Replication

September 7th 2019

To improve performance, you can duplicate data. The two big reasons for performance are:

- Keeping data geographically close, reducing latency
- Scale # of machines that can serve requests, increasing throughput.
- Provide fault tolerance via the backups.

Assume for now that the dataset can be stored on one machine.

Synchronous replication means that changes will be guarenteed to propagate, but real distributed systems can't generally make this guarantee. So the async replication model guarantees neither, since nodes may be offline when changes are propagating.

Single-Leader Replication

One replica is designated as the leader, and all writes must go through it. Then, the write is propagated to following replicas (read-only replicas, from the clients perspective)

The leader sends it to the replicas usually via a log.

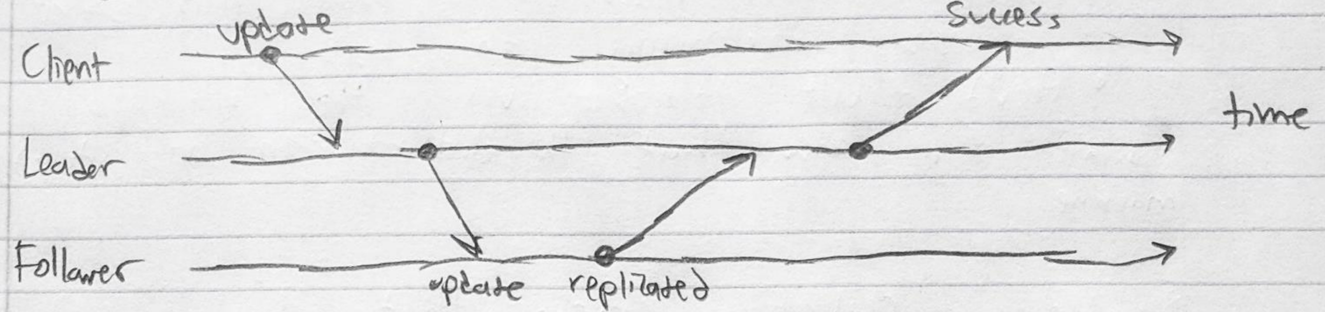
These servies use this strategy:

Relational DBs	Non-relational	Message Brokers
MySQL	MongoDB	Kafka
PostgresQL	Rethink DB	RabbitMQ.
Oracle Data Guard	Espresso	

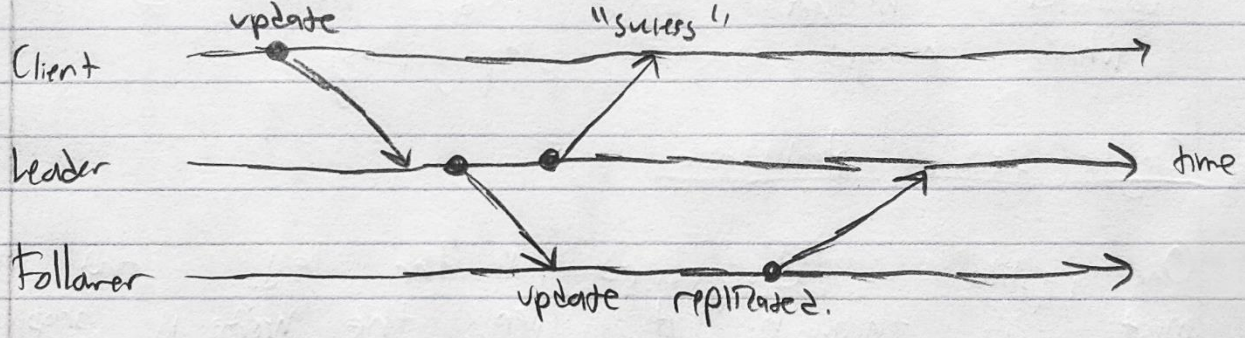
Synchronous vs. Asynchronous Replication

September 7th 2019

In synchronous systems, updates from the client need confirmation from the leader (and the followers) before being notified of success.



In asynchronous systems, there's no info about what the followers are doing from the client's perspective.



Updates are normally propogated pretty fast (≤1 second) but COULD take minutes if a follower is recovering from failure, or if the network is faulty, etc.

A node failure in a synchronous system causes the whole system to stall, so its impractical to have everything be synchronous.

In semisynchronous systems, SOME of the followers (though usually one, in practice) are synchronous, and the rest aren't.

