Data Replication and Consistency

Reference for study:

Van Steen, Tanenbaum, "Distributed Systems", chapter 7

Data Replication

- Data Replication is used to improve
  - Reliability
  - Performance
- We focus on **performance-related** data replication:
  - scaling out technique for size and geographical scalability
- Ideally, data replication should be hidden to the user (transparency), but data replication introduces a problem: **consistency**
  - when replicated data change, inconsistent states arise
    - replicas become temporarily not identical (breaking transparency)
    - cannot avoid, but can try to hide (paying a cost)

The Cost of Consistency

- Inconsistencies can be resolved by copying changed data to all replicas

  => bandwidth and computation overhead for making copies

  => additional latency introduced to hide inconsistent states

=> Replication can improve performance and scalability, **but** keeping replicas consistent has a performance cost

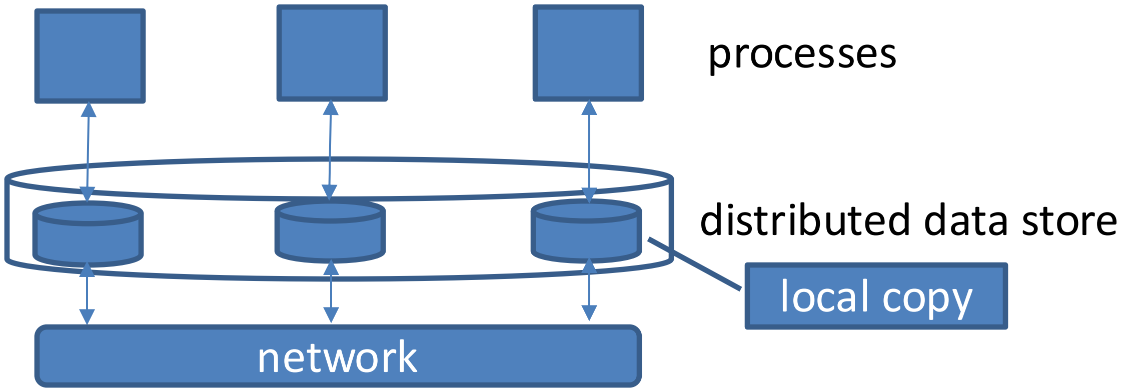
What is the balance?

Depends on the **consistency requirements** we have

=> trade consistency for performance

Consistency Models

- Consistency Model:
  - contract between processes and data store
  - in practice, between programmer and data store
  - "if programmers respect certain rules, then the data store guarantees certain consistency properties in read/write operations"
- Usually, this is the reference system model



Strict Consistency

- Each data change is propagated to all replicas before executing the next operation on the data store
- In practice,
  - each read returns the last written data
  - data changes cannot run concurrently with other operations
- Easy to understand and to use by programmers
- High Performance cost

=> To improve performance, consistency requirements must be relaxed

- the model becomes less easy to understand/use by programmers

Data-Centric Consistency Models

- Consistency properties are expressed in terms of how read/write operations behave on the global data store
  - what does a read return?
  - Strict Consistency:
    - each read returns the last written data
- Relaxing strict consistency means relaxing its constraints on read and write operations

