Designing Data Intensive Applications

CHAPTER 5

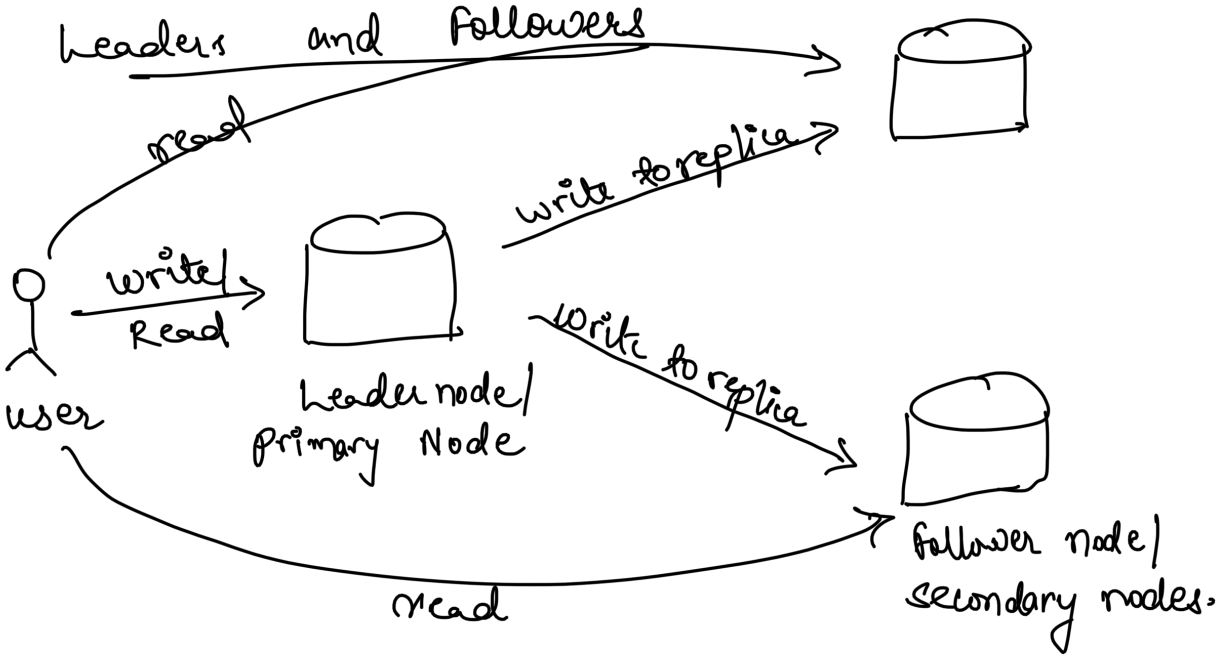
Replication

Keeping copy of data at multiple places.

why?

↳ keep data in same region as users → reduce latency.
↳ Availability → system is operational if certain machine goes down
↳ Can serve more Read TPS.

How to Replicate?



Synchronous vs Asynchronous Replication

① User is sent ack once write is completed on leader & followers	① User is sent ack once write is complete on leader node. Follower nodes are updated in background.
② Data is always consistent	② eventually consistent.

③ write can't be processed if any nodes goes down.

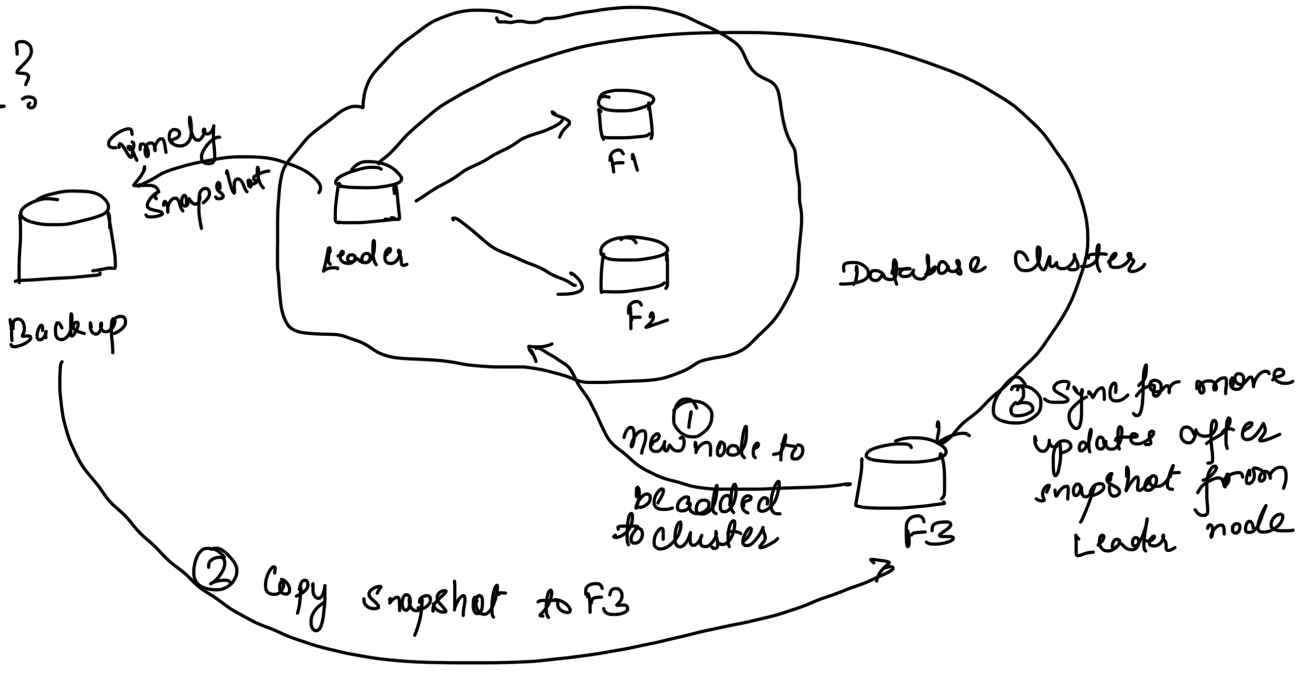
④ for write → leader node should be available.

