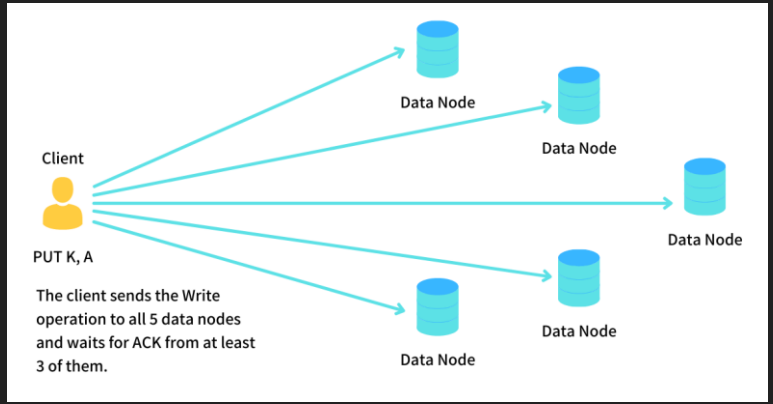
Leaderless Replication

Types of replication

- Single leader
- Multi leader
- Leaderless

Leaderless Replication Overview

- Any replica can accept a write from a client
- Send reads and writes to all nodes in parallel, once a certain predefined threshold of nodes return a success value, the client is told that the read/write was successful



Keeping data up to date

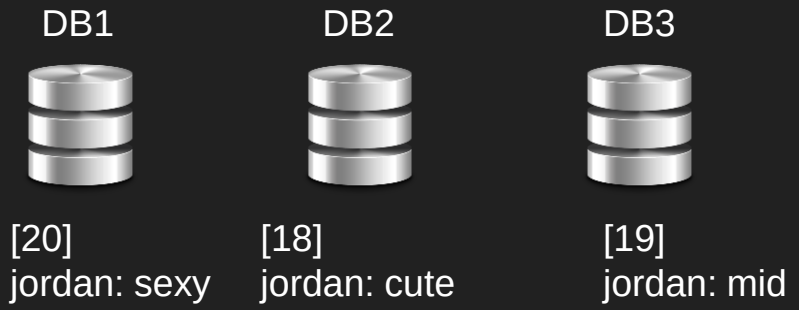
- Anti-Entropy
- Read Repair

Anti Entropy

Background process looks at multiple nodes and their stored values, use version numbers of the data in order to attempt to make sure that each replica holds the most up to date copy of the data

Read Repair

Idea: Since we are reading from multiple replicas in parallel, take the most up to date piece of data and propagate it to the other replicas that had outdated data



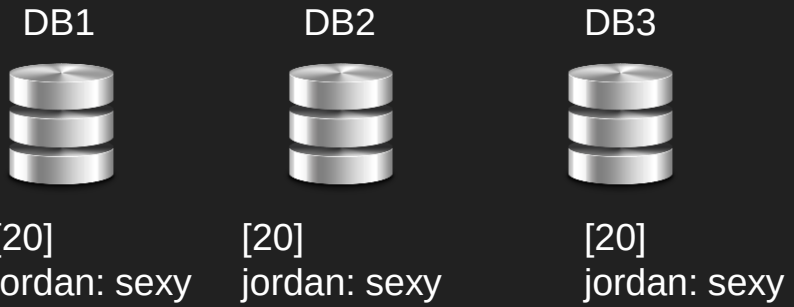
Read Repair

Idea: Since we are reading from multiple replicas in parallel, take the most up to date piece of data and propagate it to the other replicas that had outdated data

Value of key jordan?
argmax [(20, sexy), (18, cute), (19, mid)]
Result: jordan = sexy
Need to write back to db2 and db3

Read Repair

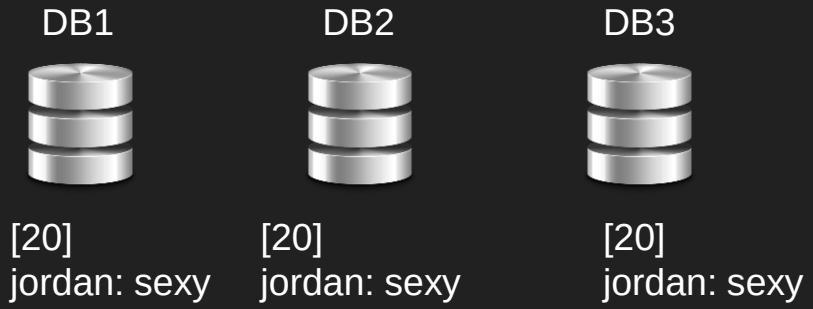
Idea: Since we are reading from multiple replicas in parallel, take the most up to date piece of data and propagate it to the other replicas that had outdated data



Value of key jordan?
argmax [(20, sexy), (18, cute), (19, mid)]
Result: jordan = sexy
Need to write back to db2 and db3

Read Repair

Idea: Since we are reading from multiple replicas in parallel, take the most up to date piece of data and propagate it to the other replicas that had outdated data



Value of key jordan?
argmax [(20, sexy), (18, cute), (19, mid)]
Result: jordan = sexy
Need to write back to db2 and db3

Problem: How can we guarantee that at least one of the replicas that we read from has the most up to date data?

