第7讲 一致性和复制

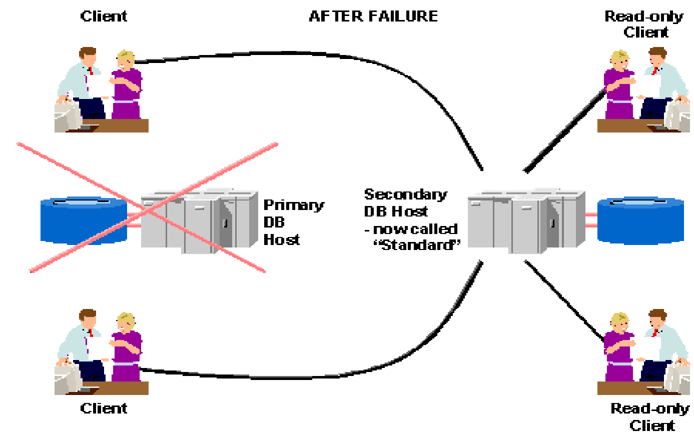
§7.1 复制与副本管理

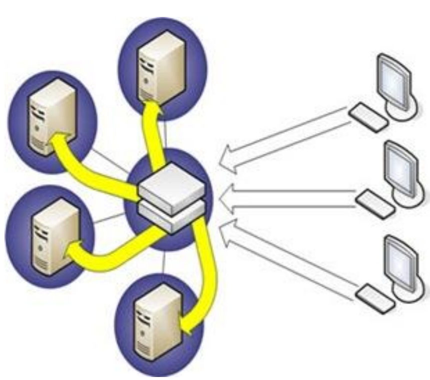
§7.2 副本一致性模型

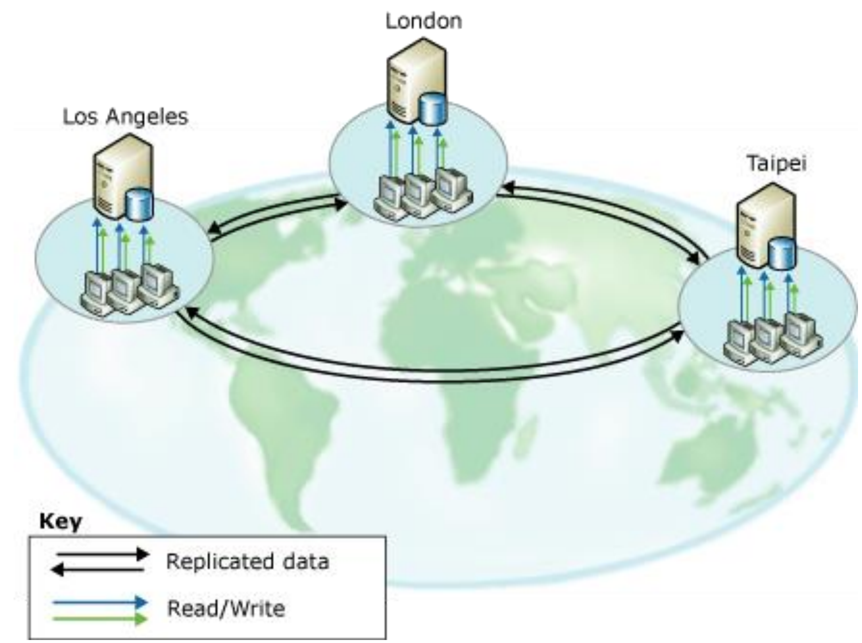
§7.3 副本一致性协议

§7.1 复制与副本管理

- Replication：
  - 数据或者资源部署多个副本
  - 每个副本都向客户端服务
    - 与备份不同
- 作用：
  - For Reliability
  - For Scalability
    - Number, location

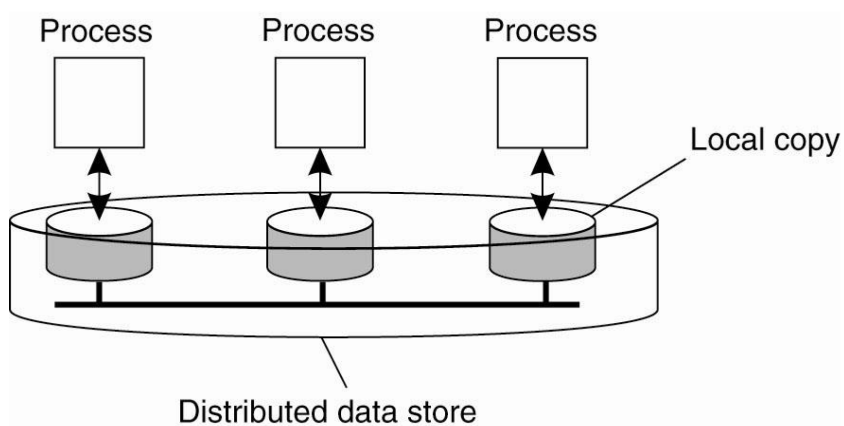






Major Issues in Replication

- Replica placement
  - 何处、何时、由谁来负责副本
  - Tradeoff between access benefit and update cost
- Consistency among replicas
  - 何种机制来保持副本的一致性
  - 确保冲突的操作在所有副本按照相同的顺序执行
    - 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



Replica Management

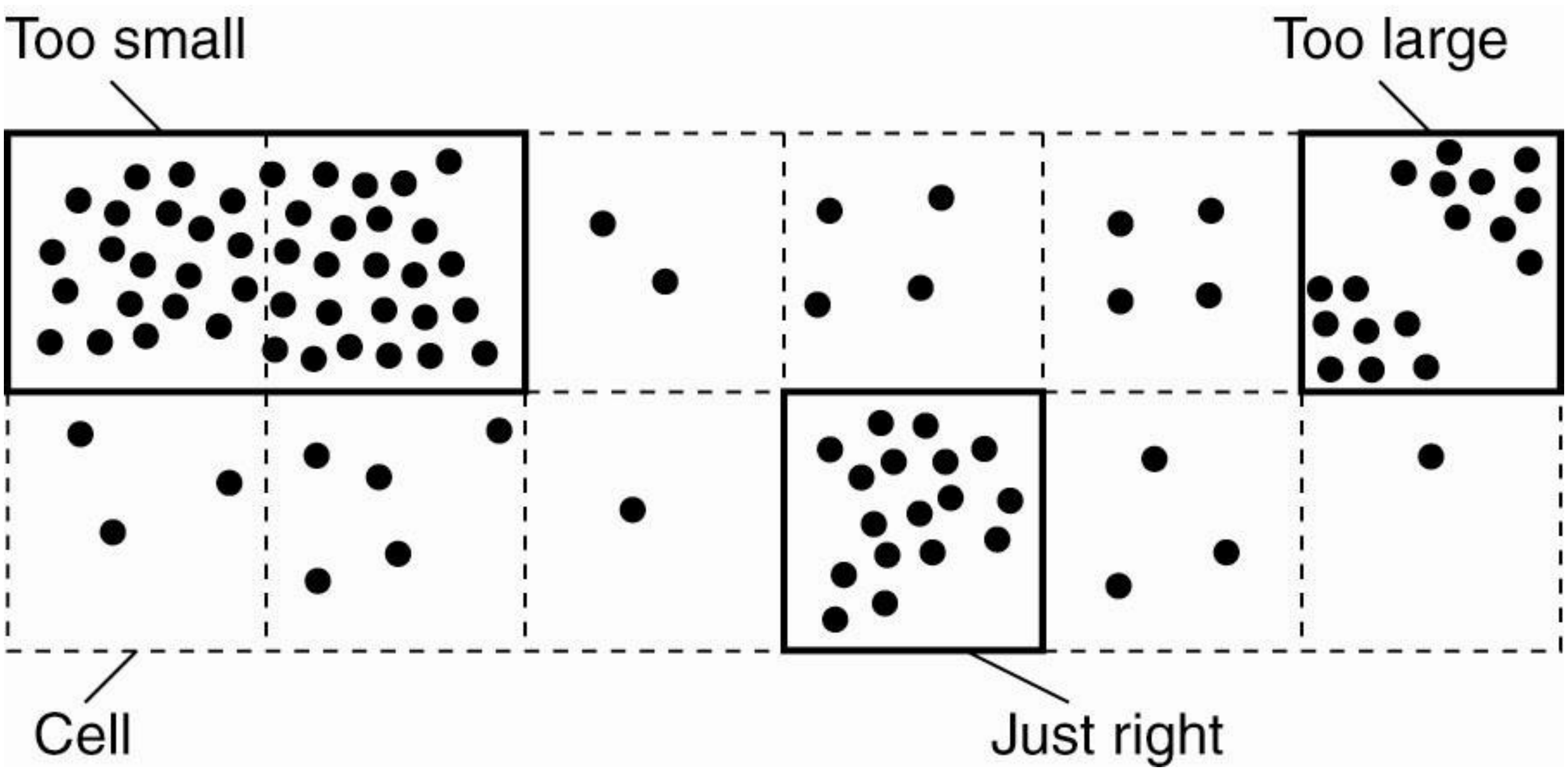
- To decide where, when, and by whom replicas should be placed
  - Replica Server Placement
    - Finding the best locations for replica servers
  - Data Content Placement
    - Finding the best servers for placing content
- To distribute content to replicas

