Replica Management

Data at Scale

Replication Problems

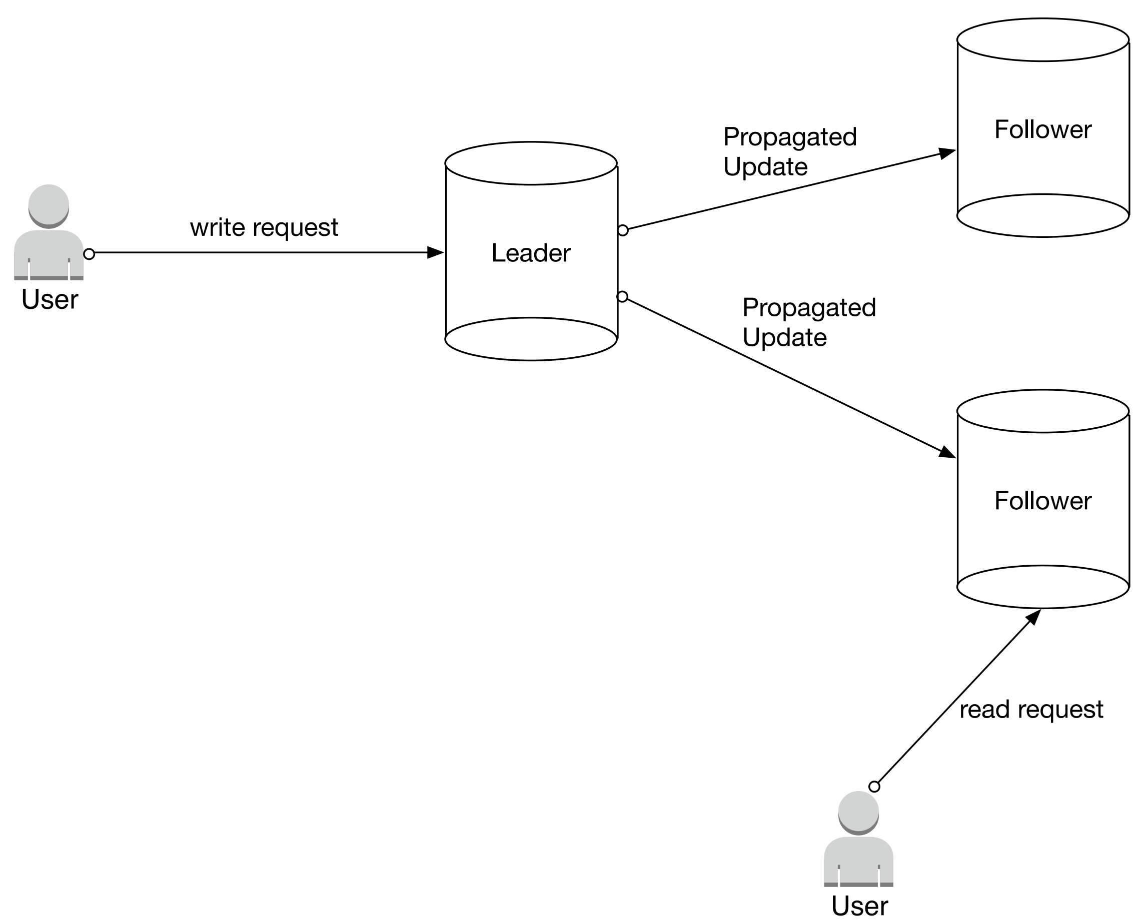
- Replication is done in order to
  - Keep data close to users (and thus reduce latency)
  - Failure tolerance (data is available when one replica is accessible)
  - Scale out the number of machines that can serve read queries

Sharding

- Sharding:
  - Divide data into shards and distribute them to different servers

Replica Management

- Need to insure that all replica are updated.
- Traditional method: Primary copy / leader / master-slave



Replica Management

- Can implement quasi-synchronous or asynchronous updates
  - Latter: a replica is updated later than the others

Replica Management

- Synchronous updates
    - Can use 2-phase or 3-phase commit
    - Absolute synchronous updates are not possible
    - Gets into problems with a failed follower

Replica Management

- Mixed synchronous and asynchronous updates
  - Only one follower is updated synchronously
  - Guarantees that updates are not lost when the leader fails

Replica Management

- Asynchronous updates:
  - Clients that read from different replica might get inconsistent data
- TASK
  - Give an example how serializability is violated

Replica Management

- Example:
  - T1: l(x)l(y)r(x)r(y)w(x)w(y)u(x)u(y)
  - T2: l(x)l(y)r(x)r(y)l(x)l(y)
- History at Site 1 that stores the preferred copy of x:
  - $l_1(x)l_1(y)r_1(x)w_1(x)w_1(y)u_1(x)u_1(y)l_2(x)r_2(x)u_2(x)$
- History at Site 2 that stores the preferred copy of y:
  - $l_2(y)r_2(y)u_2(y)l_1(x)l_1(y)w_1(x)w_1(y)u_1(x)u_1(y)$
- Locks need to be acquired and released globally

Replica Management

- Creating a new follower:
- Instead of locking the whole database
  - Step 1: Create a “snapshot” of the distributed system
    - Easy, because of leader
    - This induces leader to log all updates after the snapshot by creating a note to the
  - Step 2: Copy the snapshot to the new follower
  - Step 3: New follower obtains log of updates since snapshot
  - Step 4: Once the backlog is processed, follower moves to normal processing

