并行与分布式计算 - June 26’th, 2018

Data Processing Concepts

`https://github.com/rh01`

复制和一致性

Introduction to Replication

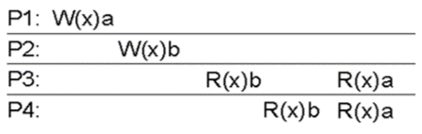
- Two Reasons for Replication:
  - Reliability & availability
    - keep working after one replica crashes, increasing system availability
    - protect against accessing corrupted data
  - Performance: reduce latency in data access and improve scalability in number and geographical area
- Price for Replication:
  - Costs of storage and communication
  - Replication → Consistency: globe synchronize or relax consistency constrains
    - Consistency Models and its implementation: Distribution and Consistency Protocols

Data-Centric Consistency Models

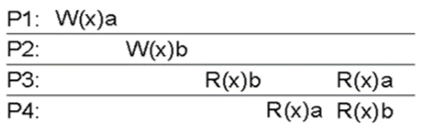
- A consistency model is a contract between processes and the data store which says if processes agree to obey certain rules, the store promises to work correctly. Normally, a process that performs a read operation on a data item expects the operation to return a value that shows the results of the last write operation on that data.
  - Strict consistency
  - Sequential consistency
  - Causal consistency
  - Entry consistency
  - Continuous consistency

Sequential Consistency

- 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 some sequential order and the operations of each individual process appear in this sequence in the order specified by its program.



(a)



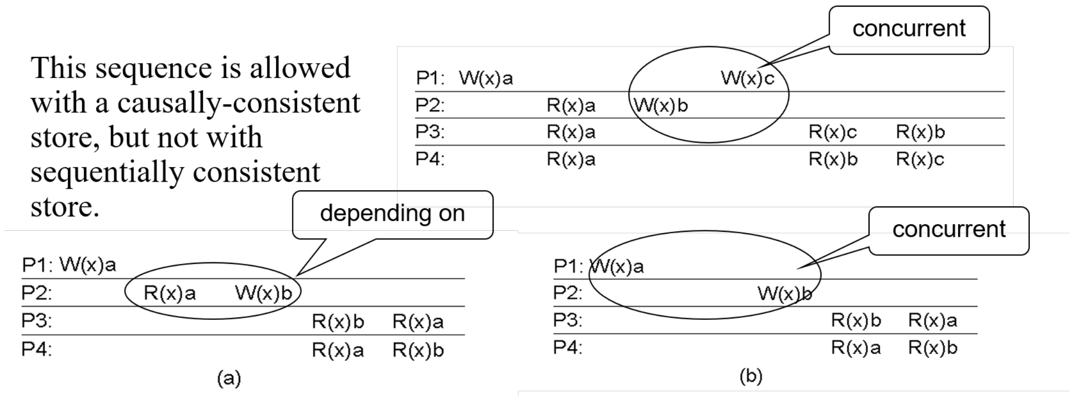
(b)

(a) A sequentially consistent data store.

(b) A data store that is not sequentially consistent. Where W(x)a stands for a write by process P to data item x with the value a and R(x)b stands for a read from data item x by P returning b.

Causal Consistency

- Writes that are potentially casually related must be seen by all processes in the same order.
- Concurrent writes may be seen in a different order on different machines.

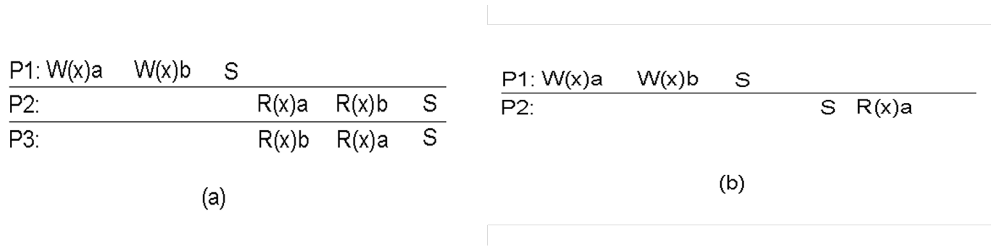


(a) A violation of a casually-consistent store.