Replica Server Placement

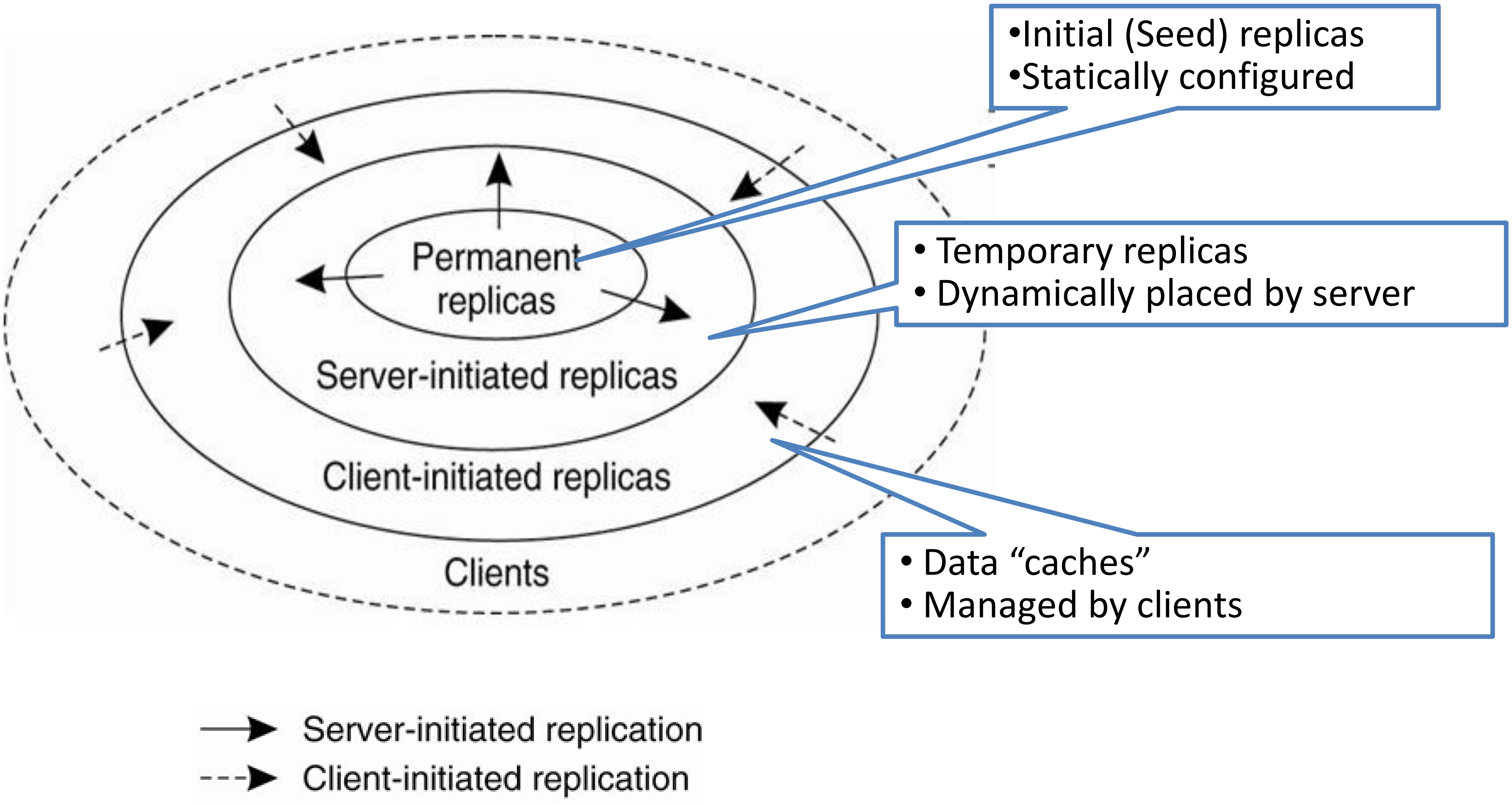
- *K* out of *N* locations
- Client-aware method: high complexity
  - Objective function: average distance between clients and servers
- Client-unaware method: lower than previous, but still high complexity
  - Assuming clients are uniformly distributed
  - Greedy: choose routers with the largest links
- Region-based method: low complexity
  - A region is identified to be a collection of nodes accessing the same content, but for which the internode latency is low.

Region-based method

- The entire space is partitioned into cells.
- The *K* most dense cells are then chosen.