Continuous Consistency

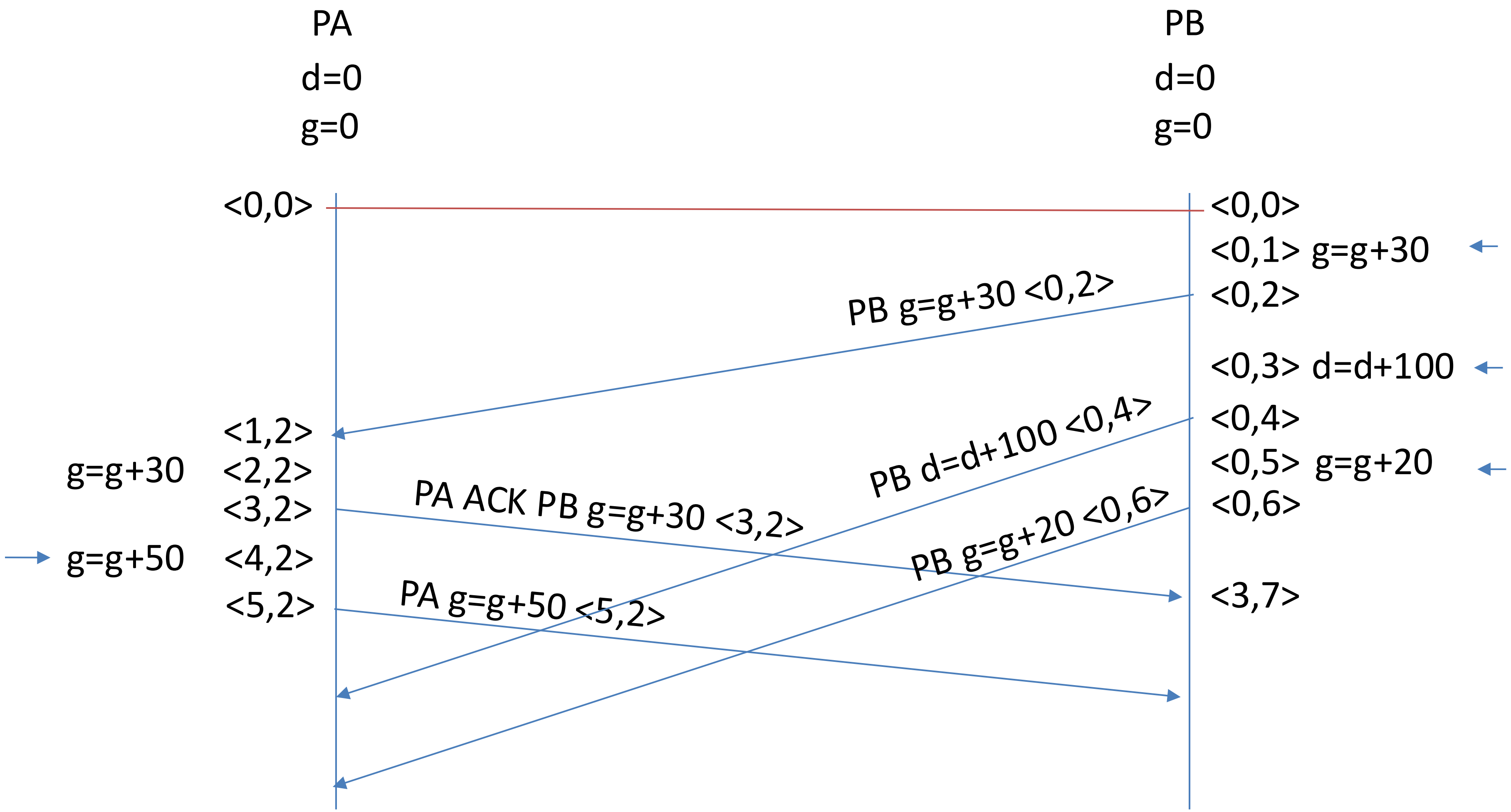
- Each read returns a value "close" to the last written one
- Consistency expressed/measured as
  - how much deviation (from strict consistency) is tolerated
- Consistency measures form a continuum along 3 axes:
  - numerical deviation (absolute/relative)
    - difference of numerical data between replicas
  - staleness deviation
    - difference of last update time between replicas
  - deviation in the ordering of write operations
    - in some cases, this deviation is temporary (operations applied on local copy but not yet confirmed by the other replicas)

    => maximum number of operations not yet made permanent

Consistency Units (Conits)

- Continuum consistency usually is measured referring to data units (conits) rather than to the whole data store
- Tradeoffs for conit granularity
  - large conits tend to lead soon to large inconsistencies
  - small conits lead to large number of conits to manage

Example
(see Van Steen, Tanenbaum)



Sequential Consistency (Lamport)

- "The result of any execution is the same as if
  1. the (read and write) operations by all processes on the data store were executed in some sequential order
  2. the operations of each individual process appear in this sequence in the order specified by its program."
- Any interleaving of read/write operations that satisfies 2. is possible (nondeterminism), but all processes see the *same* interleaving

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

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

Sequential Consistency (Lamport)

- Each read returns the same value that would be returned if all the operations of the system were executed sequentially in one of the possible sequential orders.

Causal Consistency

- Writes that may be causally related must be seen by all processes in the same order, respecting causality.
- Concurrent writes may be seen in a different order on different machines.