(b) A correct sequence of events in a casually-consistent store

Consistency on a group of operations

- Used to guarantee that a group of operations are executed on the up-to-date local data.
- Synchronization variable S:
  - All local writes by process P are propagated to the other copies.
  - Writes by other processes are brought in to P’ s copy.
- Synchronization variable S limits only the time when consistency holds, rather than limiting the form of consistency.



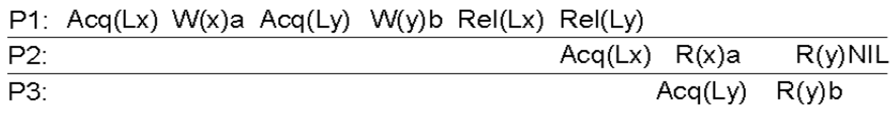
(a) A valid sequence of events for weak consistency.

(b) An invalid sequence for weak consistency.

Entry Consistency

- A group of operations are bracketed by the pair of operations Acquire and Release:

  - Acquire: enter a critical region
  - Release: exit a critical region
- A data store exhibits entry consistent, if it obeys the following rules:
  - An acquire access of a synchronization variable is not allowed to perform with respect to a process until all updates to the guarded shared data have been performed with respect to that process
  - Before an exclusive mode access to a synchronization variable by a process is allowed to perform with respect to that process, no other process may hold the synchronization variable, not even in nonexclusive mode.
  - After an exclusive mode access to a synchronization variable has been performed, any other process's next nonexclusive mode access to that synchronization variable may not be performed until it has performed with respect to that variable's owner.


P1: Acq(Lx) W(x)a Acq(Ly) W(y)b Rel(Lx) Rel(Ly)
P2: Acq(Lx) R(x)a R(y)NIL
P3: Acq(Ly) R(y)b

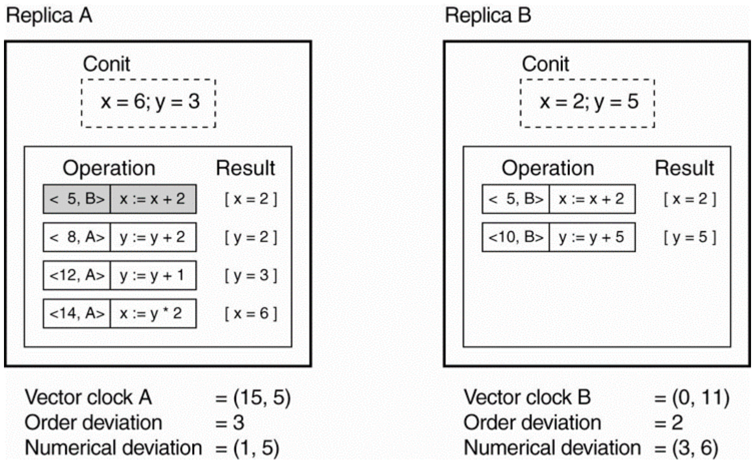
- A valid event sequence for entry consistency.
- The user cannot feel the existing of entry consistency while using of distributed shared objects.
  - client calls distributed objects -acquires synchronized objects -copies the newest object to client -releases lock after finishing method

Continuous Consistency

- Metrics for defining inconsistencies:
  - (numerical deviation) deviation in numerical values between replicas,
    * an absolute numerical deviation, a relative numerical deviation
    * in terms of the number of updates that have been applied to a given replica, but have not yet been seen by others. In a form of (value, weight) in the next page.
  - deviation in staleness between replicas,
  - (ordering deviation) deviation with respect to the ordering of update operations.
- Conit: a consistency unit, which specifies the unit over which consistency is to be measured.

Continuous Consistency-Example

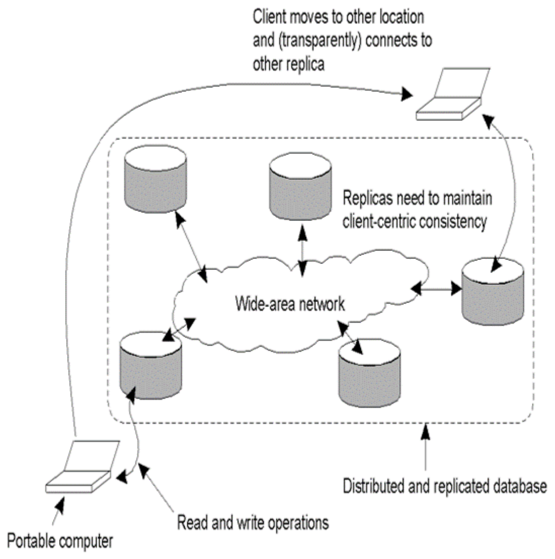
- Variables x and y are initialized to 0
- <t, i > expresses an operation that was carried out by replica i at its logical time t.