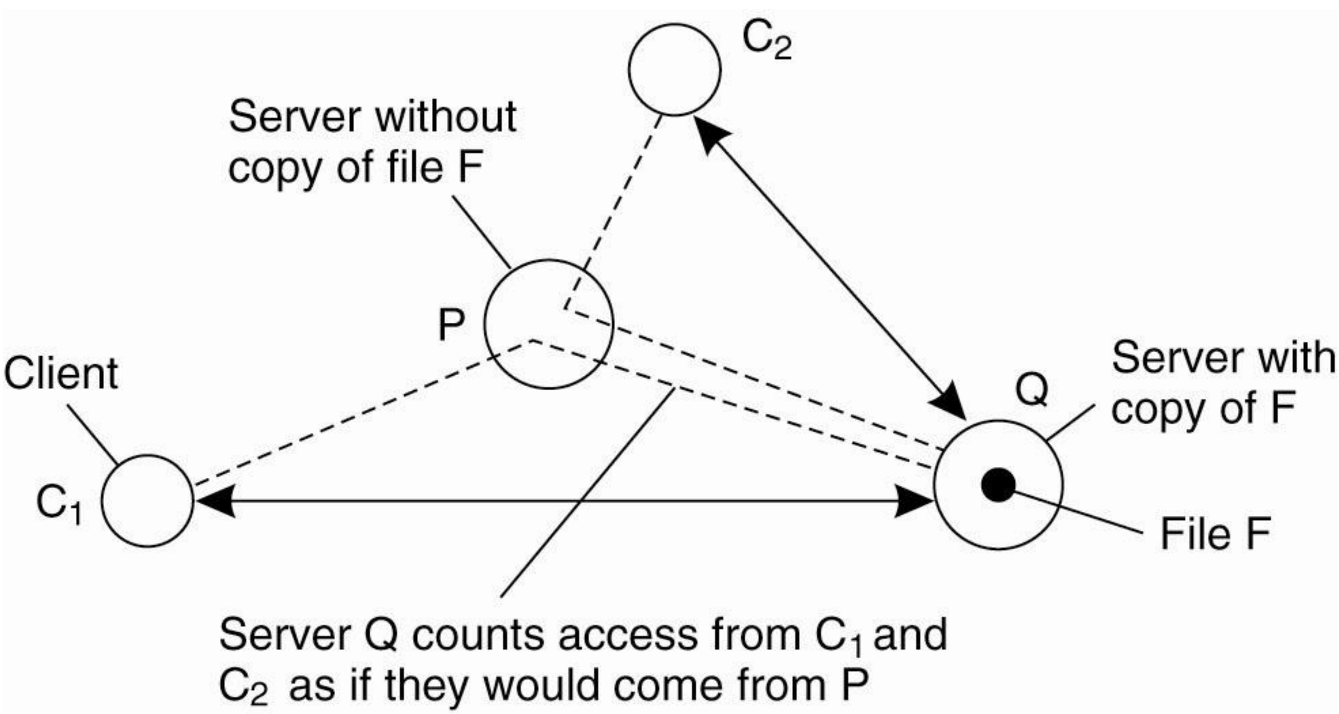


Data Content Placement



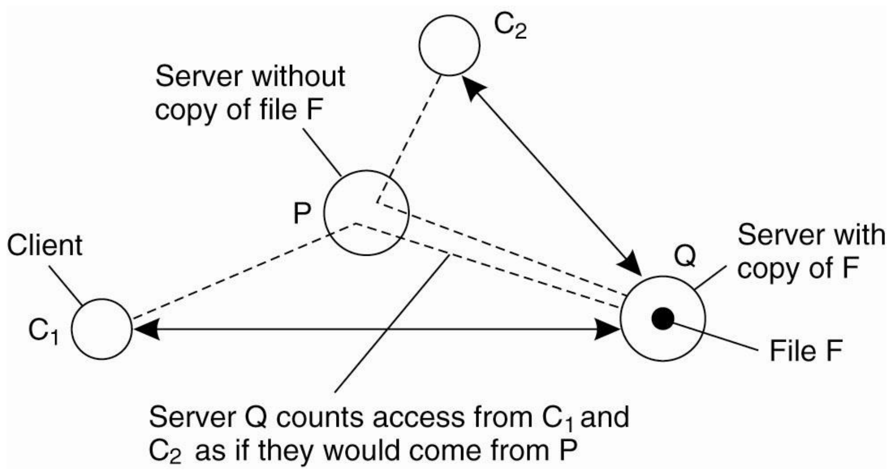
Server-initiated Replicas

- Two considerations
  - To reduce the load of servers – near server
  - To reduce the access delay at clients – near client



Server-initiated Replicas

- 对来自不同客户端的请求计数



  - 记录每个文件/数据的访问次数，当作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；

Client-initiated Replicas

- Client determine what to cache
- Servers may be involved for consistency
- Caches may be shared by more than one clients

Content Distribution

- To propagate update to replica servers

- What to distribute

  1. Propagate only a notification of an update.
  2. Transfer data from one copy to another.
  3. Propagate the update operation to other copies

- 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

Content Distribution

- How to distribute
  - Push: suitable for permanent/server-initiated replicas
    - High read-to-update ratio
  - Pull: suitable for caches
    - Low read-to-update ratio

Issue	Push-based	Pull-based
State at server	List of client replicas and caches	None
Messages sent	Update (and possibly fetch update later)	Poll and update
Response time at client	Immediate (or fetch-update time)	Fetch-update time

Content Distribution

- How to distribute:
  - Lease-based: hybrid of push + pull
    - Lease（租约）是服务器所作的承诺
    - 在lease指定的时间内服务器会把更新推给客户
    - lease到期则需要客户端通过pull方式更新
  - 关键问题：确定租约期限

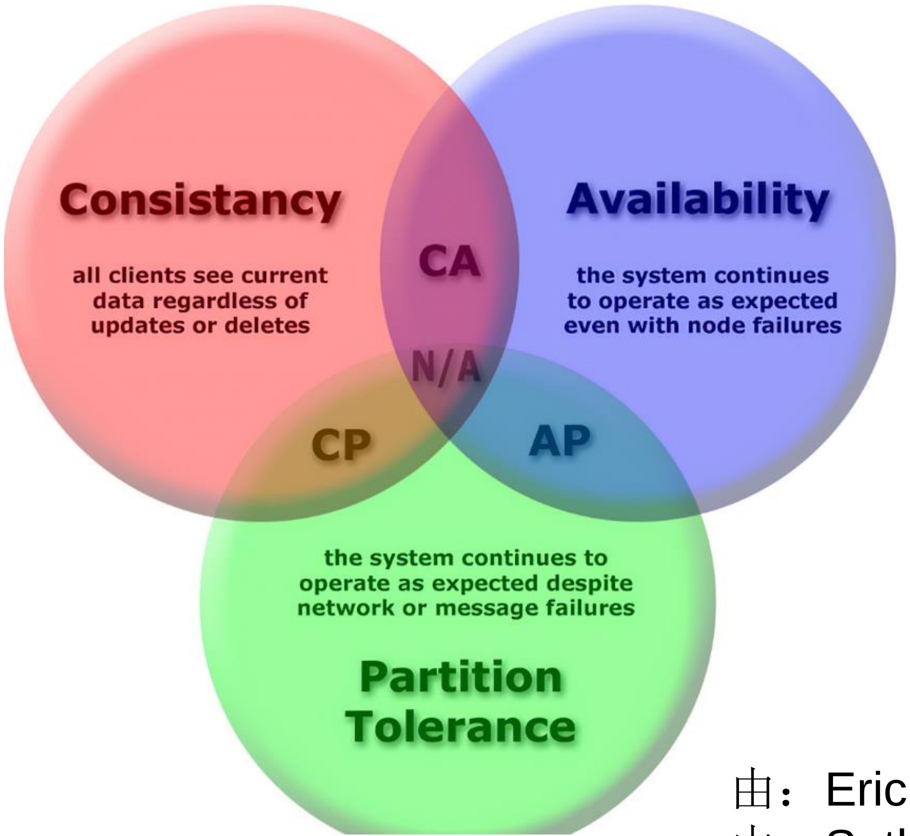
- Age-based leases: An object that hasn't changed for a long time, will not change in the near future, so provide a long-lasting lease
- Renewal-frequency based leases: The more often a client requests a specific object, the longer the expiration time for that client (for that object) will be
- State-based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

§7.2 Consistency Models

- Consistency Model
  - A contract between processes and the data store.
  - It says that: if processes obey certain *rules*, the store promises to work *correctly*.
- "Correctness":
  - If a process performs a read operation on a data item, the operation should return a value that shows the results of the last write operation on that data.
- Which is the last write, without global clock?
  - Consistency model defines the values that a read operation can return.
  - Tradeoff between *consistency level* and *maintenance cost*

CAP理论

- 指一个分布式系统中 CAP三者不可得兼



①**一致性：**客户端的读操作要么读到最新的数据，要么读取失败。

②**可用性：**任何客户端的请求都能得到响应数据。

③**分区容忍性：**当消息丢失或延迟到达时，系统仍会继续提供服务，不会挂掉。

由：Eric A. Brewer 在PODC 2000特邀报告中提出

由：Seth Gilbert, Nancy Lynch 正式证明

Seth Gilbert, Nancy Lynch, Brewer's Conjecture and the Feasibility of Consistent, Available, Partition-Tolerant Web Services，ACM SIGACT News 33(2), June 2002 pp 51–59.

