Summary of Database Replication: A Tutorial

By S. Xiao Fernández Marín

1 Summary

Why it is needed several copies of the data?

- **scalability:** things go faster, depending on how many clients
  r(requests,, easier using hash)
  w(update data,, harder bc conflicts)

- **fault tolerance:** high availability, if one server crashes, you can redirect to another server if some fails

- **decreasing response time:**
  geographic distribution), the latency is limited by speed of light so there is a long response time if the server is far (norway to NZ)
  off-line (disconnected) operation e.g: google docs: work offline and then when connecting to internet, update.

Terms:

- **Read-one-write-all-available (ROWAA)** NOT ALWAYS IMMEDIATELY
  - Reads executed on one node - enough only read from one node to get the value
  – Writes executed on all nodes - all the nodes have updates

- **Transaction location**
  - Primary - all the updates in that server or part of the db or all the db,, distribute the weight of primary
  - Update any were - clients update on any server

- **Synchronization strategy**
  - Eager: early on propagating updates to other servers,, to commit a transaction we have to replicate all the updates to all the servers first. *Coordinate early* and *Can guarantee strong consistency*
  - Lazy: not propagate before, we can do it later. *Don't coordinate* and *Weak consistency*: Stale, Temporarily inconsistent.

- **1-copy semantics:** single centralize system, we assume that we have one single server even if we have more that one server
  - Behave as if there was one centralized copy of the data: No distribution,, clients doesn't see the replication, they see an equivalent of a centralized system... we want the system to behave like this
  - Atomic operations: No in-between states;; no full transactions, just atomic

- **Two-phase commit (2PC):** 1 phase commit: one decide other do
  way of agree as a DS to abort or commit an atomic transaction and allow nodes to detect deadlocks without having to coordinate with other servers
  - Distributed agreement on whether to commit or abort
  - coordinator: not the same one for all the transactions but the same one foe the same transaction.. if there are more, behave as 1,, gather votes form all the participants
  - participators-¿ can decide to commit or abort- if someone says to abort, it aborts. to commit, everyone agrees to commit

Depending on the transaction location ans the synchronization strategy

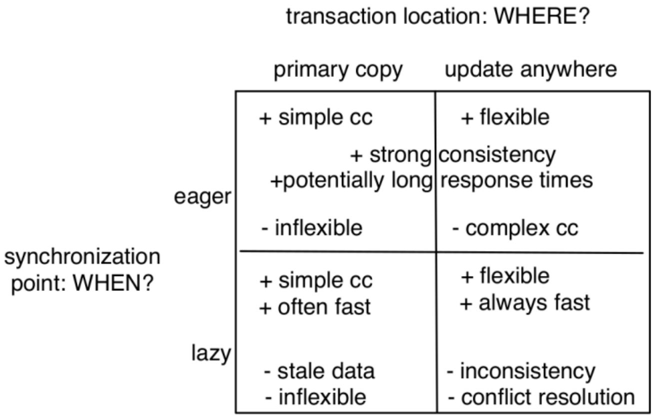

Figure 1: Categories

- **Eager primary copy** update every w,can update in central node
  • All read/write transactions on primary node
  - Local reads,, fast
  - Writes are distributed to all nodes,, can be bottleneck, we have to send them to the primary node

  • Read-only transactions on any node

  • Nearly the same as centralized system
  - Globally serializable
  - Applications must know if transactions are r/w or read-only

  • Updates are expensive

- **Eager update anywhere** update every w, can update in all nodes
  • All transactions can happen anywhere, no need to diferentiate between the transactions of r-only or r/w

  • Same semantics as centralized system
  - Globally serializable
  - Applications don't have to know it's distributed

  • Distributed deadlocks may happen bc update in any node and we can read from another

  • Both reads and updates are expensive but very flexible

- **Lazy primary copy** update later, can update in central node
  • All read/write transactions on primary node
  - Local reads
  - Local writes (sent to other nodes post commit) – someone reads old info bc central node didn't update the other ones

  • Read-only transactions on any node

  • Low overhead

  • Weaker consistency
  - Stale reads

  • Weaker durability in case of primary copy fail during commit

- **Lazy update anywhere** update later, can update in all nodes
  • All transactions can happen anywhere

  • Low overhead

  • Durability not guaranteed

  • Atomicity may be lost

- Weak consistency and isolation

  - Needs conflict resolution in case of concurrent updates; one node can do an update and another one other in the same bank account eg

Eventual consistency
• Temporary inconsistencies are allowed
- May even be the normal state

• In a steady state, consistency is achieved
- Left alone
- Without any updates
- Eventually

• Consistent values vs. consistent history
– Different histories can lead to the same values

Session consistency
• A client first issues read/write transaction T1, then read-only transaction T2

• Will T2 see T1's changes?
– Lazy replication may not have propagated fully

• Global transaction IDs
– Freshness requirements on read replicas

• Requires application/driver support

Other parameters

- **Cluster vs. WAN**
  • Cluster, everything close together
  - Low latency
  - High bandwidth

  • WAN, more far away
  - High latency
  - Low bandwidth
  - Geographic partitioning?

- **Statement vs. object replication**
  • Statement replication, mandas la funcion
  - E.g., SQL statement
  - Executed separately on each node
  - Efficient when statement is smaller than changed data
  - Semantically difficult, e.g., RAND()

  • Object replication, mandas el objeto
  - Executed once
  - Changed data is replicated
  - Efficient when changes are small

- **Concurrency control**
  • Pessimistic: **Locking**
  - leads to possible distributed deadlocks conflicts

  • Snapshot isolation
  - Only write/write conflicts

  • Optimistic: Timestamps
  - No deadlocks
  - Centralized or distributed validation

- **Architecture**
  - Kernel-based, replication in the core
  - Built-in replication
  - White box

  - Middleware
  - Black box, helps replication
  - Gray box, some db primitives replication with help of special APIs
  - Centralized or distributed middleware layer

- **Replication hierarchies**
  - Write vs. read
  - Replication doesn't give scalability if 100- 100% reads is infinitely scalable

  - Fan-out
  - How many secondaries to each primary?
  - Replication is not free

  - How many levels?
  - Lazy replication lag increases with each level