Replica A
Conit
x = 6; y = 3
Operation
Result
< 5, B> x := x + 2 [x = 2]
< 8, A> y := y + 2 [y = 2]
<12, A> y := y + 1 [y = 3]
<14, A> x := y * 2 [x = 6]
Vector clock A = (15, 5)
Order deviation = 3
Numerical deviation = (1, 5)
Replica B
Conit
x = 2; y = 5
Operation
Result
< 5, B> x := x + 2 [x = 2]
<10, B> y := y + 5 [y = 5]
Vector clock B = (0, 11)
Order deviation = 2
Numerical deviation = (3, 6)

Client-Centric Consistency Models

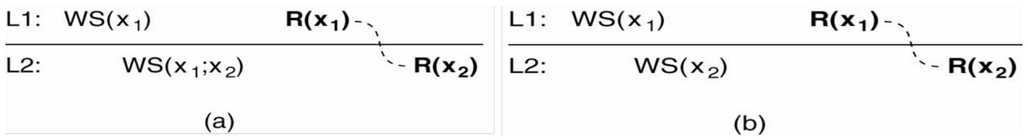
- Some data stores lack/have few simultaneous updates. The question needed to be solved is how fast the updates are made available to only-reading processes.
- Eventual Consistent data store
  - If no updates take place for a long time, all replicas will gradually become consistent.
  - Tolerate a relatively high degree of inconsistency.
- Client-centric consistency provides guarantees for a single client concerning the consistency of accesses to a data store by that client.
  - Monotonic-read consistency
  - Monotonic-write consistency
  - Read-your-writes consistency
  - Writes-follow-reads consistency



Client moves to other location and (transparently) connects to other replica
Replicas need to maintain client-centric consistency
Wide-area network
Distributed and replicated database
Read and write operations
Portable computer

Monotonic Read Consistency

- If a process reads the value of a data item x, any successive read operation on x by that process will always return that same value or a more recent value.
    - WS(xi) is the abbreviation of Write Series on data item x on site i.
    - If operations in WS(xi[t1]) have been performed at local copy Lj at a later time t2, we write WS(xi[t1];xj[t2]), abbreviated as WS(xi;xj)



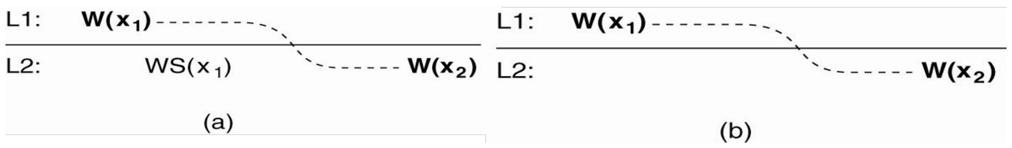
The read operations performed by a single process P at two different local copies of the same data store.

(a) A monotonic-read consistent data store

(b) A data store that does not provide monotonic reads.

Monotonic Writes Consistency

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



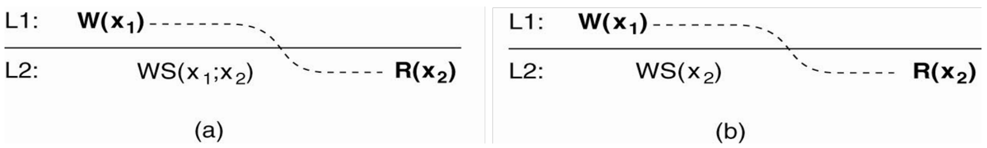
The write operations performed by a single process P at two different local copies of the same data store

(a) A monotonic-write consistent data store.

(b) A data store that does not provide monotonic-write consistency.