Consistency Models

- Data-centric consistency models
  - Continuous consistency (on data content)
  - Update order consistency (on data operations)
    - Sequential consistency
    - Causal consistency
    - Grouping operations
- Client-centric consistency models
  - Monotonic reads
  - Monotonic writes
  - Read your writes
  - Writes follow reads

Continuous Consistency

- Defining the degree of “inconsistency”
- Deviation in numerical values between replicas
  - 如：股票价格，可以规定差值不超过0.01元，或0.5%
- Deviation in staleness between replicas
  - 如：天气预报数据，可以规定差别不超过1小时
- Deviation with respect to the ordering of update operations
  - 本地更新但未达成全局一致的那些临时操作的数量
  - 如：可以规定允许3个或5个操作为临时性，有可能需要回滚重新执行

Consistency Unit

- Conit：the unit of in consistency maintenance.
- 可以用于定义一致性水平；Conit大小影响数据更新成本
- 如：3个值（g,p,d）设置为一个Conit
  - 顺序偏差：未被另一副本提交的操作数量
  - 数值偏差=（未接收到的更新次数，偏差权重）
    - 偏差权重= 已提交的值与未收到的操作产生的结果之间的最大差值

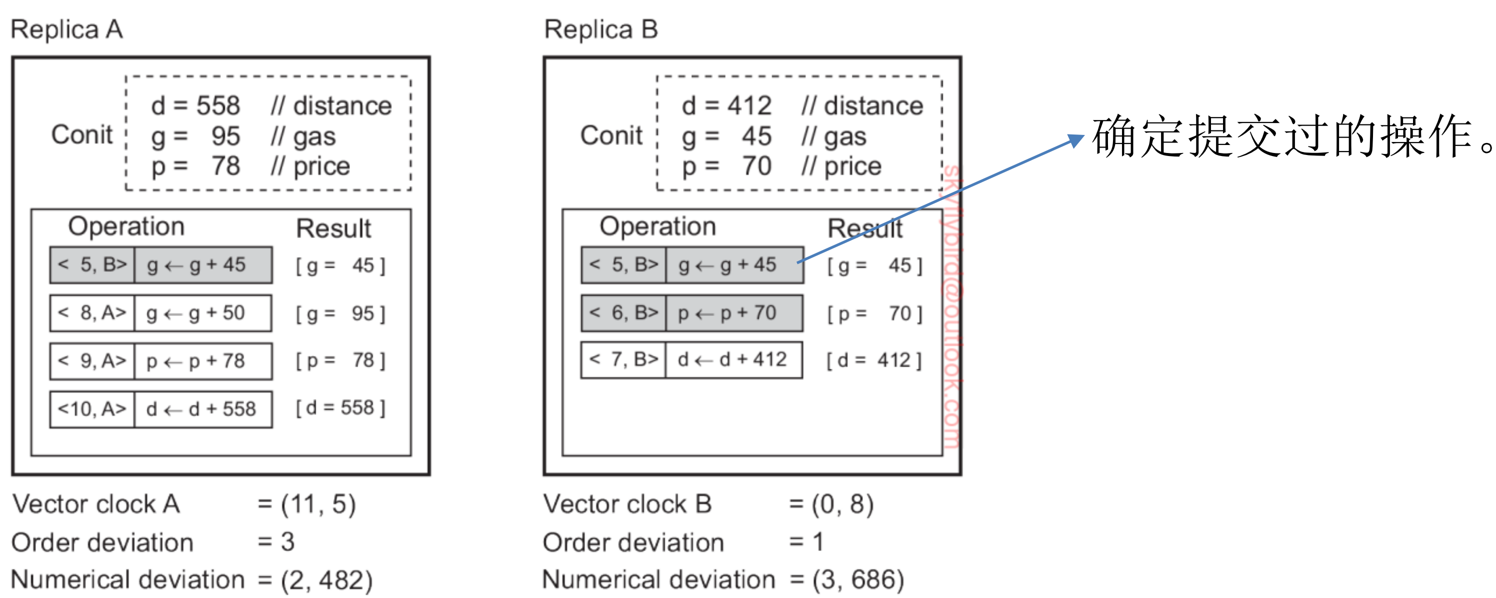


Figure 7.2: An example of keeping track of consistency deviations.

Sequential Consistency

- Notations

P1:	W(x)a		
P2:		R(x)NIL	R(x)a

- Definition: *the result of any execution is the same as if*
  - *the (read and write) operations by all processes on the data store were executed in same sequential order and*
  - *the operations of each individual process appear in this sequence in the order specified by its program*

Sequential Consistency

- No "time" in the definition of sequential consistency model
- A operation sequence is *valid* provided that *all processes see the same sequence.*

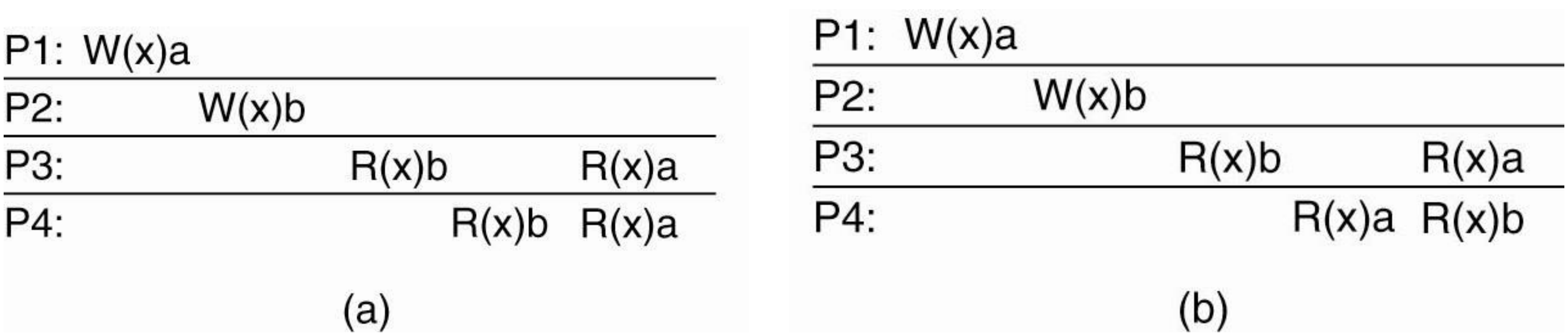


Figure 7-5. (a) A sequentially consistent data store.
(b) A NOT sequentially consistent data store.

Sequential Consistency

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

Figure 7-6. Three concurrently-executing processes.

x ← 1;	x ← 1;	y ← 1;	y ← 1;
print(y, z);	y ← 1;	z ← 1;	x ← 1;
y ← 1;	print(x, z);	print(x, y);	z ← 1;
print(x, z);	print(y, z);	print(x, z);	print(x, z);
z ← 1;	z ← 1;	x ← 1;	print(y, z);
print(x, y);	print(x, y);	print(y, z);	print(x, y);
Prints: 001011	Prints: 101011	Prints: 010111	Prints: 111111
Signature: 001011	Signature: 101011	Signature: 110101	Signature: 111111
(a)	(b)	(c)	(d)

Figure 7-7. Four valid execution sequences for the processes of Fig. 7-6. The vertical axis is time.

