Single Leader Replication

What is replication?

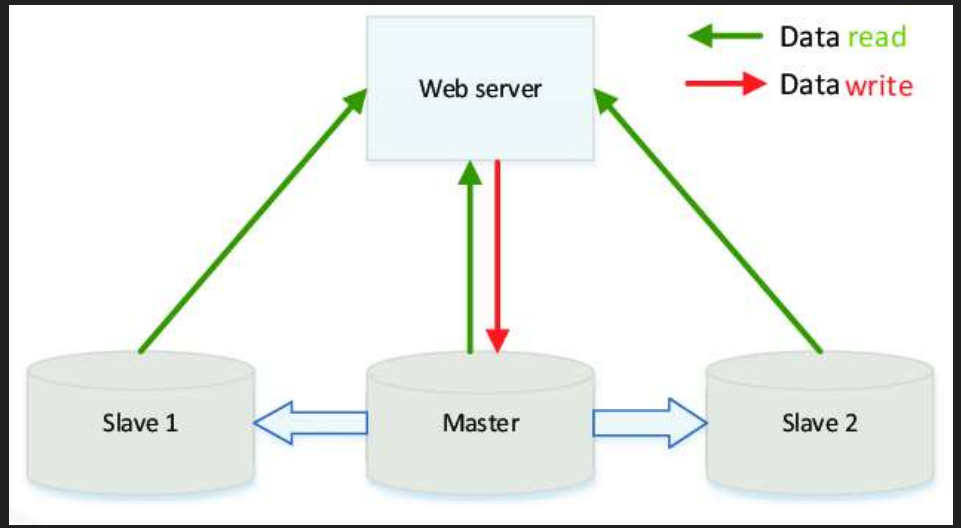
Having redundant copies of database so that application does not fail in the event of a crash, reduce load on each database serving requests

Types of replication

- Single leader
- Multi leader
- Leaderless

Single Leader Replication Overview

- All writes to one master database
- Master database sends list of writes to other databases via replication log
- Reads can come from any database



Increasing availability

- Adding a follower
- On follower crash
- On leader crash

Adding a follower

Step 1: Initialize the follower using a consistent snapshot of the leader database, which is associated with some position in the replication log

Step 2: The follower can now begin accepting changes from the leader

Dealing with a follower crash

At the time of crash, the follower knows where it was up to in the replication log.

Step 1: On reboot, fetch all of the new changes from leader node

Step 2: Start implementing the changes in the replication log from the index at which the follower node had previously failed.

Dealing with a leader crash (failover)

Step 1: Determine a new leader via some source of consensus (perhaps the most up to date replica)

Step 2: Configure all clients to send writes to new leader

Step 3: Configure all other followers to get changes from new leader

Problems with failover

- Some writes from the previous leader may have been propagated by only some or no replicas
  - Leads to either lost data or inconsistent replicas
- Accidental failovers due to network congestion can hurt our database performance even more
- If the old leader comes back, need to ensure that it does not think it is the leader and continue to accept writes (split brain)

Types of Single Leader Replication

- Synchronous
  - Client does not receive success message until all replicas complete the write
  - Strong consistency
- Asynchronous
  - Client receives success message the second that master completes the write
  - Eventual consistency

Tradeoffs of consistency types

- Strong consistency
  - Data always up to date, but writes take much longer
- Eventual consistency
  - Writes much faster, but clients can make stale reads to a replica

Dealing with eventual consistency

- Reading your own writes
  - Don’t see write after just making it
- Monotonic reads
  - Reads look like they are going back in time
  - Fix by having each user read from the same replica
- Consistent prefix reads
  - Causal relationship between writes lost because preceding write takes longer to replicate (due to being on different partitions)
  - Put causally related events on same partition or if not possible keep track of causal dependencies

Reading your own writes

Problem: You may write a change, read from a replica, and then not see the change

Solutions:

- Always read from leader for editable areas of application
- Always read from the leader or up to date replica for some period after a client write

Monotonic Reads

1:05 - Kate Upton to Jordan: “What’s up”

1:10 - Jordan to Kate Upton: “Not much playing video games”

1:15 - Kate Upton to Jordan: “Wanna come over?”

Monotonic Reads

1:05 - Kate Upton to Jordan: “What’s up”

1:10 - Jordan to Kate Upton: “Not much playing video games”

Monotonic Reads

1:05 - Kate Upton to Jordan: “What’s up”

Consistent Prefix Reads

REPLICA 1

6:22 - (Person A) Oh no the Knicks blew another game

6:25 - (Person B) Screw me!!

Consistent Prefix Reads

REPLICA 2

6:25 - (Person B) Screw me!!

Consistent Prefix Reads

REPLICA 2

6:25 - (Person B) Screw me!!

6:27 - (Person C) On my way!

Replication Log Implementation Options

- Copy over SQL statements
  - Bad because SQL statements are nondeterministic (for example current time)
- Use internal write ahead log
  - Not scalable if changing database engine, says which bytes were changed, other database may have different data in different locations on disk
- Use logical log
  - Describes which rows were modified and how, allows for more future proofing if the underlying database engine changes in the future

Conclusion - Single Leader Replication

Pros:

- All writes go through one machine which ensures consistency across replicas

Cons:

- Since all writes have to go through one machine write throughput may not be acceptable
  - Either need to partition the dataset and have a single leader per partition, or look towards other replication strategies