Quorums

Idea: Let’s imagine we have n database replicas

- Suppose a client only has a successful write if at least w nodes are written to
- Suppose a client only has a successful read if at least r nodes are read from
- If w + r > n, at least one of the nodes that is read from will have an updated copy of the data and can thus read repair the others

Typically, set n to be an odd number and w = r = (n+1)/2, can tolerate up to (n-1)/2 node failures in this case

Quorum illustration

7 database replicas

W = 4

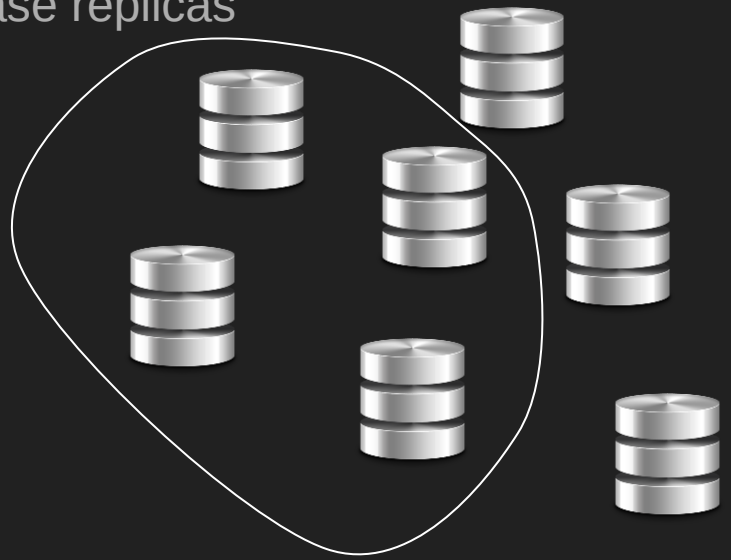
R = 4

Quorum illustration

7 database replicas

W = 4

R = 4

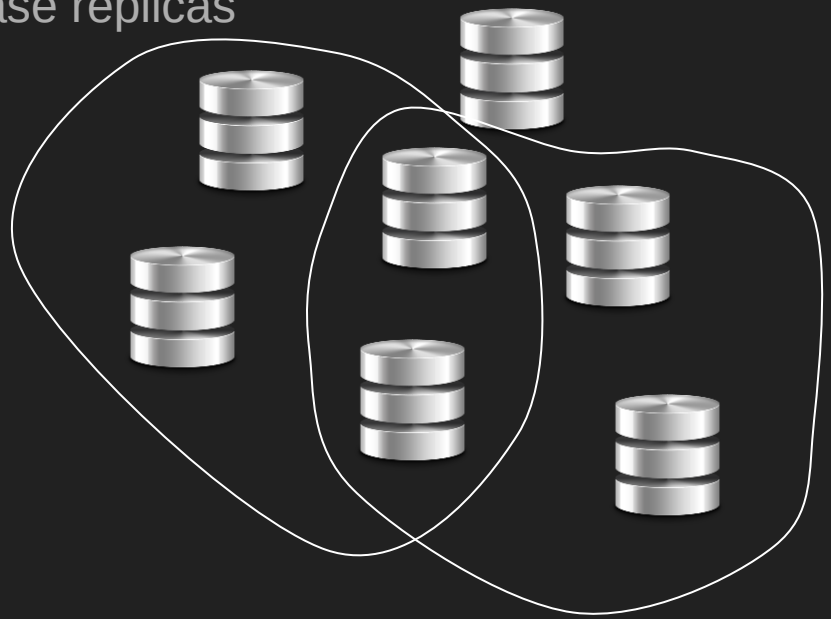


Quorum illustration

7 database replicas

W = 4

R = 4



Quorums are not perfect

- If we try to write to w nodes to make a quorum, but fewer than w writes succeed, the client is told that their write failed, but the nodes that did successfully process the write do not undo it
- If you restore a failed node with an up to date value from one key using a node with an older value for that key, it will lose the up to date value
- Can still be write conflicts
- Sloppy quorums

Write Conflict Example

DB1

DB2

Me

Angry Ex Girlfriend

Write Conflict Example

Write Conflict Example

Sloppy Quorums

- Let’s say I’m a Facebook SWE, and all the profile data is stored on 99 database nodes, W = 50, R = 50
- However, all of the 99 profile nodes have crashed, so I instead redirect all writes for the data to 50 of the nodes which hold Facebook messages
- Even if the profile nodes come back up, I can read 50 of them and not get up to date data
- Need to perform a hinted handoff, and transfer the data back to the right place

Leaderless Replication Conclusion

- Pros:
  - Can work quite well in practice with a cross datacenter solution by having the quorum write parameter be small enough that all writes can go to one datacenter
- Cons:
  - Slower reads as a result of multiple queries to nodes (if using read repair)
  - Still have to reason about write conflicts
  - Quorums are not perfect, provide illusion of strong consistency when in reality this is often not true