Totally 720 (6!) sequences; 90 of them are valid.

Causal Consistency

- Definition
  - *Writes that are potentially causally related must be seen by all processes in the same order.*
  - *Concurrent writes may be seen in a different order on different machines.*
- Recall "Causality"
  - If event *b* is *caused* or *influenced* by an earlier event *a*,
  - Everyone else should first see *a*, then see *b*.

Weaker than sequential consistency

Causal Consistency

P1:	W(x)a			W(x)c		
P2:		R(x)a	W(x)b			
P3:		R(x)a			R(x)c	R(x)b
P4:		R(x)a			R(x)b	R(x)c

Figure 7-8. A causally-consistent sequence but not sequentially consistent

Causal Consistency

P1:	W(x)a			
P2:	R(x)a	W(x)b		
P3:			R(x)b	R(x)a
P4:			R(x)a	R(x)b

(a)

P1:	W(x)a			
P2:		W(x)b		
P3:			R(x)b	R(x)a
P4:			R(x)a	R(x)b

(b)

Is the operation sequence causally consistent?

Grouping Operations

- Operation consistency with mutual exclusion mechanism

ENTER_CS, R(), W(), R(),…, LEAVE_CS

- The consistency granularity is higher
  - Critical section: a group of reads and writes.
- Operation:
  - Synchronization variables (locks)
  - Acquire → Read/Write → Release

Grouping Operations

- Acquiring a lock can succeed only when all updates to its associated shared data have completed.
  - 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。
- Exclusive access to a lock can succeed only if no other process has exclusive or nonexclusive access to that lock.
  - 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。
- Nonexclusive access to a lock is allowed only if any previous exclusive access has been completed, including updates on the lock’ s associated data.
  - 如果一个进程要以非互斥模式进入临界区，必须确保临界区获得了被保护共享数据的最新副本。

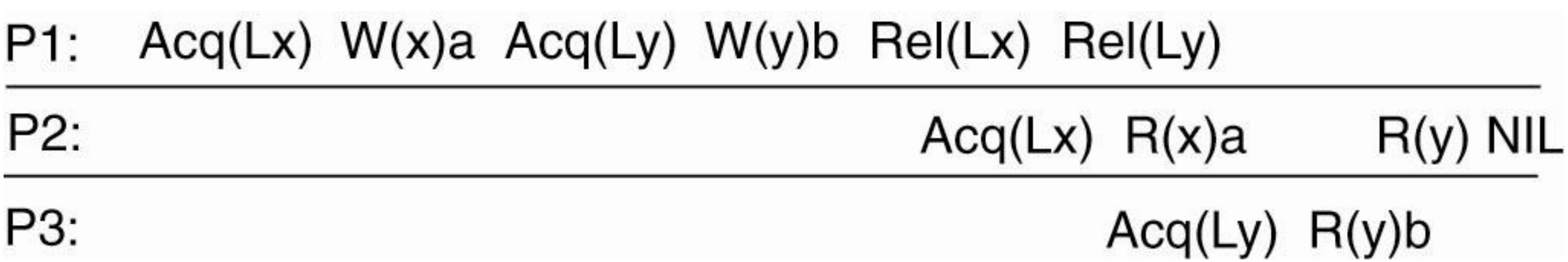


Figure 7-10. A valid event sequence for entry consistency

Client-centric Consistency Models

- Weaker than data-centric ones
- Assuming restricted concurrency
  - E.g.
    - A database may be rarely updated
      - Few write-write conflicts
    - A database may be updated by only a special process
      - No write-write conflicts
    - Users may allow a quite high degree of inconsistency
      - E.g. web pages
- Guarantee *eventual consistency*

Eventual Consistency

- Suitable for
  - (large-scale) distributed and replicated databases that tolerate a relatively high degree of inconsistency.
- Key point:
  - If no updates take place for a long time, all replicas will gradually become consistent.

  (In the absence of updates, all replicas converge toward identical copies of each other.)

Eventual consistency essentially requires only that: updates are guaranteed to propagate to all replicas.

Eventual Consistency

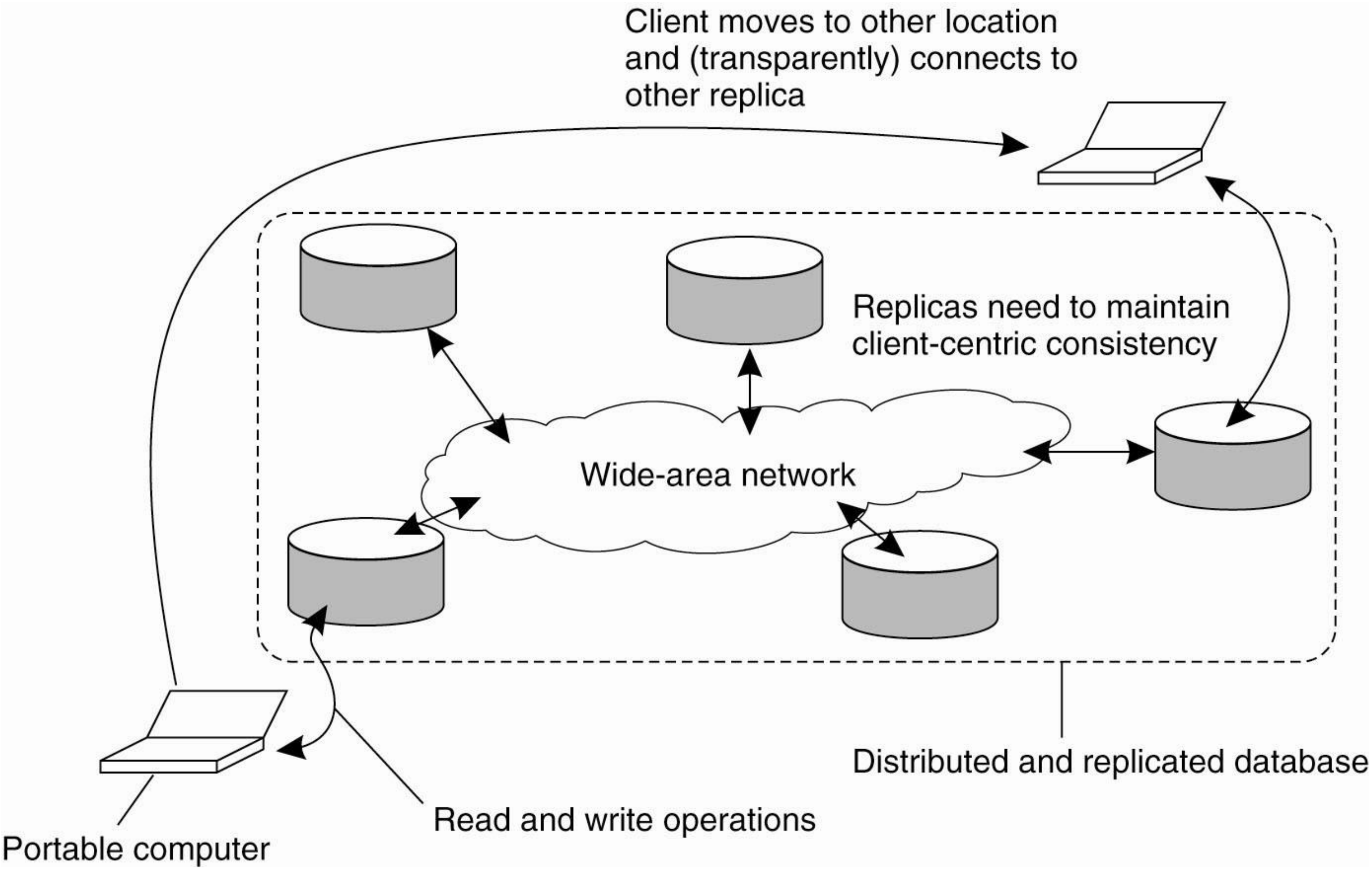


Figure 7-11. Inconsistency in eventually consistent database

Client-centric Consistency

- To avoid inconsistency in eventually consistent systems
- Key points:
  - Provides guarantees for a single client concerning the consistency of accesses to a data store by that client.
  - No guarantees are given concerning concurrent accesses by different clients.
- Four Models
  - Monotonic reads
  - Monotonic writes
  - Read your writes
  - Writes follow reads

Monotonic Reads 单调读

- If a process reads the value of a data item *x*, then any successive read operation on *x* by that process will always return that same value or a more recent value.

