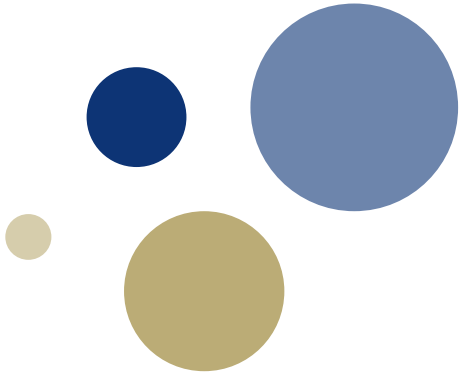
NTNU

Kunnskap for en bedre verden



Dynamo: Amazon's Highly Available Key-value Store

DeCandia et al.

Svein Erik Bratsberg, IDI/NTNU

What is Dynamo?

“Dynamo is a highly available key-value storage system that some of Amazon’s core services use to provide an “always-on” experience. To achieve this level of availability, Dynamo sacrifices consistency under certain failure scenarios. It makes extensive use of object versioning and application-assisted conflict resolution in a manner that provides a novel interface for developers to use.”

Amazon's Services' Requirements

- Strict operational requirements on performance, reliability and efficiency
- Needs to be highly scalable

"Reliability is one of the most important requirements because even the slightest outage has significant financial consequences and impacts customer trust."

Why not use a traditional SQL-DBMS?

- Most of Amazon services only store and retrieve data by primary key and do not require the complex querying and management functionality offered by SQL
- Isolation and strong consistency is not the most important properties
- Replication technologies typically choose consistency over availability, making data unavailable instead of inconsistent
  - leading to: Network partitions make data unavailable

System requirements and assumptions

- Simple read and write queries
- Data items uniquely identified by keys
- No operation span multiple data items

Dynamo targets applications that need to store objects that are relatively small (usually less than 1 MB).

Efficiency and Other Assumptions

- Commodity hardware infrastructure.
- Measures in the 99.9th percentile of the distribution.
- Services must be able to configure Dynamo to meet requirements in performance, cost efficiency, availability, and durability guarantees.
- Non-hostile environment: No security related requirements.
- Heterogeneous machines

Amazon's SLA

- A single page request can include requests to over 150 services
- Measure performance by average, median and expected variance (traditionally)
- Amazon wants to build a system that is good for ALL customers, so they focus on the 99.9th percentile.

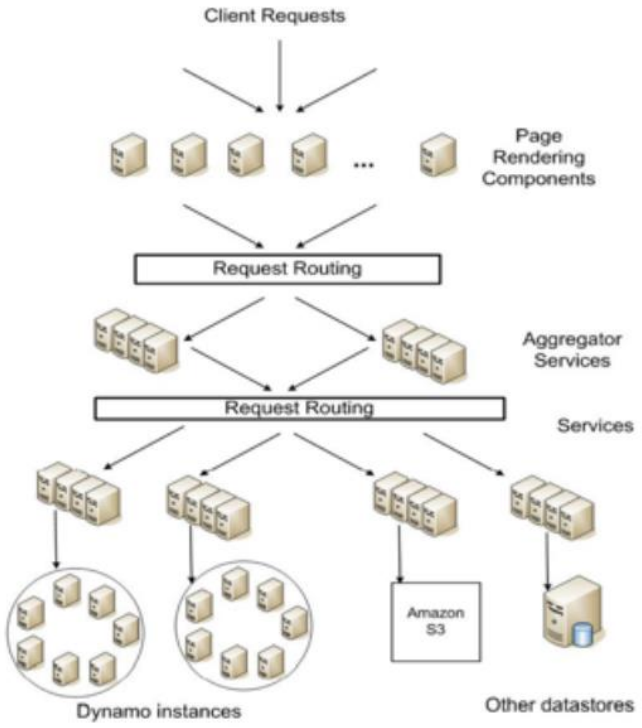


Figure 1: Service-oriented architecture of Amazon's platform

Design Considerations

- Give services control over their system properties, such as durability and consistency, and to let services make their own tradeoffs between functionality, performance and cost-effectiveness.
- Eventually consistent data store.
- “Always writeable” data store - conflict handling happens on read!
- Both client application and data store can resolve conflicts.
- Principles
  - Incremental scalability
  - Symmetry
  - Decentralization
  - Heterogeneity

