# Dynamo Architecture Analysis

Note - Based on Dynamo paper published

## Part II - System Architecture 1

Summary

**Table 1: Summary of techniques used in *Dynamo* and their advantages.**

| Problem | Technique | Advantage |
|---|---|---|
| Partitioning | Consistent Hashing | Incremental Scalability |
| High Availability for writes | Vector clocks with reconciliation during reads | Version size is decoupled from update rates. |
| Handling temporary failures | Sloppy Quorum and hinted handoff | Provides high availability and durability guarantee when some of the replicas are not available. |
| Recovering from permanent failures | Anti-entropy using Merkle trees | Synchronizes divergent replicas in the background. |
| Membership and failure detection | Gossip-based membership protocol and failure detection. | Preserves symmetry and avoids having a centralized registry for storing membership and node liveness information. |

① System Interface

get(key) → look up object replica associated with key

↳ return single object or list along with context

↳ in case of conflicting changes.

put(key, context, object)

↳ includes object metadata example version

passed via MD5 hash → generate 128 bit identifier

↳ determine storage node.

## ② Partitioning Algorithm → dynamically partition data across the nodes.

↳ Consistent Hashing is used.

watch consistent Hashing video on channel - 'MsDeep Singh'

Benefit of consistent Hashing

addition / removal of node affect only neighbouring data.

DDB uses concept of virtual nodes in Hash Ring to avoid hotspots.

↳ single node is assigned multiple positions in ring.

### Benefits of virtual nodes

① node removed → even load distribution in remaining nodes

② node added → node accepts equal load from all available nodes.

③ Heterogeneity in Infra → no of virtual nodes can be decided basis capacity of node.

Example m4.large → 10 virtual nodes

m4.4x large → 40 virtual nodes

## ③ Replication

↳ for high availability and durability.

④ Each key is assigned to coordinator node. → node handling read/write

↳ store key locally within its range.

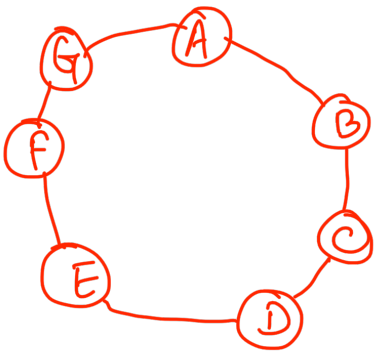
↳ replicates to (N-1) successor nodes in clockwise.

first node among top N nodes in preference list

↳ only physical nodes should be considered.

to avoid any forecast issues, the preference list of nodes contains >N nodes.

put(K) → belongs between A & B




① Should be stored on B as per consistent Hashing logic

② for replication, stored on C and D along with node B.

## ④ Data versioning

ⓐ async data replication in nodes – eventually consistent.

ⓑ Data is not replicated to all nodes due to network faults?

requirement – no data loss. example Amazon shopping cart

[watch 'why Cassandra don't use Vector Clocks video']

possibility of data addition to older version due to above.

↓

each put() operation creates new & immutable version of data

↓

system tries to handle it → syntactic reconciliation

↓

if version branching happens → due to multiple concurrent updates

↓

should be handled by client on next read.

⇒ Vector clocks are used to determine objects causality.

→ client specify version (→ context) to be updated.

→ Downsides → Vector clocks size grow if many severs are coordinating the write.

How to avoid?

ⓐ write handled by one nodes from preference list.

ⓑ write is handled by nodes other than preference list if network faults. → vector clock size grow

size of clock is limited to threshold.

oldest value removed after threshold.

## ⑤ Execution of get() and put()

All nodes in DDB are same → any node can receive read/write req.


Node selection

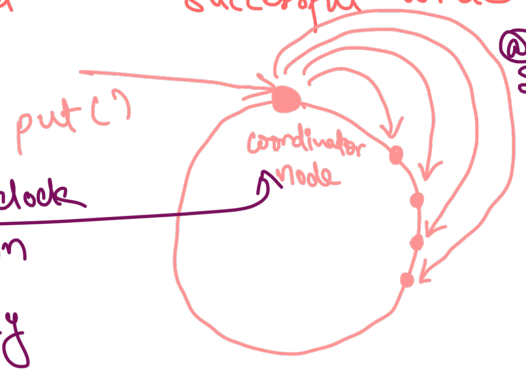
ⓐ via load balancer → ① as per load ② no data code at client end
② request will be forwarded to coordinator node.

ⓑ partition aware client library → ① directly to specific coordinator node.
② lower latency

### Consistency among replicas

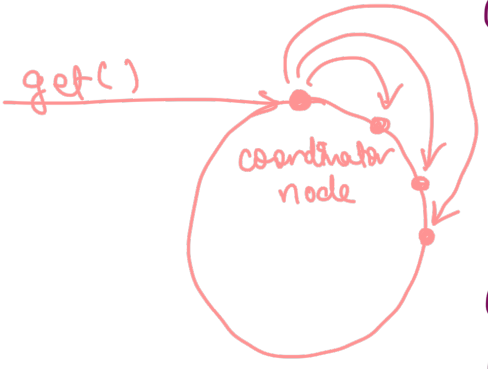
$R + W > N$ ← total no of nodes

- R: min no of nodes for successful read
- W: min no of nodes for successful write
- $N$: are configured < N to avoid extra latency




ⓐ Send write request to high ranking N nodes from preference list
ⓑ write success → (w-1) nodes respond

ⓐ generate vector clock for new version
ⓑ persist locally




ⓐ request data from highest ranked N nodes for data
ⓑ min R responses → return result to client
ⓒ if multiple versions → return all
ⓓ reconciliation of divergent versions
ⓔ write back final version

We'll be continuing System Architecture in next video.

→ How reads and writes are handled in failure scenarios.

Subscribe for more such content

YT- MsDeep Singh

Happy Learning