(if a process has seen a value of *x* at time *t*, it will never see an older version of *x* later.)

E.g. mail service

"读到值只会是越来越新"

Monotonic Reads

L1: $W_1(x_1)$ $R_1(x_1)$

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

(a)

L1: $W_1(x_1)$ $R_1(x_1)$

L2: $W_2(x_1|x_2)$ $R_1(x_2)$

(b)

L_i: 本地副本i

$W_i(x_n)$: 进程i对x写为版本n

$R_i(x_n)$: 进程i读到x版本n

$W_i(x_m;x_n)$: 在完成版本m之后，进程i写入x的版本n

$W_i(x_m|x_n)$: 进程i并发写入x的两个版本版本m和n

Fig. 7-16: (a): Guaranteed (b): Not.

Monotonic Writes 单调写

- A write operation by a process on a data item x is completed before any successive write operation on x by the same process.

(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。
If need be, the new write must wait for old ones to finish.)

E.g. software library

“自己提供的新版本不会被旧版本覆盖”

Monotonic Writes

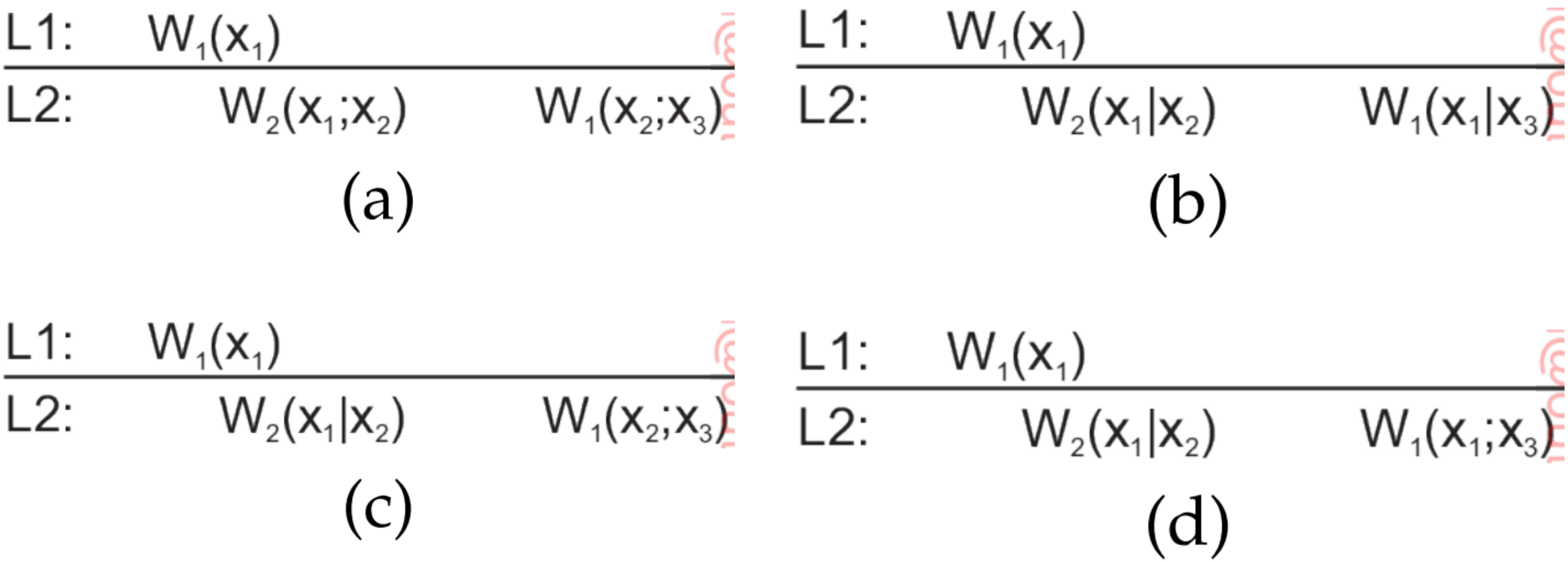


Fig. 7-17: (a) yes. (b) no. (c) no. (d) yes (although x1 has apparently overwritten x2).

Read Your Writes 读写一致性

- The effect of a write operation by a process on data item x will always be seen by a successive read operation on x by the same process.

(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。)

E.g. Password changing

“自己总是看到自己写过的最新版本”

Read Your Writes

L1: $W_1(x_1)$

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

(a)

L1: $W_1(x_1)$

L2: $W_2(x_1|x_2)$ $R_1(x_2)$

(b)

Fig. 7-18: (a): Guaranteed (b): Not.

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；

Writes Follow Reads写读一致性

- A write operation by a process on a data item x following a previous read operation on x by the same process is guaranteed to take place on the same or a more recent value of x that was read.

(进程对数据项 x 所执行的任何后续写操作都会在 x 的副本上执行，此时该副本已经具有该进程最近读取的或更新版本的值。)

E.g. bbs, newsgroup

“所见到过的必须已经出现”