Adding new follower

↳ in case more replicas are required to handle load.

↳ replace failed nodes in cluster.

How?



Handling Node Failures

① Follower Failure

↳ keep log of data changes on local disk. Once restarted, recover using log.

↳ sync with leader node for updates during downtime.

② Leader Failure

↳ A follower is promoted as new leader.

ⓐ Identify Leader has failed → health checks

ⓑ Choose new leader → best candidate could be node which has most recent replicated data.

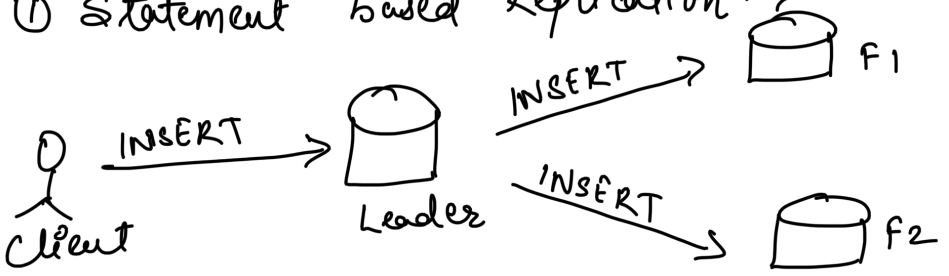
ⓒ System use new leader → new leader accepts writes from clients.

↳ If crashed node comes back, it is promoted to follower.

{ more Details on above in coming discussions }

Replication logs

① Statement based Replication →



→ statements received from clients are logged and forwarded to Clients.

Problems
1. Functions like NOW() or RAND() can store different value on nodes.
↳ If query depend on existing data, they must be executed in same order on each replica. → else can cause different effect.

② WAL – Write ahead log
append only file to maintain the operations on DB → discussed in CH-3

③ Logical (Row based) Replication

Use different log format for replication & storage engine.
↳ issue in #2 where WAL is used by replication & internal BTree.

① Row Insert → new values for all columns
② Row delete → primary key of row
③ Row Update → new values for columns that changed.

④ Trigger Based Replication
write custom code that is triggered for any operation in DB.

Replication Lag

↳ application can see outdated data in case of async replication.
↳ eventual consistency.

① Read After Write Consistency → "read your own writes"
The user who made an update, will see recent data on read.
This might not be the case for other users.
↳ ⓐ Decide when to read from leader and when from Followers.

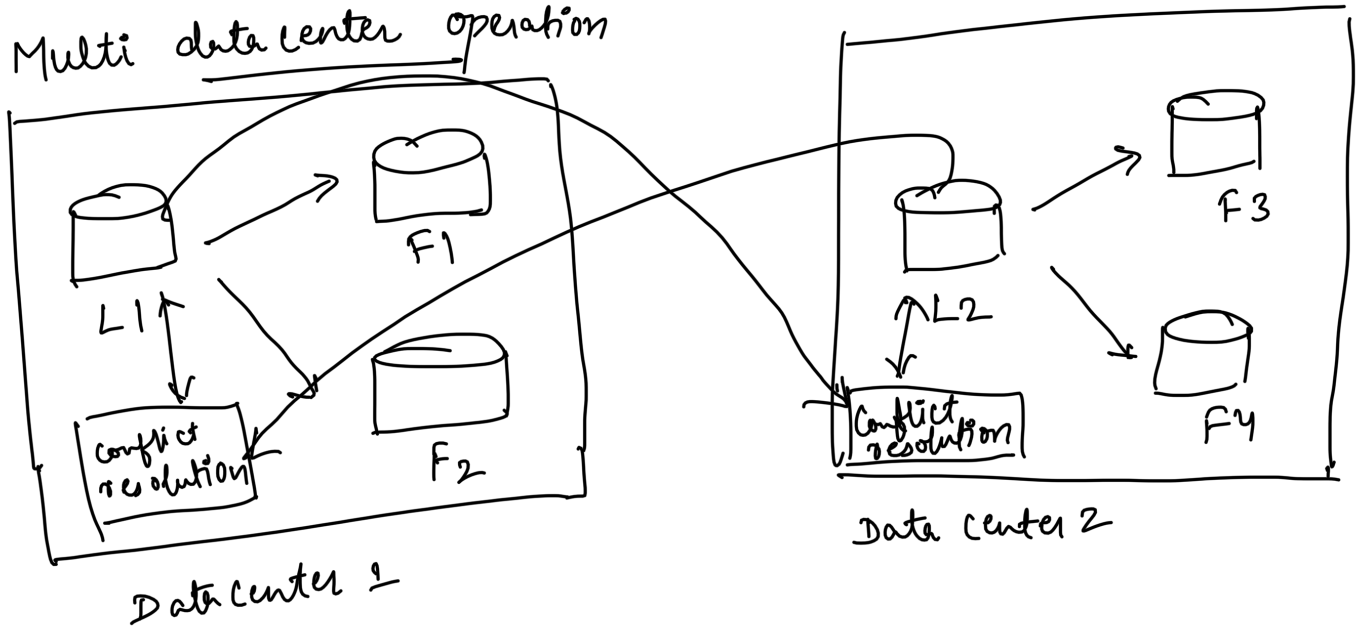
⑤ for 1minute after write, read only from leader.
what if same user reads info on multiple devices?
↳ can user requests routed to same datacenter to handle this.