Key Concepts in Dynamo

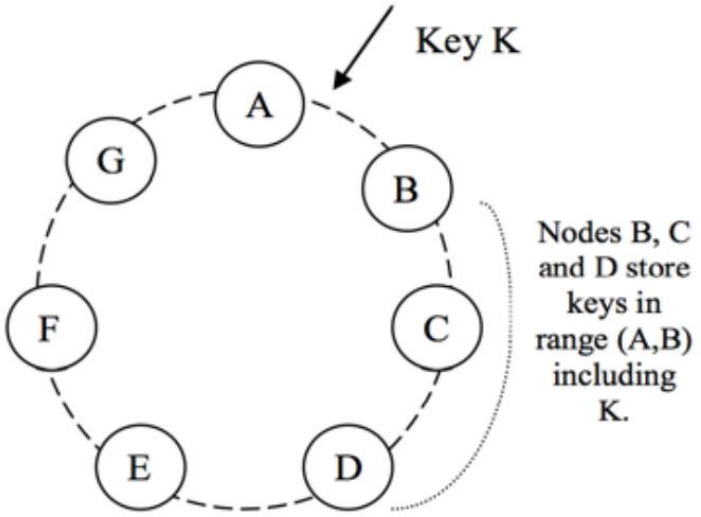
Interface:

- get(key)
- put(context, key, object)

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.

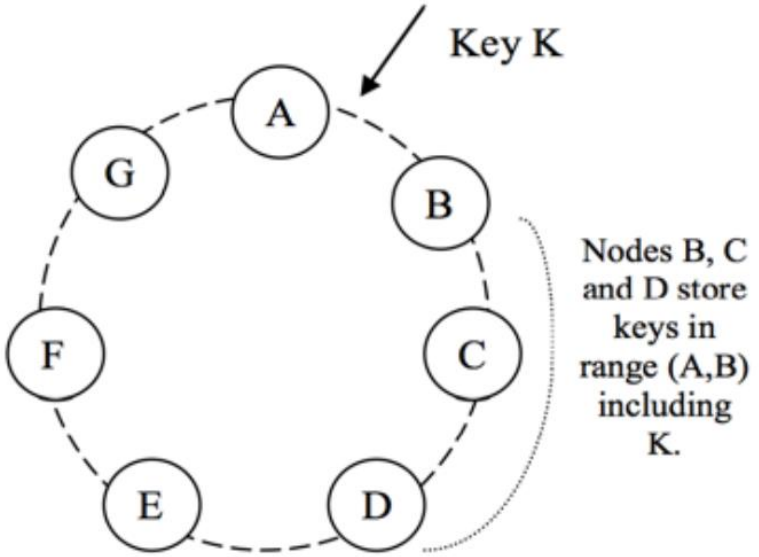
Consistent Hashing

- Departure or arrival of a node only affects its immediate neighbors!
- Assign nodes a random responsible area of the hash ring (circular keyspace)
- Challenges:
  - non-uniform data and load distribution.
  - oblivious to the heterogeneity of hardware
- Solutions:
  - virtual nodes - more powerful computers take responsibility of more virtual nodes