Read Your Writes Consistency

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

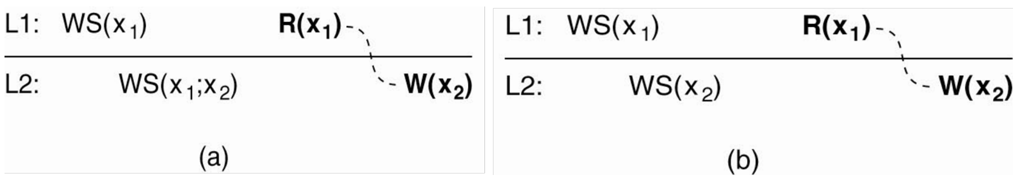


(a) A data store that provides read-your-writes consistency.

(b) A data store that does not.

Writes Follow Reads Consistency

- 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.



(a) A writes-follow-reads consistent data store

(b) A data store that does not provide writes-follow-reads consistency

Replica Management

- Replica-server placement: find the best locations to place a server that can host a data store
- Content replication and placement: find the best servers for placing content
  - Permanent Replicas: place in advance
  - Server-Initiated Replicas
  - Client-Initiated Replicas: client caches
- Content distribution
  - State versus Operations
  - Pull versus Push Protocols
  - Unicasting, Multicasting, or Epidemic Protocols

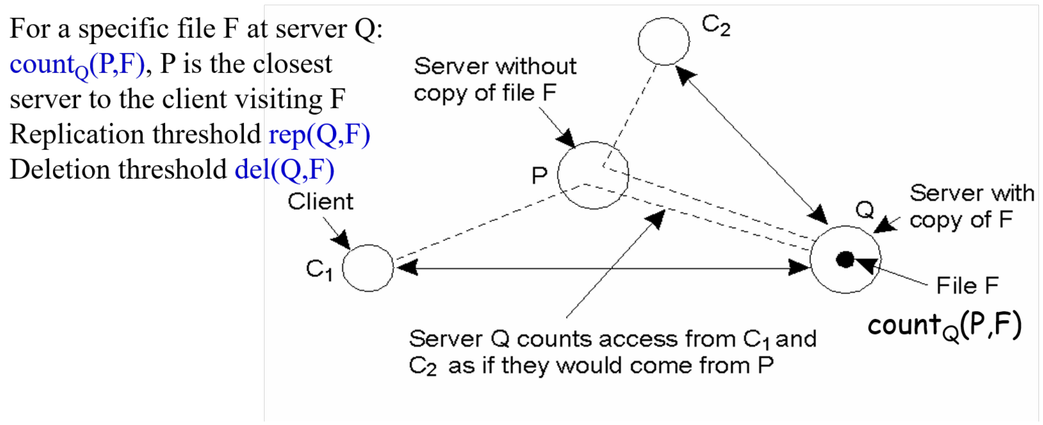
Replica-Server Placement

- An optimization problem in which the best K out of N locations need to be selected (K<N)
- Several heuristic algorithms:
  - (distance) Select one server at a time such that the average distance (in terms of latency or bandwidth) between the server and its clients is minimal
  - (topology) Place a server on the router with the largest number of network link in an autonomous system (AS)
    * An AS can be viewed as a network running the same routing protocol and managed by a single organization
  - (region) Select the most demanding regions, and let one of the nodes in such a region act as replica server
    * Region: a collection of nodes accessing the same content, but for which the inter-node latency is low
    * Identify the K largest clusters and assign a node from each cluster to host replicated content

Server-Initiated Replicas

- Target Environment: rarely-modified data store, e.g. in Web hosting services which replicate files to servers close to demanding clients

- How to migrate or replicate specific files on a server to servers in the proximity of demanding clients?


For a specific file F at server Q:
$count_Q(P,F)$, P is the closest server to the client visiting F
Replication threshold $rep(Q,F)$
Deletion threshold $del(Q,F)$

C_2
Server without copy of file F
P
Client
C_1
Q
Server with copy of F
File F
$count_Q(P,F)$
Server Q counts access from C_1 and C_2 as if they would come from P

圖 11：Counting access requests from different clients.