② Monotonic Reads
users gets different data on subsequent reads.
↳ can same user always read from same replica.

③ Consistent Prefix Reads
If writes are performed in certain order, reads should happen in same order. → violation of casuality.
→ generally happens in partitioned databases.

Multi Leader Replication
↳ write can happen on any of leader nodes.
↳ Each Leader simultaneously acts as follower to other leaders.

① Multi data center operation



Benefits
① latency improvement - write can happen in specific data centers.
② Fault Tolerant → data center outage.
↳ network issues

Downsides
① same write can happen in both data centers concurrently.
↳ requires conflict resolution.

Examples

↳ Clients with offline operation — calendar application on multi-devices.

↳ collaborative editing - Google docs - conflict resolution for concurrent users.

Handling of write conflicts

① Conflict Avoidance → avoid conflicts to occur.

↳ request from particular user are always routed to same leader
↳ single reader in user's view

② converging to consistent state

↳ assign each write a unique ID. example Timestamp.
conflicts can be resolved basis "Last write wins".

↳ assign each replica unique ID.
writes from higher number replica takes precedence over other.

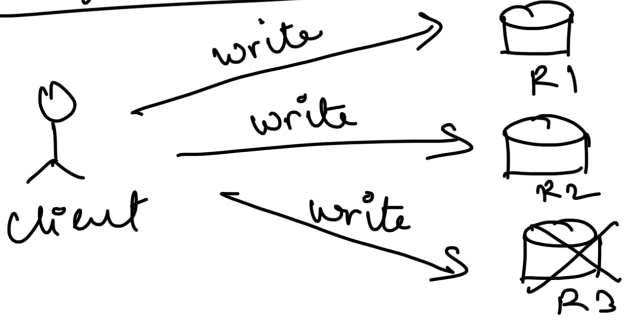
③ Custom Application Code

↳ write your custom conflict handler.

Leaderless Replication

↳ any replica can accept writes from clients.
ex - cassandra, voldemort

What if a node is down on write?



↳ only R1 R2 are available
↳ write is successful basis quorum
↳ on Read as well, request is sent to all nodes, and then recent value can be picked up basis version no.

How will be R3 in sync once it comes online?

① Read Repair → on read, we identified which Replica has stale data, so write back to R3.

② Anti-entropy process → background process to check differences between replicas.

Quorums

n - replicas
w - no of nodes to confirm on write for it to be success
r - no of nodes to be queried for read.

$w + r > n$

↑ we'll get up-to date reads.

General recommendation

n → odd number
w, r → $(n+1)/2$

① If you set lower values of r and w (r or $w <= n$) → possibility of stale reads.

② even with ($w + r > n$), there is possibility of stale reads.

ⓐ Sloppy quorum →
↳ In case of network disruptions
↳ will you return errors?
↳ OR return response (maybe stale data) without reaching quorum.
↳ Sloppy quorum

reads and writes still require r and w successful responses but the response might not be from designated node.

↳ return temporary respone and once node is back, restore it to correct state.

hinted handoff.

ⓑ Two concurrent writes →

Multiple nodes will have confusion around which is most recent write.

ⓐ Last write wins → each replica stores most recent value and overwrite older values.

↳ as we say writes are concurrent, so write order is undefined. Attach timestamp to each write and choose most recent one.

↳ This might lead to data lose.

↳ Approach for handling this could be to assign version numbers to each write operation and subsequently merge basis that on re-writes.

↳ The old values are deleted directly, kept in database with help of delete markers - called tombstones.

Hope you enjoyed the chapter Summary.

Subscribe for more such content.

YT Channel - MsDeep Singh

Happy Learning ☺