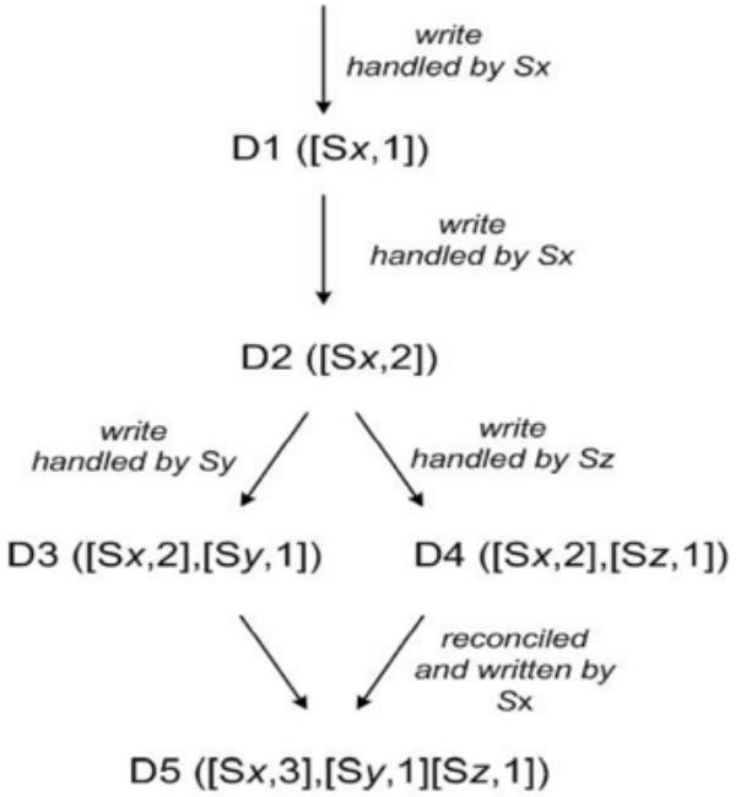
Replication

- Each data item is replicated at N hosts, where N is a parameter configured “per-instance”
- Each key, k, is assigned to a coordinator node
- The coordinator is in charge of the replication of the data items that fall within its range.
- A preference list store the nodes responsible for a key
- The preference list is constructed so the top N nodes are N physical nodes



Data Versioning

- Eventual consistency
- Each modification is a new and immutable version of the data.
- Usually the system handles version branching, but in some cases multiple versions are returned for the client for reconciliation.
- Vector clocks to keep track between different versions



Execution of get() and put()

Generic load-balancer

- Sends request to a random node, which then forwards it to the first of the top N nodes in its preference list
- Client does not have to link any code specific to Dynamo in its application

Partition aware client library

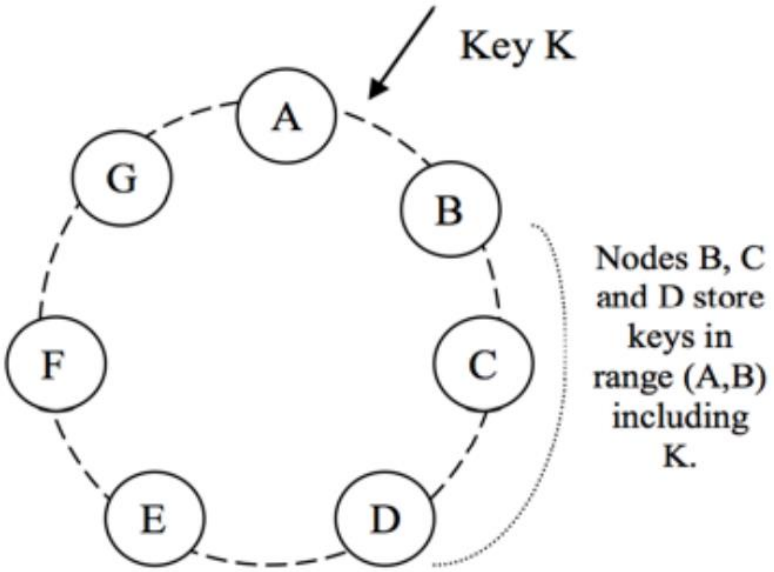
- Sends request to one of the top N nodes in the preference list
- Lower latency

Durability Tuning

- Dynamo uses two key configurable values **R** and **W**.
- **R** is the minimum number of nodes that must participate in a successful read operation.
- **W** is the minimum number of nodes that must participate in a successful write operation. E.g.: if coordinator gets a write, it writes a new version locally then sends the new version to N healthy nodes. If W-1 nodes respond, the write is successful.
- R and W does not necessarily need to be the top N nodes, rather the top N healthy nodes.

