**Database Replication**

Replication

1. Having a copy of DB helps in **fault tolerance**, hence data is available, even if one of the DB is down.
2. Having replica of DB in multiple geographical locations **reduces latency**. Suppose a YouTube video was stored first in America, but someone from India wants to access it. Each time request and response from India to America might take time due to network speed/ latency, hence keeping a copy of same video in Mumbai, will be faster to access within India.
3. Generally master DB is used for Writes and all slave DBs are used for write operations. Having DB replicas **increases overall performance** of system as their might be multiple teams who wish to read data from database at once, hence keeping multiple slave DBs will reduce their latency too.

Replication Lag

The time required to update data from master DB to slave DB is called replication lag.

MASTER SLAVE

x = 1

x = 1

Replication lag (r time) x = ~~1~~  -> 2

(t2 time)

x = 2 (t1 time)

Suppose a value in master Db gets updated in t1 time.

Value is updated in slave DB in t2 time.

Replication lag is r.

If, r < (t2 – t1) [at t2, user will access fresh value when read from slave DB]

Else, r > (t2-t1) [at t2, user will access stale value when read from slave DB].

This problem becomes worse when there are multiple replicas of DB.

Hence, for one property in DB, people might get 2 different values when read from different DBs, hence leading to inconsistency.

To solve this, there are various consistency models or consistency algorithms.

Consistency Algorithms:

S1

1. **Synchronous replication (Read after write)**

Write 1

S2

ACK1

Write 2

master

ACK2

Write 3

S3

ACK3

Here, whenever a Write operation is performed on master DB.

All slave DBs are also updated immediately and all slaves send an acknowledgement to master that write to slave is done.

Once master receives all acknowledgements, it responds to write issuer that write operation is complete.

Advantage:

1. User always reads fresh value from any DB as write is not completed until all slaves are written.
2. Replication lag = 0

Disadvantage:

1. Write operation becomes time taking than read, as all slaves need to be updated.
2. Suppose, a slave DB goes down and it is unable to send acknowledgement, in that case, write operation is said to have failed.

To solve the problem of synchronous replication we have async replication.

1. **Asynchronous replication**

Here, as soon as a write occurs on master DB, it does not wait for acknowledgement from all Slaves.  
Writing to slave happens in background.

Write issuer is informed that write is successful as soon as master DB is written.

Advantage:

Writes are faster.

Write does not fail if any slave cannot send acknowledgement.

Disadvantage:

1. User might see stale data in few slave DBs i.e., we cannot expect high consistency in this algorithm.

Both algorithms should be used in different use cases.

Suppose it’s a banking application, where consistency of data is of utmost importance, use synchronous replication.

If some inconsistency can be tolerated, then use asynchronous replication.

It’s a performance⬄consistency trade-off.

High performance - async replication.

High consistency – sync replication.

1. **Semi-synchronous replication**

Here, master DB waits for only one or two acknowledgements and then tells write issuer that write is complete.

This ensures that at least one/two slaves are updated.

Writes are still faster.

Disadvantage: Not all slaves might have fresh values.

**Snapshot of DB**

Snapshot is different than replica.

Snapshot contains state of DB at particular time.

Having a snapshot taken at specific time intervals, help us in rolling back to that state in case there is any database failure.

Replica helps in fault tolerance and high availability.