=> Weaker than sequential 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

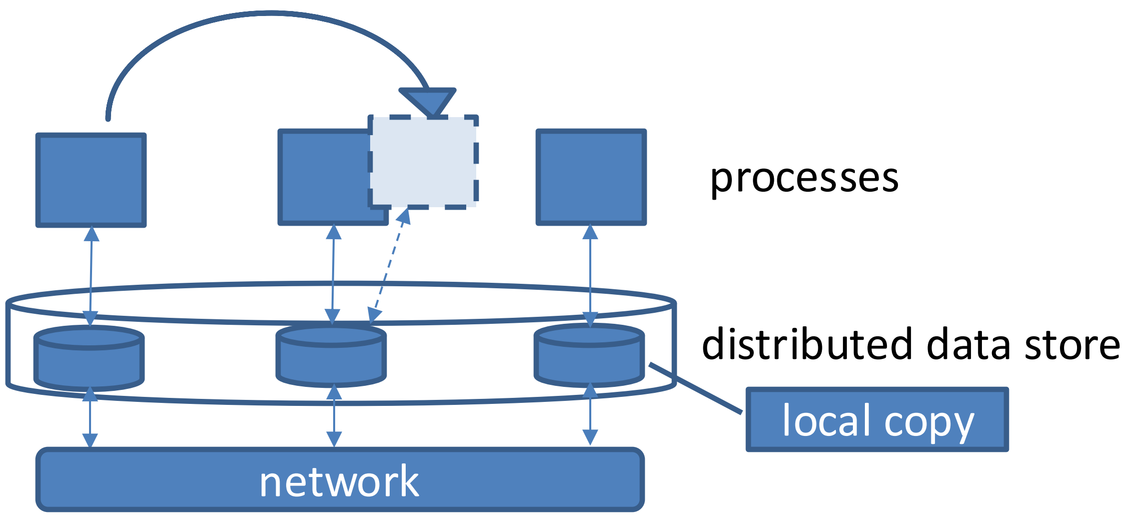
P1	W(x)a
P2	R(x)a W(x)b
P3	R(x)a R(x)b
P4	R(x)b R(x)a

Eventual Consistency

- In some applications, it is acceptable to have yet weaker consistency guarantees
  - Data stores that are written rarely by few processes but read frequently
    - slow propagation of updates (e.g., DNS, static Web pages)
- One possibility for these cases is **eventual consistency**:
  - each read returns the last written data or an older version
  - in the absence of write-write conflicts, if no updates take place for a long time, all replicas become identical (but how long it takes is not strictly specified)

Eventual Consistency

- Eventual consistency works well when there are few updates and when processes always access the same replica.
- Process mobility may cause issues to processes

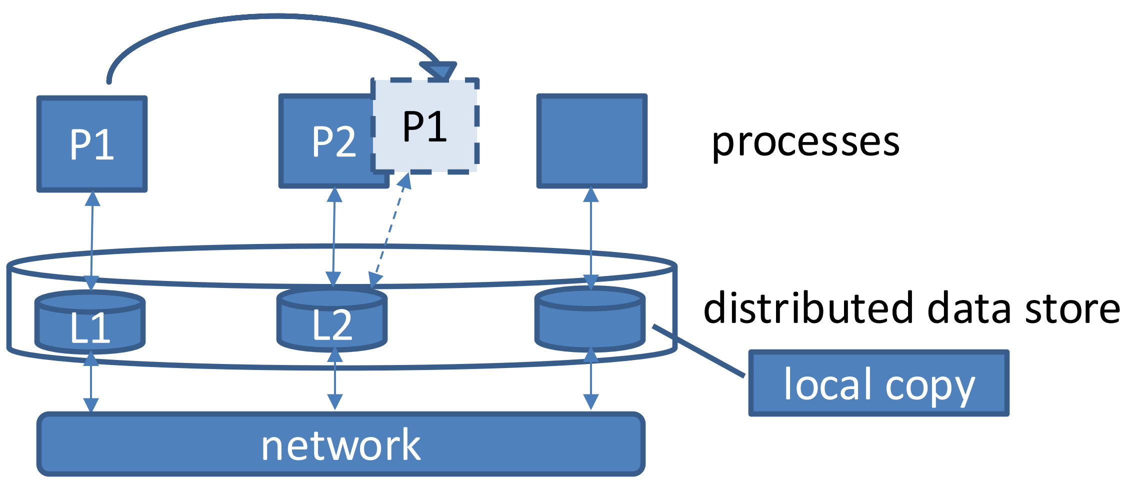


Client-Centric Consistency Models

- Data-centric consistency models provide system-wide data consistency properties
- Client-centric consistency models provide consistency properties for each single process (client)
  - These models can be added to limit the issues related to mobility with eventual consistency

Client-Centric Consistency Models with Eventual Consistency

xj	**version** j of variable x
Wk(xj)	Pk writes xj
Wk(x1;x2)	Pk writes x2 which follows from x1
Wk(x1|x2)	Pk writes x2 concurrently with x1 (potential write-write conflict)
Rk(xj)	Pk reads xj



Monotonic Reads

- "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"

L1	W1(x1)	R1(x1)
L2	W2(x1;x2)	R1(x2)

L1	W1(x1)	R1(x1)
L2	W2(x1|x2)	R1(x2)

Monotonic Writes

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

L1	W1(x1)		
L2		W2(x1;x2)	W1(x2;x3)

L1	W1(x1)		
L2		W2(x1|x2)	W1(x2;x3)

Read Your Writes

- "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"

L1	W1(x1)
L2	W2(x1;x2) R1(x2)