Writes Follow Reads

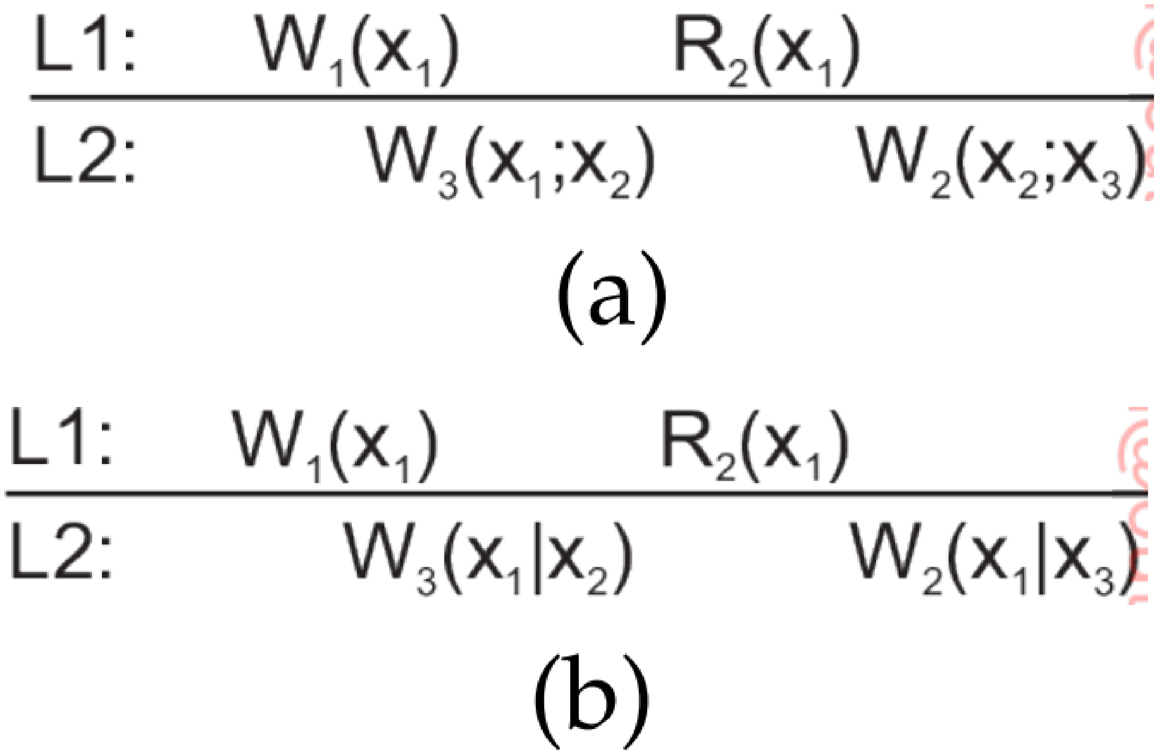


Fig. 7-19: (a): Guaranteed (b): Not.

§7.3 Consistency Protocols

- Data-centric consistency models
  - Continuous consistency (Data content consistency)
  - Update order consistency
    - Sequential consistency
    - Causal consistency
    - Grouping operations
- Client-centric consistency models
  - Monotonic reads
  - Monotonic writes
  - Read your writes
  - Writes follow reads

连续一致性：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:

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

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:

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

连续一致性：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

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

- 基本操作
  - S_i 扩散写操作 originating from S_j to S_k（写操作传播机制）
  - S_k会获得$TW[i,j]$，如发现更新操作步调不一致，把写操作从日志中转发给S_i
  - 转发操作可以把S_k的视图$TW_k[i,k]$往$TW[i,k]$靠近
  - 当应用程序提交一个新的写操作时，S_k会把其视图往$TW[k,k]$推，从而导致

    $TW[k,k]$- $TW_k[i,k] > \frac{\delta_i}{N-1}$
  - 但是本方法能够确保$TW[i,k]$- $TW_k[i,k] \leq \delta_i$

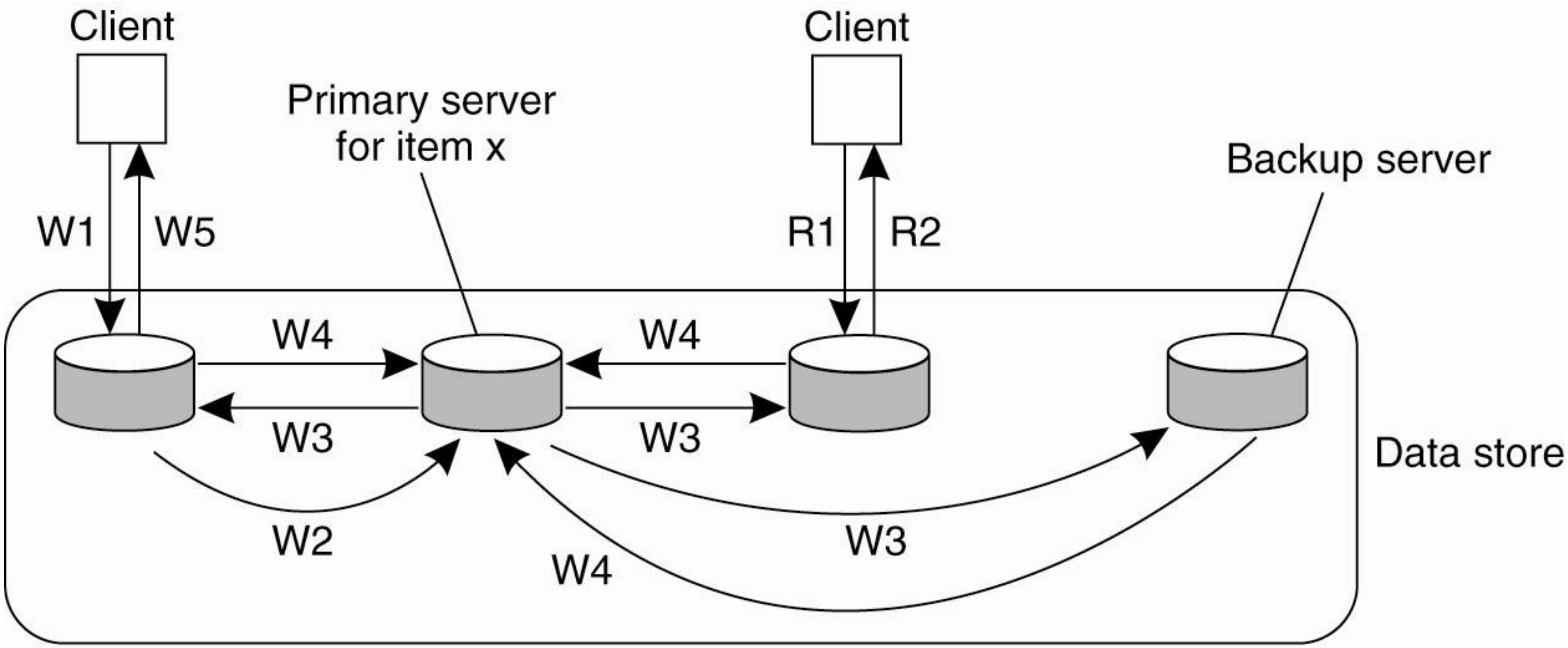
连续一致性：限定陈旧度、顺序偏差

- 限定陈旧度
  - 服务器 S_k 保持实时向量时钟 RVC_k
    - $RVC_k[i] = T(i)$ 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 发现 $T[k]-RVC_k[i]$ 将超出指定界限，那么就拉入来自S_i的时间戳晚于$RVC_k[i]$的写操作
- 限定顺序偏差
  - 暂存写操作到本地队列
  - 当本地写队列的长度超过限定时，不再接受任何新提交的写操作，按照相应的顺序提交写操作

操作一致性协议

- Primary-based Protocols
  - Remote-write
  - Local-write
- Replicated-write Protocols
  - Active replication
  - Quorum-based