Client-Initiated Replicas

- Placement of client caches:
  - A cache is placed on the same machine as its client or a machine (in a cluster) shared by clients on the same local-area network
  - Place cache servers at specific points in a wide-area network, let a client locate the nearest server
  - Distributed cache system
- Metrics:
  - Invalidation time of cached data items
  - Cache hit

Content distribution: State versus Operations

- Propagate only a notification of an update
  - invalidation protocol: specify which part of the data store has been updated, and update the copy first whenever an operation on an invalidated copy is requested
- Transfer modified data from one copy to another, especially, to save bandwidth,
  - transfer only changes, or
  - pack multiple modifications into a single message
- Propagate the update operation to other copies
  - assuming each replica is capable of actively keeping its associated data up to date by performing operations

Content distribution: Pull versus Push Protocols

- Push-based: updates are propagated to other replicas without those replicas even asking for the updates.
- Pull-based: a server or client requests another server to send it any updates it has.

Issue	Push-based	Pull-based
State of 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

表 1: A comparison between push-based and pull-based protocols in the case of multiple-client, single-server systems.

Hybrid form of Pull and Push Protocols

- A lease is a promise by the server that it will push updates to the client for a specified time.
  - Server push updates within lease. When a lease expired, the client is forced to poll the server for updates, or requests a new lease for pushing updates
- Ways to dynamic adapt the expiration time:
  - Depending on the last time data was modified (Age-based leases)
  - Depending on the frequency a client requests its cached copy to be updated (Renewal-frequency leases)
  - Depending on the state-space overhead

Content distribution: Unicasting vs. Multicasting

- Unicasting: send N separate messages to N servers, one to each server, often combined with a pull-based approach to propagating updates
- Multicasting: send updating message with multicast supported by underlying network, often combined with a push-based approach
- Epidemic Protocols: implement update propagation in eventual consistent data stores

Consistency Protocols

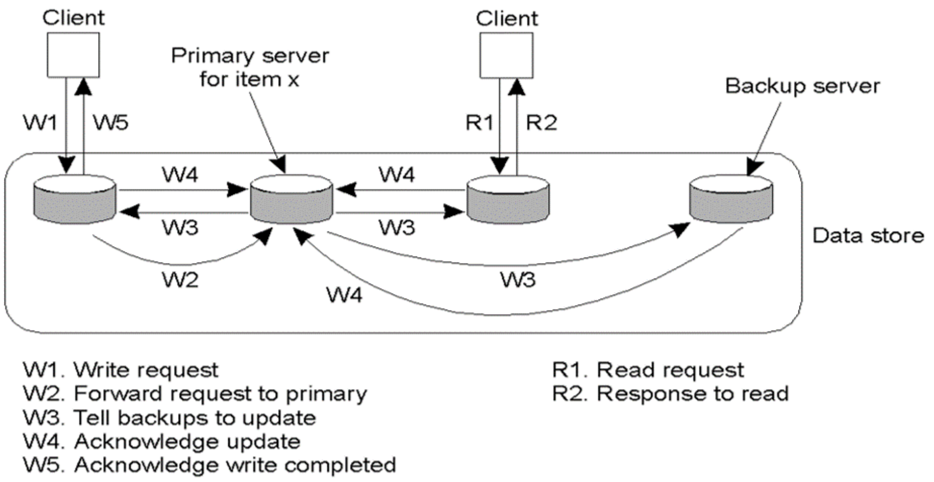
- Primary-based protocols: each data item x has an associated primary responsible for coordinating write operations on x.
- Replicated-write protocols: multiple replicas can carry write operations.
- Implement continuous consistency
- Implementing client-centric consistency
- Cache-coherence protocols

Primary-Based Protocols

- Primary-based protocols: each data item has an associated primary responsible for coordinating write operations.
- Remote-Write Protocols/Primary Backup Protocols: perform read operations on a locally available copy, but should forward write operations to a fixed primary copy
- Local-Write Protocols: moving the primary to the process where the write operation is initiated

Primary-Based Protocols: Primary-Backup Protocols

- The principle of primary-backup protocol which provides a straightforward implementation of sequential consistency.
  - Blocking version vs. Non-blocking version