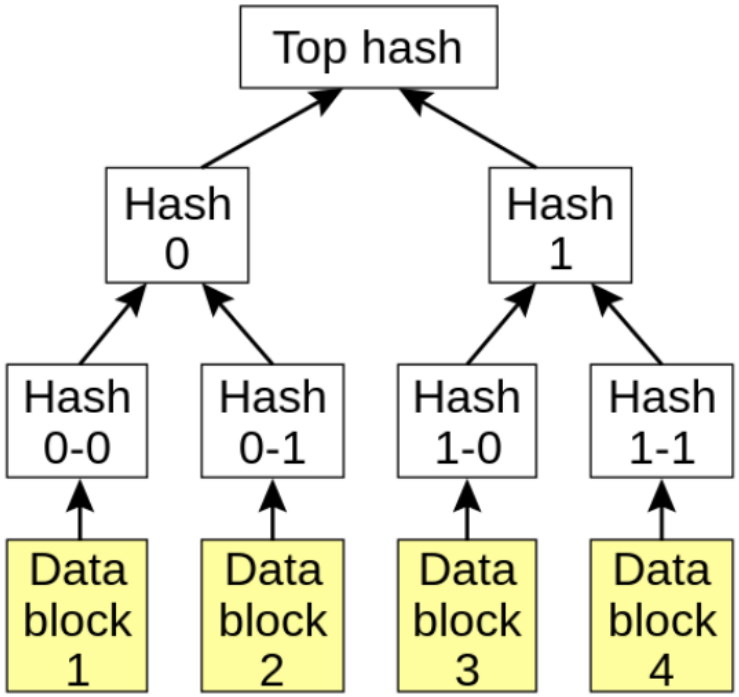
Handling failure - Hinted handoff

- Node A is temporary down during a write operation.
- Node D will receive the replica and a hint in its metadata that the object originally belongs to A.
- This will be kept in a separate local database and once A recovers, the object is transferred back to A and deleted.
  - Works best in low churn environments



Permanent failure – Replica reconstruction

- In some scenarios replicas become unavailable
- To detect inconsistenies between replicas Dynamo uses **Merkle Trees.**
- An entire branch can be checked independently for consistency without transferring entire data set or tree
  - Each node maintains a separate Merkle tree for each key range it hosts.
  - Two nodes exchange the root of the Merkle tree corresponding to the key ranges that they host in common.
  - Use tree traversal to determine if there is need for synchronization



Membership and failure detection

A gossip-based protocol

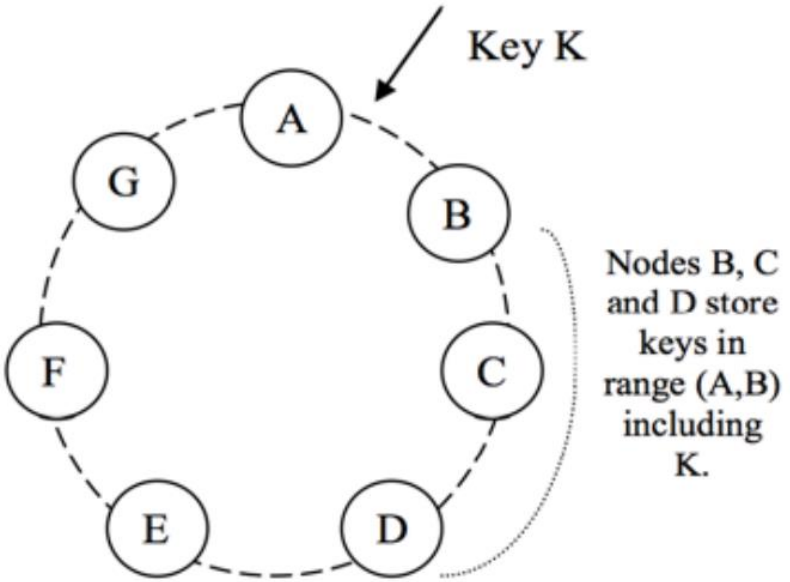
- Each node contacts a peer chosen at random every second
- The two nodes efficiently reconcile their persisted membership change histories.
- Seeds (nodes known to all nodes)
- Local notion of failure detection

Joining the ring:

- When a node starts for the first time, it chooses its set of tokens and maps nodes to their respective token sets.
- Reconciled using the same gossip-based protocol.
- Partitioning and placement information also propagates and each storage node is aware of the token ranges handled by its peers.

