- 将与 r 有关的写操作标识符加入 Rset (c)。



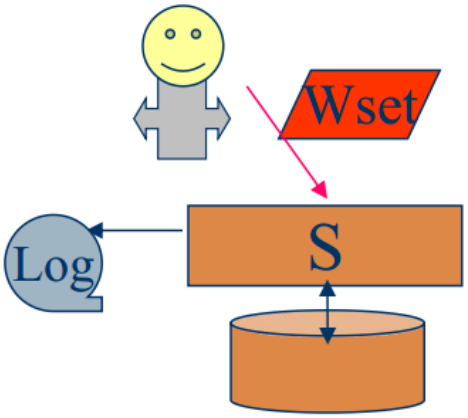
（3）实现技术：单调写一致性

- 当客户 c 执行写操作 w;
- 它的服务器 s 检查 c 的写集合 Wset (c)， 是否所有写操作已在 c 的本地执行；
- 如果在 c 的本地没有执行， 则请求执行；
- 执行写操作 w;
- 将 w 的标识符加入 Wset (c)。



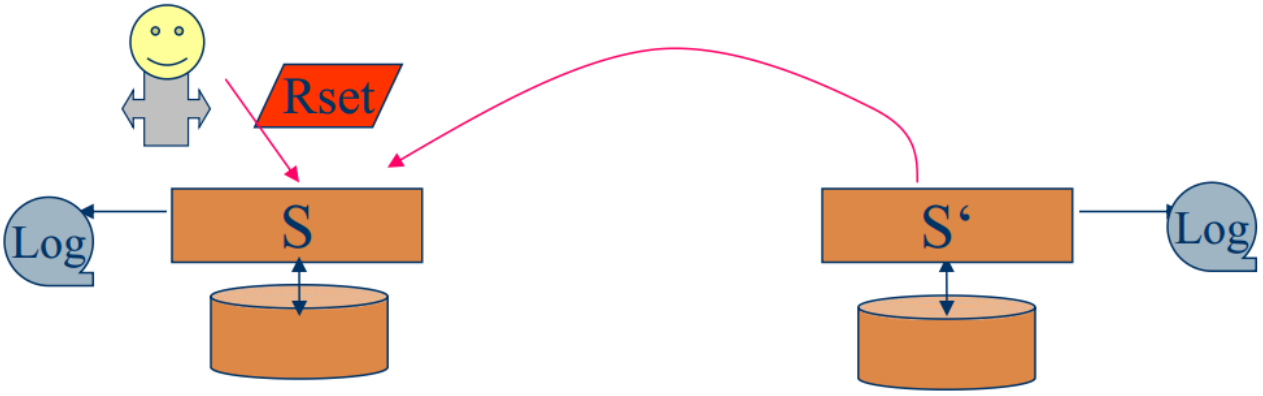
（3）实现技术：读自己写一致性

- 当客户 c 执行读操作 r；
- c 的服务器 s 检查 c 的写集合 Wset (c)， 是否所有写操作已在 c 的本地执行；
- 如果在 c 的本地没有执行， 则请求执行；
- 执行读操作 r;



（4）写跟随读一致性

- 当客户 c 执行写操作 w 时，
- c 的服务器 s 检查 c 的读集合 Rset (c)， 是否所有写操作已在 c 的本地执行；
- 如果在 c 的本地没有执行， 则请求执行；
- 将这些写操作标识符加入 Rset (c)。



（5）优化方法

性能问题 ： Wset(c)和 Rset(c)将非常大， 造成性能降低

基于会话(session)的优化方法

- 减少写集和读集中的元素个数

- 定义 session: 打开， 关闭操作。 对应一个应用程序的开始和结束。
- WS(c)和 RS(c)在 session 打开时建立， 关闭时清除。

基于时间戳向量的优化方法

- 减小写集和读集表示形式的大小
- 对一个写操作 WID,赋予时间戳 ts(WID)
- 服务器 Si 维持一个时间戳向量 $RCVD(i)$ 。 $RCVD(i)[j]=Si$ 收到的来自 Sj 的最近写操作的时间戳。
- 对写集或读集 A 维持一个时间戳向量 VT(A)。 VT(A)[i]为在 Si 上的最大时间戳
- 并操作： $VT(A+B)[i]=max\{VT(A)[i],VT(B)[i]\}$
- 包含关系： $VT(A)\leq VT(B)\leftrightarrow VT(A)[i]\leq VT(B)[i]$