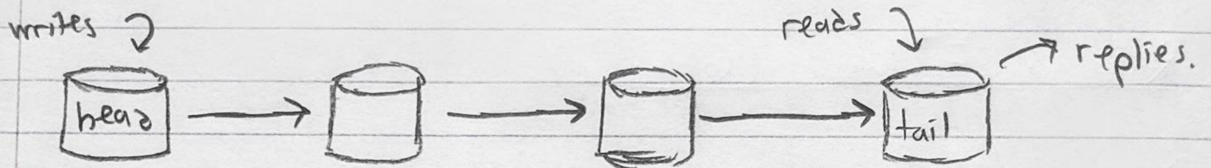
September 7th 2019

Asynchronous systems will:

- → lose updates if the leader dies, as the replicas never receive them
- → continue processing writes regardless of follower status.

Synchronous Replication: Chain Replication

The idea is to make the replicas into a "linked list":



The head failing means its successor becomes the new head.
The tail failing means its predecessor becomes the new tail

The internal nodes failing results in their removal, but the coordination required is complicated as not all messages from the failed node may have reached its successor. (See the paper for more details).

The histories of updates needs to be kept, so the failed node's successor can determine which ones it needs.

Adding a node results in it becoming the new tail, and the previous tail propagates everything to it.

A downside is that while this increases fault-tolerance, load is not scaled as only one server handles each type of request

This is used in some systems, like Microsoft Azure Storage. It is ideal for low-demand high-availability systems.

Recovery

September 7th 2019

Node recovery has a lot of similarity to how nodes would scale up, since both imply that some node is not up-to-date (either by being new or offline for a while).

Depending on the DB, this can be any level of automated

Recovery: Followers (Catch-up).

Since followers keep an update log, it can determine which updates it's missing by asking the leader.

If the node is brand new (by normal scaling up) or has been offline for way too long, it can do this:

1) Copy over a snapshot from the leader
2) Find all missing updates since the snapshot was taken.

The snapshots don't need to be fresh every time, backups can also work (and may already exist).

Recovery: Leaders (Failover)

The general process is as follows:

1) Detect leader failure: this can be done by any node
2) Elect new leader: usually want the new leader to be a more up-to-date one
3) Reconfigure routing: new writes need to go the new leader
4) Step down old leader:

August 12th 2019

Stuff that can go wrong:

- The new leader may be missing some updates, meaning writes from the old leader may be in limbo.

  Most commonly, those writes will be discarded, which can cause correctness errors.

  This problem is always present in asynchronous replication. A mitigating strategy would be to have some sychronous replicas (so semi-synchronous system) for backup.

- It may be possible for multiple leaders to exist at the same time (split brain). Without resolution, data can be lost or corrupted.

- Determining the timeout before re-electing leader is tricky:
  ↳ too short ⇒ unnecessary failovers
  ↳ too long ⇒ long recovery

  There may also be scenarios where the source of stress is only temporary:
  ↳ load spikes, causing responses to go above timeout
  ↳ network failure, causing message delays.

  These can trigger failovers that don't need to occur.

<u>Replication Logs</u>

When a leader makes a change, it will send the change to its replicas by a replication log.

August 12th 2019

Each replica maintains its own log of events it has processed.

Statement-Based Logs

Each event would be every write request (e.g. every SQL statement for a relational DB). Can be broken with:

- Nondeterministic functions like NOW(), which return different things depending on the replica.
- If statements depend on prior events (like UPDATE ... WHERE ...), then the events must be replayed sequentially.

  This is very limiting when multiple transactions are executing
- Anything where the execution has downstream effects that arent totally deterministic will break (like user-defined functions, stored procedures, etc.)

The edge cases are really tricky. and can be limiting overall, leading to many other log formats to be preferred.

Write-Ahead Logs

The WAL is an append-only sequence of byte blocks, recording all writes to the DB.

The leader sends the log to its followers, and you can rebuild the leader's state on them by replaying all the changes.

August 12th 2019

To allow crash recovery, the WAL is updated before modifying the main structure (like a B-Tree).

There are some disadvantages:

- Very low level, so it becomes dependant on the storage medium, due to details about disk blocks.
- Results in software updates that need downtime, as attempting to upgrade the followers followed by a failover for the leader is generally infeasible.

Used in PostgresQL and Oracle DB.

Logical Logs (Row-Based Logs)

An attempt to decouple the logical information and the data format, row-based logs keep a record of row-level changes to the table.

- For insertions, log contains new values of all columns.
- For deletions, log uniquely identifies the row that's been deleted (usually the primary key, otherwise all columns need to be individually logged).
- For updates, log uniquely identifies the row with its new values (of at least the values that were changed).

Any transaction that modifies several rows will generate multiple records.

The decoupling of the storage medium means:

- Easy backwards compatibility
- Leader & followers can run different versions
- Heterogenous storage.

Used by MySQL's binlog