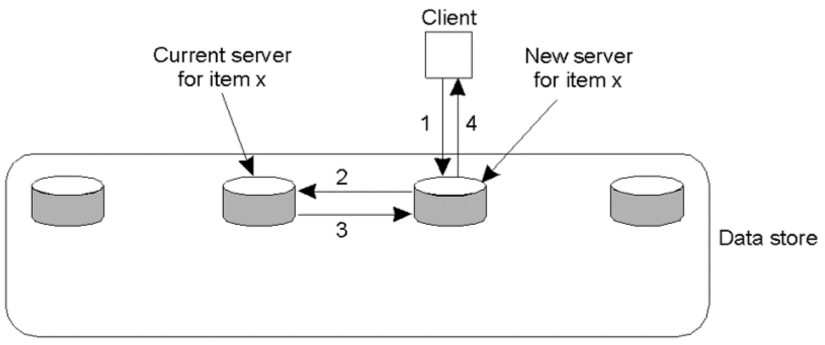


Primary-Backup System

- A primary-backup system implements sequence consistency if the primary is correct, since the primary sequences all the operations upon the shared objects. If the primary fails, then the system retains sequence consistency if a single backup becomes the new primary (by election) and if the new system configuration takes over exactly where the last left off
- Methods: the primary uses view-synchronous group communication to send the updates to the backups; the primary broadcasts heartbeat messages
  - view-synchronous group communication implements reliable group communication with dynamic groups in the presence of failures
  - Given at most m servers can fail over a given time and considering only crash failure, at least (m+1) replicas are sufficient to tolerate the crash of m servers, the smallest failover time is +2 +T, where is the interval between the reception of two consecutive heartbeat messages, T is the election time, and is the maximum message propagation delay from the primary to a backup server

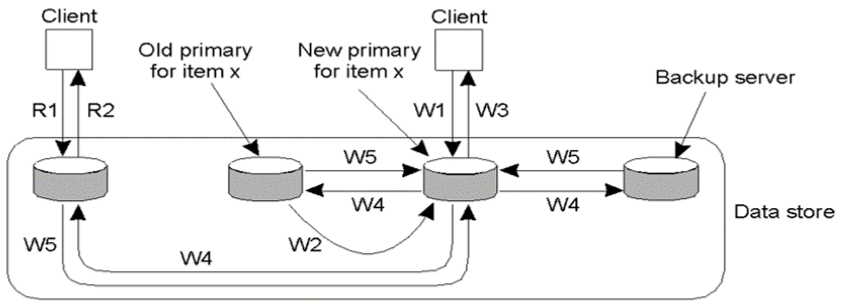
Primary-Based Protocols: Local-Write Protocols

- A kind of local-write protocol is the one in which a single copy is migrated between processes.
- How to locate the data with this fully migrating approach? Name service



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

- Another kind of local-write protocol is the one in which the primary migrates to the process that wish to perform a write operation.
- Non-blocking version for propagate updates to the replicas: a fixed central server may be adopted
- It is applicable to mobile computers working in a disconnected mode.



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

Replicated-Write Protocols

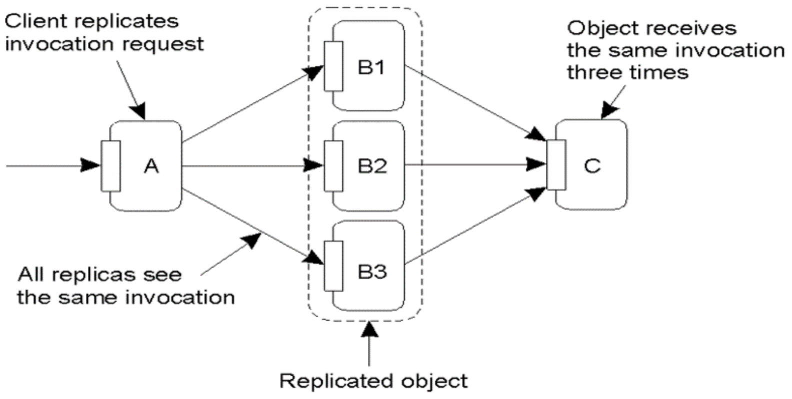
- Replicated-write protocols: multiple replicas can carry write operations.
  - Active Replication: forward an operation to all replicas
  - Quorum-Based Protocols: based on majority voting