Replica Management

- Dealing with failure
  - Follower failure
    - If a follower <u>knows</u> that it has failed
      - Synchronize logs with leader
  - Leader failure
    - Much more complicated

Leader Failure

- Dealing with leader failure:
  - Need to promote a follower
  - Reset writes
  - Inform others

Leader Failure

- Detecting Failure
  - Many sources of failure
  - Detecting failure:
    - Heart-beat monitoring
  - Electing a new leader
    - Example of a distributed consensus protocol
  - Reconfiguration to new leader

Leader Failure

- Problems:
    - Asynchronous replication:
        - Writes are still pending, old leader has not received all acknowledgments
            - Some solutions throw away updates that have not been performed by all
            - This violates durability of committed transactions

Leader Failure

- Problems:
    - Out-of-date data can cause problems if other services use the database
        - Lead to github unavailability for 2012

Leader Failure

- Problems:
  - With a network partition, we can have two leaders
  - If the timeout for failure detection is too fast, can have the re-election of a live leader
  - *Split Brain*
    - Leads to data corruption if writes are processed differently by the two leaders

Replication Management

- Statement based replication:
    - Forward all SQL ops to all followers
        - Difficult with non-deterministic functions such as NOW( ) or RAND( )
        - Auto-increment relies on the exact order of updates
        - Statements can have side-effects (triggers, stored procedures, user-defined functions) and need to have exactly the same at each node

Replication Management

- Replication based on Write-Ahead Log (WAL)
  - Log-structured storage engine
    - Log is the main place for storage
  - B-trees
    - Each modification is first written to the write-ahead log
- Log is an append-only structure
- Replicas can be based on exactly the same log
- Used in Posgres and Oracle

Replication Management

- Replication based on a replication log
  - Separates log for storage and for replication
  - Logical log contains the new rows

Replication Management

- Trigger based replication
  - Form of replication outside of the database system
    - Triggers: Automatically executed code upon change in the database
  - Trigger based replication:
    - Usually greater overheads than replication at the database level but more flexible

Replication Consistency

- Asynchronous writes create more consistency problems
  - Several models of consistency
  - READ YOUR OWN WRITES Consistency
  - aka READ-AFTER-WRITE Consistency
    - Avoids: User writes data, then reads from a different replica that has not yet updated
    - Example:
      - User reads her profile only from the leader
      - Every one else can get profile from any node

Replication Consistency

- Implementing Read your own write consistency
  - User reads from leader if the data could have been changed
  - User reads from leader if the data could have been changed by the user himself
  - Using timestamps

Replication Consistency

- CROSS-DEVICE READ YOUR OWN WRITE CONSISTENCY
  - User can use different devices to read and write

Replication Consistency

- MONOTONIC READ CONSISTENCY
  - Avoids:
    - User reads from one replica without large lag one value
    - User reads from another replica with large lag another value
    - Read old value before new value

Replication Consistency

- MONOTONIC READ CONSISTENCY
  - If a user reads different versions of the same value, then the versions are read in the order of write times
- Implementation
  - Make users read from a user-dependent replica

Replication Consistency

- CONSISTENT PREFIX READS
  - Avoids violation of causality
    - Example: If a sequence of writes happens in a certain order, then they are read in this order

Multi-Leader Replication

- Can use multiple leaders
  - To allow more than one node to accept writes
  - Multi-databases
  - Offline installations
- Each leader acts as a follower

Multi-Leader Replication

- Handling write conflicts
  - Single leader: Leader resolves order
  - Multi-leader: Avoid conflicts
    - Each user has a single, designated leader for updates from the user

Multi-Leader Replication

- Converging toward a consistent state
  - Solve conflicts by using Last Write Wins
    - Determine last write using
      - timestamps
      - leader to receive write tags write with its ID, then the write with highest tag wins
    - Merge writes
    - Record conflict in an explicit data structure

Multi-Leader Replication

- Converging towards a consistent state
  - Using custom conflict resolution
    - Triggered On write: If a conflict is detected
      - E.g. Bucardo (replicated PostgreSQL)
    - Triggered on read: Create multi-versions

Leader-less Replication

- Used by Dynamo, Riak, Cassandra, Voldemort
  - Known as “dynamo-style”
- Requests are sent to all replica
  - Need a write-quorum of replica to update
  - Need a read-quorum of replica to read

Leader-less Replication

- Convergence to a consistent state
  - Read Repair:
    - When a client reads inconsistent data from the replica, a “read repair” is triggered
  - Anti-entropy process:
    - Background process that checks for consistency

Leader-less Replication

- Quorums for *n* nodes
    - Set read quorum *r*
    - Set write quorum *w*
    - Works if $r + w > n$

Leader-less Replication

- Why would it not work if $w + r \leq n$?

Leader-less Replication

- Quora might need to be adjusted in case of node failures
- Can use “witnesses” to provide votes without the actual value
    - Witness stores a version number

Leader-less Replication

- Sloppy quora and hand-offs
  - Can use other than the designated nodes for the record if a node is unavailable
  - Can lead to inconsistency