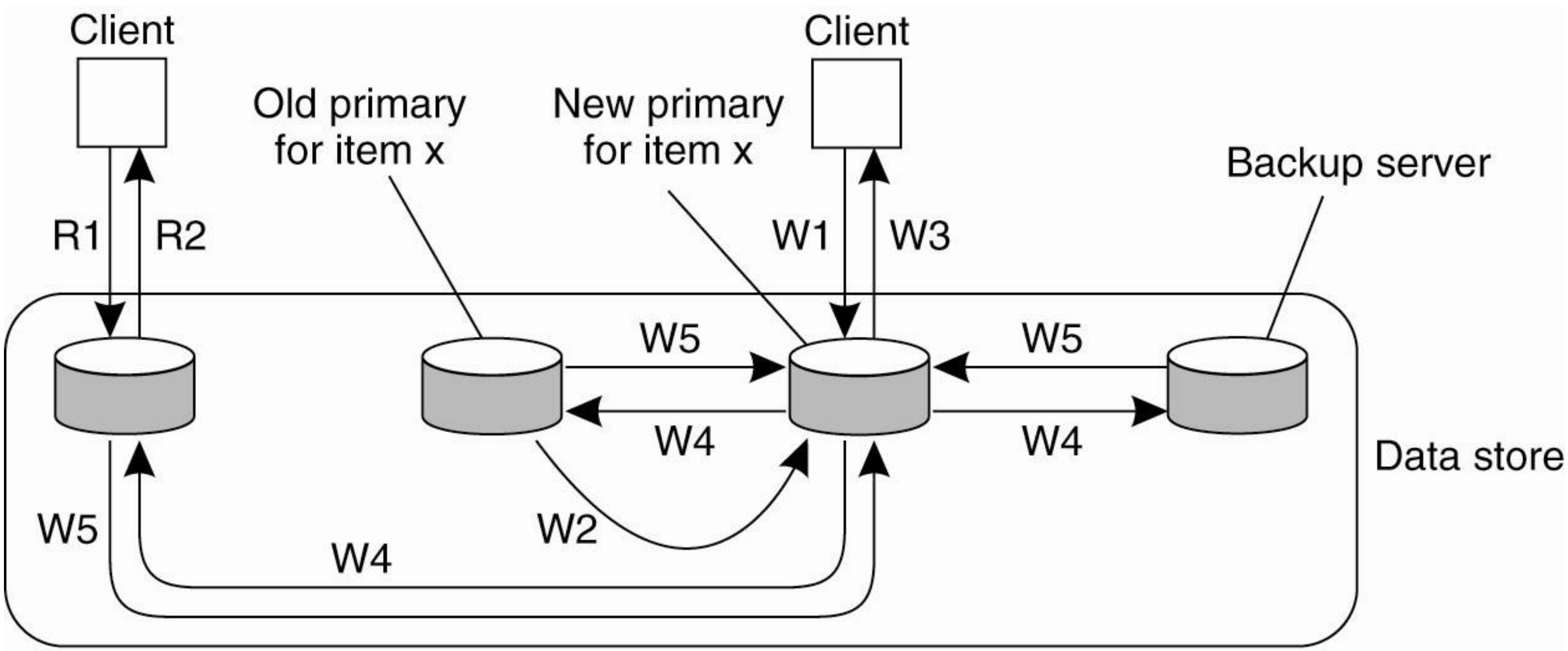
主副本协议：Remote-write Protocol



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

主副本协议：Local-write Protocol

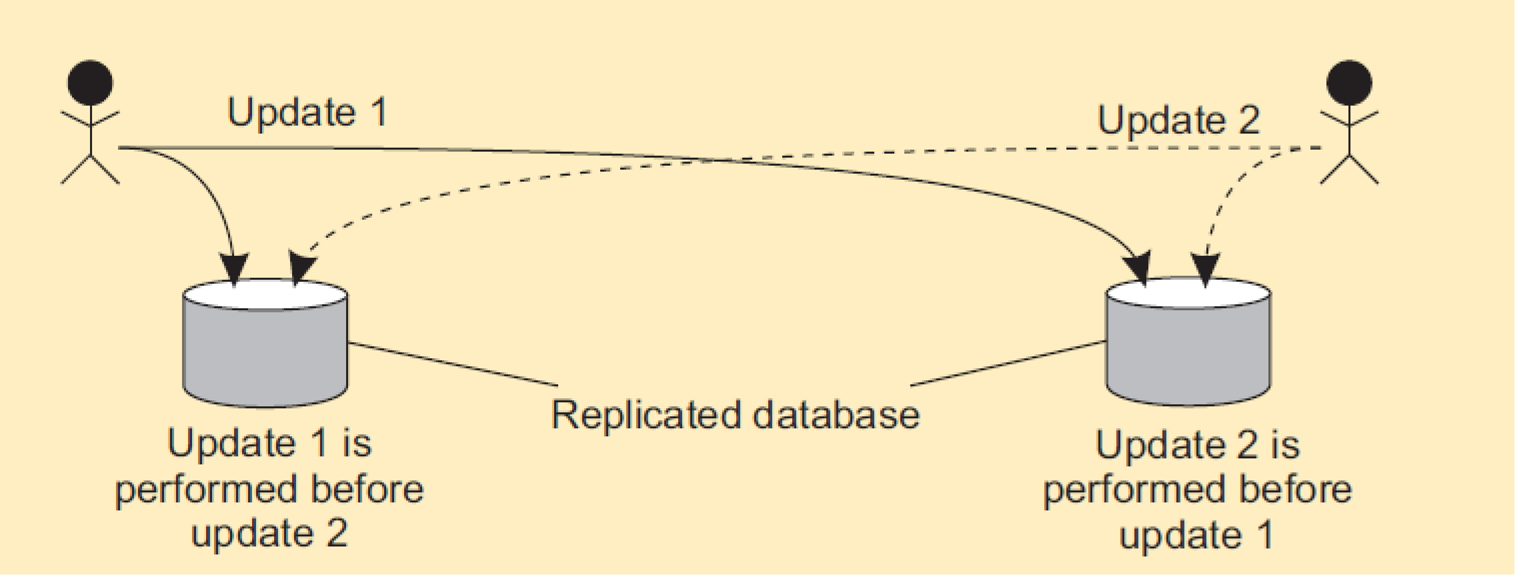


W1. Write request
W2. Move item x to new primary
W3. Acknowledge write completed
W4. Tell backups to update
W5. Acknowledge update

R1. Read request
R2. Response to read

复制写协议：Active Replication Protocol

- The operation is forwarded to all replicas
- Operations to be carried out in the same order everywhere
- Requires totally ordered multicasts using either Lamport timestamps (e.g.) or a central coordinator.



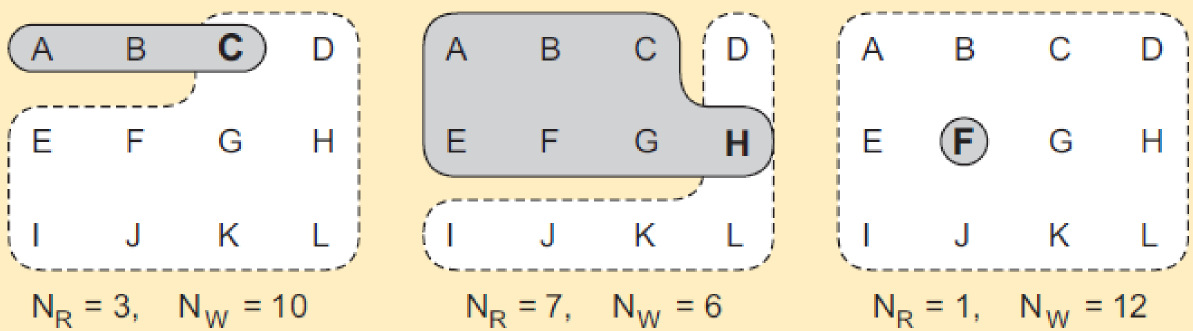
复制写协议：Quorum-Based Protocol

Quorum-based protocols

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)

1. $N_R + N_W > N$
2. $N_W > N/2$



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。

A Summary

- Replica management
  - 副本/缓存类型；副本放置、内容分发
- Consistency models
  - 数据为中心的一致性（读写并重系统）
    - 连续（持续）一致性、操作一致性
  - 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）
- Consistency protocols
  - 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

Homework Questions

1. 请分析讨论，与sequential consistency相比，eventual consistency的优势和价值，并通过例子进行说明。
2. 下面Causal consistency的操作例子，最后的两个读操作应该返回什么结果？

P1:	W(x)a			
P2:		R(x)a	W(y)b	
P3:			R(y)b	R(x)?
P4:			R(x)a	R(y)?

3. 给出一个实现数据副本的因果一致性的方法思路。