Replicated-Write Protocols: Active Replication

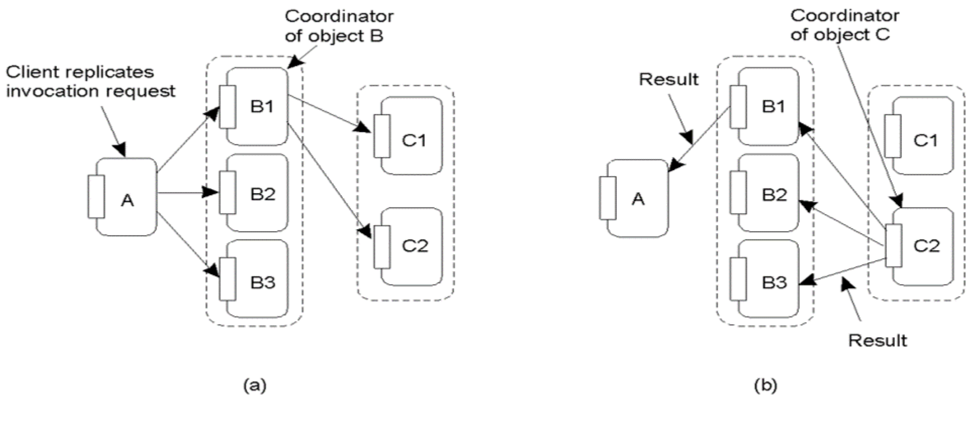
- Forwarding a write operation to multiple replicas.
- The problem of replicated invocations:
  - ensure the operations need to be carried out in sequence consistency everywhere
    - * Agreement: Every correct replica receives all the requests.
    - * Order (and Stability): Every correct replica receives the operation requests in the same order. (A replica next processes the stable request with the smallest unique identifier, where a request

is defined to be stable at replica server smi once no request from a correct client and bearing a lower unique identifier can be subsequently delivered to smi)

- send a request by the totally-ordered multicast primitive
- In presence of fail-stop of replica servers, send a request by totally-ordered reliable multicast primitive
  - The reliability of the multicast ensures that every correct replica server processes the same set of requests and the total order ensures that all correct replica servers receive the same sequence of requests and process them
  - The first response that the client receives is the desired value
- In presence of Byzantine failures of replicas, using OM(m) (m is the maximum member of faulty replicas) to reach agreement, using message timeout to deal with the absence of a message
  - At least (2m+1) responses of replicas will be required
- The problem of replicated invocations.
  - avoid replicated invocations


Client replicates invocation request
Object receives the same invocation three times
A
B1
B2
B3
C
All replicas see the same invocation
Replicated object

- How to deal with replicated invocation? Providing a replication-aware communication layer on top of which replicated objects execute.



(a) Forwarding an invocation request from a replicated object.

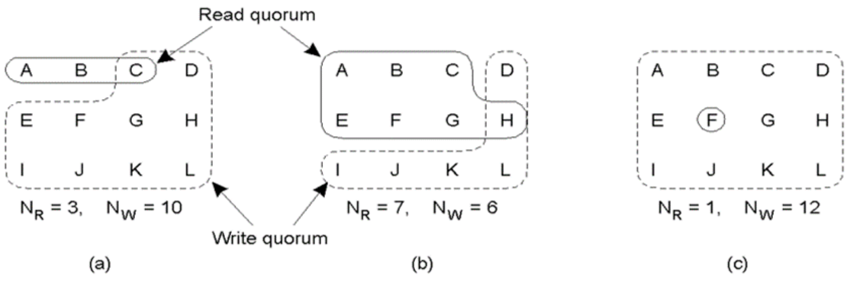
(b) Returning a reply to a replicated object.

Implementing fault-tolerant services via active replication

- Fault-tolerant state machine:
  - Each server is a state machine whose state is modified by a client request. If their initial states are identical and all updates are delivered in the same total order (in spite of failures), then all correct replicas will always be in the same state.
  - The clients multicast updates to all the replica servers
    - Both client machines and replicas may fail
    - Clients never fail, only the replicas exhibit faulty behavior.
  - If the agreement and the order requirements are satisfied, we can get a fault-tolerant service

Replicated-Write Protocols: Quorum-Based Protocols

- N replicas exist, reading a file needs a read quorum,an arbitrary collection of any NR servers, or more.
- Modifying a file require a write quorum of at least Nw servers.
- NR+NW>N avoid r-w conflict; NW>N/2 avoid w-w conflict



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)