例： 单调读一致性

- 设客户 c 在服务器 Si 上执行读操作 r;
- c 的读集为 VT(Rset)， Si 有 RCVD(i)
- 如果 RCVD(i)[j] ≤VT(Rset)[j]， 则在 Si 进行更新
- 执行后， VT(Rest)[j]=max{VT(Rest)[j], RCVD(i)[j]}

5.6 连续一致性协议

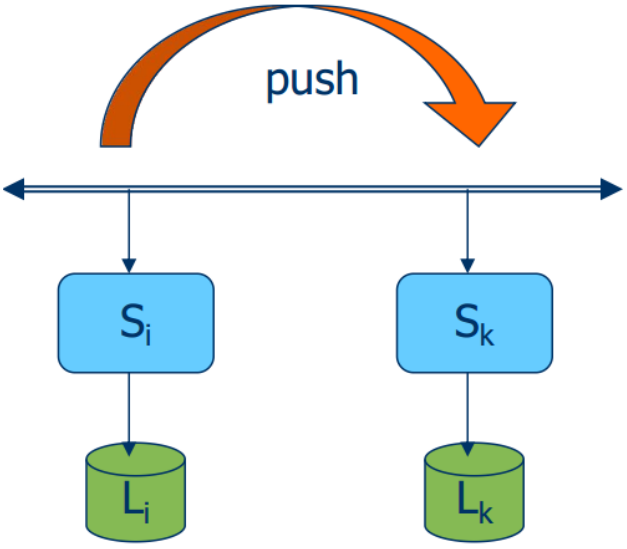
（1）数值误差限制实现

记号：

- 服务器 S_i ； 日志 L_i
- 写操作 $W(x)$
- weight(W)，W 的权重值
- Orign(W)，W 的源站点
- $TW[i,j]$ 源自 S_j ，传输至 S_i 的写操作； $TW[i,j]=\sum\{weight(W)|orgin(W)=S_j\&W\in L_i\}$

- v(t)为实际值，vi：S_i 的当前值

当不满足 $v(t) - vi \leq \delta_i$ 时，进行更新传播，执行感染协议。



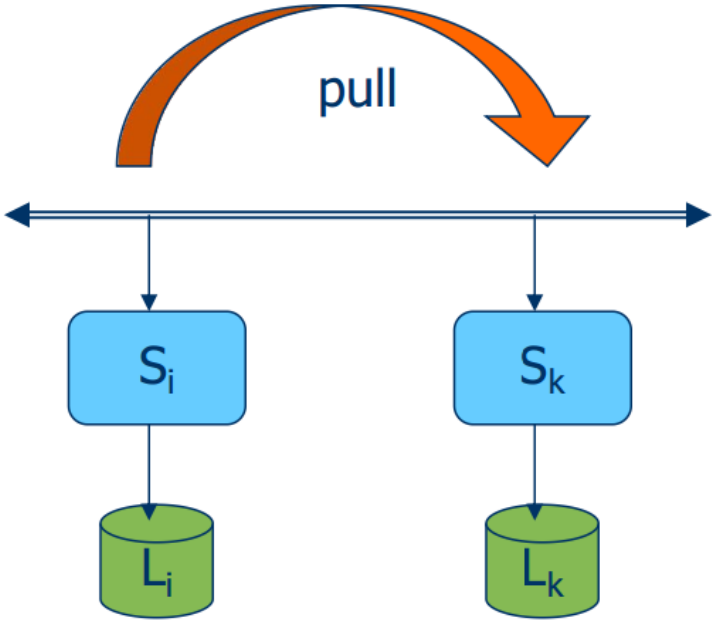
$$v(t) = v(0) + \sum_{k=1}^{N} TW[k,k]$$

$$v_i = v(0) + \sum_{k=1}^{N} TW[i,k]$$

（2）陈旧度限制的实现

记号：

- RVCi[] :服务器 Si 实时向量时钟
- T(i): 服务器 Si 的本地时间
- RVCk[j]： 已接受的 Sj 写操作时间戳
- di: 允许的最大陈旧度
- 当不满足 $T[i] - RVCi[j] \leq di$ 时， 进行更新传播， 拉取 Sj 的写操作



（3）次序误差限制的实现

记号：

- Q_i：服务器 Si 的暂时写操作队列
- length(Qi)： Qi 的长度

当不满足 $length(Q_i) \leq l_i$ 时， 提交本地的暂时写操作

提交操作

- 要进行一致性检验。 如不通过， 需回滚 Si 的暂时写操作
- 要保证全局次序