L1	W1(x1)
L2	W2(x1|x2) R1(x2)

Writes Follow Reads

- "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"

L1	W1(x1)		R2(x1)	
L2		W3(x1;x2)		W2(x2;x3)

L1	W1(x1)		R2(x1)	
L2		W3(x1|x2)		W2(x1|x3)

Replica Management

- Server Replication
  - what servers to replicate
  - placement problem: where to put servers
- Content Replication
  - what contents to replicate
  - placement problem: where to put replicas

performance

performance/reliability

Update Propagation Strategies

- Update propagation can take different forms:
  - **update notification** propagation (invalidation protocols)
    - best with low read/update
  - **data** propagation
    - best with high read/update
  - **update operation** propagation (active replication)
    - not always possible or convenient

Update Propagation Strategies

- Pull versus Push protocols
  - Push (Server-based)
    - generally used for permanent and server-initiated replicas
    - when strong consistency is required
    - best with high read/update
  - Pull (Client-based)
    - generally used for client-initiated replicas
    - when weak consistency is acceptable
    - best with low read/update
    - response time increases with cache misses

	Push	Pull
State to keep at server	list of client replicas+caches	none
Messages sent	update (+ fetch if invalidation)	poll (+ fetch if changed)
Response time at client	0 (or fetch time if invalidation)	poll (+fetch) time

Protocols for Continuous Consistency

- Processes perform writes on local copies tentatively
- Local writes at a process are propagated to the other processes
- Processes that receive updates detect/resolve any conflicts and apply updates to their local copy
- Processes monitor deviations of consistency metrics and make corrective actions
  - stop executing more writes on the local copy
  - force or request update propagation

Protocols for Sequential Consistency

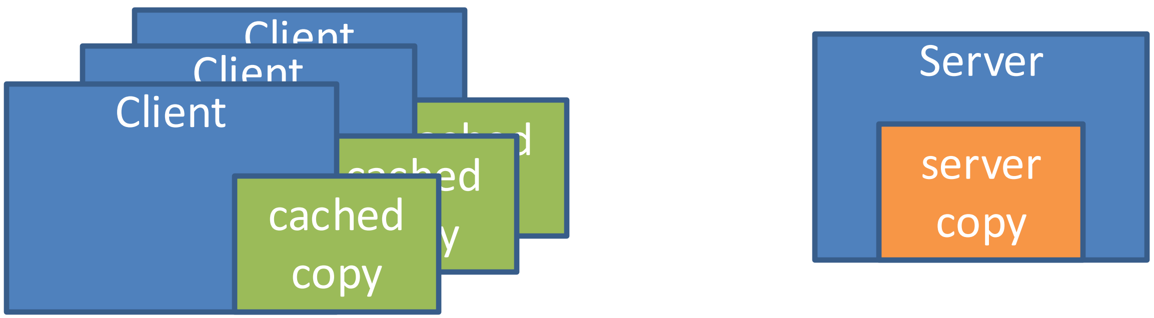
- Primary Based Protocols
  - for each data item x, a primary process is responsible for coordinating write operations on x
  - Remote Write (Primary backup) Protocols:
    - the primary for x is fixed
  - Local Write Protocols
    - the primary for x migrates to the process that needs to write (migration implies transferring the primary copy to the writer)
  - Examples:
    - **MongoDB**, **Amazon RDS**, **Azure SQL Database** (distributed databases, Primary backup)
    - **Hazelcast** (distributed memory, Local Write)

Protocols for Sequential Consistency

- Replicated Write Protocols
  - write operations can be initiated at different replicas
  - Active Replication Protocols
    - global ordering of operations can be kept consistent by performing **ordered multicast** or by using **centralized sequencers**
  - Quorum-based Protocols
    - Versions of data are recorded
    - Each process willing to write or read must start a voting procedure
    - Write and read require agreement with the majority of processes
    - Example: **Google Cloud Spanner** uses Paxos to reach consensus about writing

Cache Coherence Protocols

- Protocols specific for caches
- Typically used in client-server systems (like web apps)



Cache Coherence Protocols

- Coherence Strategy (**when** inconsistencies are detected)
  - client validates consistency **before** proceeding with transaction
  - client validates consistency **while** proceeding with transaction (optimistic approach) and aborts transaction if validation fails
  - client proceeds with transaction and validates consistency **at end** of transaction (if validation fails transaction is aborted)
- Enforcement Strategy (**how** consistency is enforced)
  - **Read-only** caches with push or pull mechanisms
  - **Read-write** caches:
    - protocols like primary-based local-write protocols (client requests lock on data and client's cache becomes a temporary primary)
    - concurrent writes are admitted (with a conflict resolution strategy)

Example: Cache Management in Web Applications