Implement Continuous Consistency

Bounding Numerical Deviation

- Goal: for any time t, to let the current value vi at server Si deviate within bounds from the actual value v(t) of x.
- If v(0) is the initial value of x, then
- v(t) =v(0) + $\sum_{k=1..N}$TW [k, k] and
- vi=v(0) + $\sum_{k=1..N}$TW[i, k]
- N is the number of available replica servers
- We want to enforce v(t)-vi <= δi (δ is the upperbound)
- TW[i, j]= $\sum$ weight(W)| origin(W)=Sj & W$\in$ Li

  - TW[i, j] represents the aggregated writes executed by server Si that originated from server Sj.
  - TW[i, i] represents the aggregated writes submitted to Si

Bounding Staleness Deviations

- Let server Sk keep a real-time vector clock RVCk where RVCk[i] = T(i) means that Sk has seen all writes that have been submitted to Si up to time T(i), where T(i) denotes the time local to Si.
- Whenever server Sk notes that T(k) - RVCk[i] is about to exceed a specified limit, it starts pulling in writes that originated from Si with a timestamp later than RVCk[i].

Bounding Ordering Deviations

- Enforce a globally consistent ordering of tentative writes by primary-based or quorum-based protocols.

A Naïve Implementation of Client-Centric Consistency

- 客户保留两个写标识符集合，每个写操作的标识符由执行写的服务器给出：
  - 客户的读集合 Rset：由这个客户执行的读操作相关的写标识符组成
  - 客户的写集合 Wset：由这个客户执行的写操作的标识符组成
- (Monotonic-Read Consistency) 客户要在一个服务器上执行一个读操作
  - 服务器去检查客户的读集合，即，检查是否所有所标识的写操作已在本地执行。如果没有，那么，(*) 该服务器和其他服务器交互，确保该服务器在做读操作之前，将数据更新到最新；或者 (*) 将读操作传递到一个已发生了写操作的服务器上
  - 在读操作完成之后，那些与读操作有关的、在所选中服务器上执行的写操作的标识符，都被加入到客户的读集合中
- (Monotonic-Writes Consistency) 客户要在一个服务器上执行一个写操作
  - 服务器先检查客户传递来的写集合，确保所标识的写操作要先完成，然后执行这个写操作
  - 在完成写操作后，该操作的写标识符被加入到这个客户的写集合
- (Read Your Writes Consistency) 在服务器执行读操作之前，找到一个已执行完那些写操作的服务器，从它那里取得那些写操作的结果
- (Writes Follow Reads Consistency) 在服务器执行写操作前，让服务器先得到客户的读集合，把读集合中的写操作都执行一遍，然后执行写操作，然后把写操作的标识符加入写集合，同时，把读集合的标识符加入到写集合中

改进读集合和写集合的表示方法

- 缩短读集合、写集合的生命周期，把客户的读操作、写操作按照应用语义分成一个个会话，会话结束，读集合、写集合被清空
- 服务器 S_i：WVC_i 记录服务器 S_i 收到和处理的写操作的时间戳，其中 WVC_i[j] 是由服务器 S_j 发出的、由 S_i 接收和处理的最近的写操作的时间戳
- 客户：用向量时间戳表示读集合和写集合。为每个会话 A，构造向量时间戳 SVC_A，会话时间戳反映了客户看到的已经作为会话一部分执行的最新的写操作。也就是说，可以用一个时间戳表示所有来自同一个服

务器的客户看见的写操作

  - SVC_A[j] = max{timestamp(W) | W $\in$ A && origin(W)=S_j}
  - 如果 SVC_A [j] > WVC_i[j]，那么，S_i 还没有看到客户已经看到的源于 S_j 的写操作

- 用法：客户在会话 A 中登录服务器 S_i，它把 SVC_A 传递给 S_i。服务器 S_i 执行完写操作后，S_i 将更新它当前的时间戳 WVC_i。然后，客户的 SVC_A 将被更新：
- SVC_A[j] = max{SVC_A[j], WVC_i[j]}

Cache-Coherence Protocols

- Classify caching protocols according to
  - Coherence detection strategy: check at run-time
    - * In the case of distributed database, verify whether a cached data item is consistent with the version stored at the server while accessing the cached data item, or while committing a transaction
  - Coherence enforcement strategy
    - * No shared data at client caches
    - * Shared data permitted at client caches, the server sends invalidation to the cache or propagates the update
- What happens when a process modifies cached data.
  - Read-only cache: a pull-based approach
  - Write-through cache
  - Write-back cache