Adding and removing nodes

- Node X is added between A and B
- When X is added to the system, it is in charge of storing keys in the ranges (F, G], (G, A] and (A, X].
- This relieves Node B, C and D of some responsibility.
- These nodes will then offer to transfer the appropriate set of keys
- Removing a Node will lead to the process happening in reverse.



Software Architecture

In Dynamo, each storage node has three main software components:

1. **request coordination**
   - The coordinator executes the read and write requests on behalf of clients by collecting data from one or more nodes (in the case of reads) or storing data at one or more nodes (for writes).
   - Uses a state machine for each request.
   - Handles the request and performs read repairs if needed.
   - read-your-writes consistency: don't read stale data
2. **membership and failure detection**
3. **local persistence engine**
   - Dynamo's local persistence component allows for different storage engines to be plugged in
   - determined by access pattern and object size distribution

Experiences

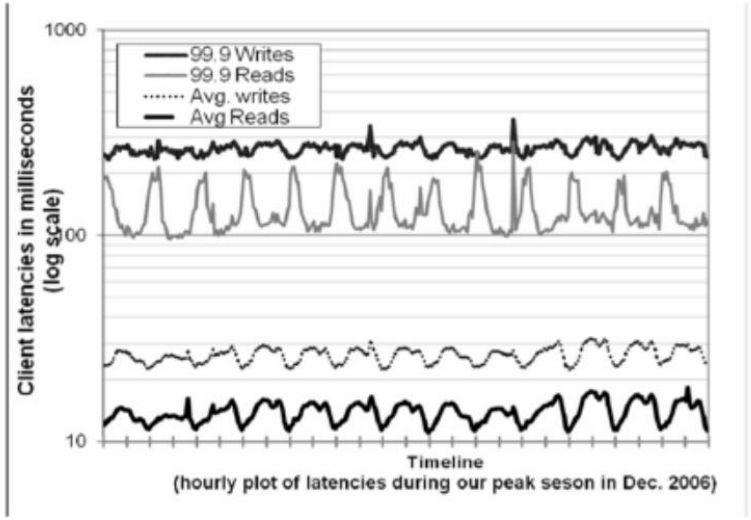
Reconcilement done in different ways:

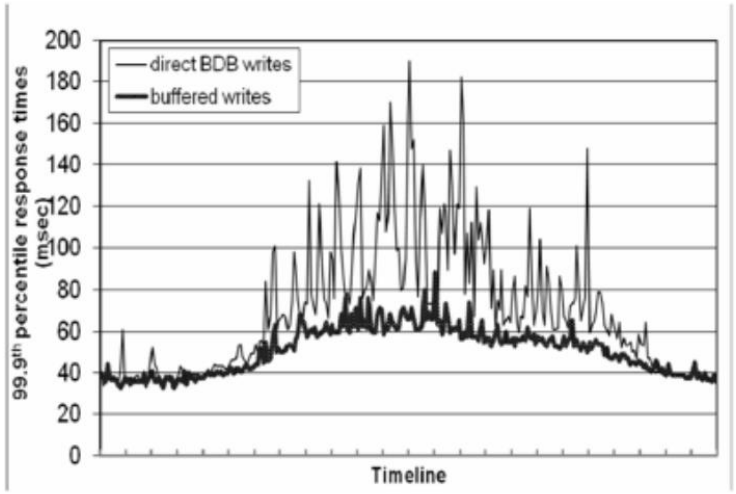
- Business logic specific
  - The client application handles divergent versions
- Timestamp based
  - Last write wins
- High performace read engine with good durability
  - R = 1, W = N

“The main advantage of Dynamo is that its client applications can tune the values of N, R and W to achieve their desired levels of performance, availability and durability.”

“The common (N,R,W) configuration used by several instances of Dynamo is (3,2,2).”

Performance





Load distribution

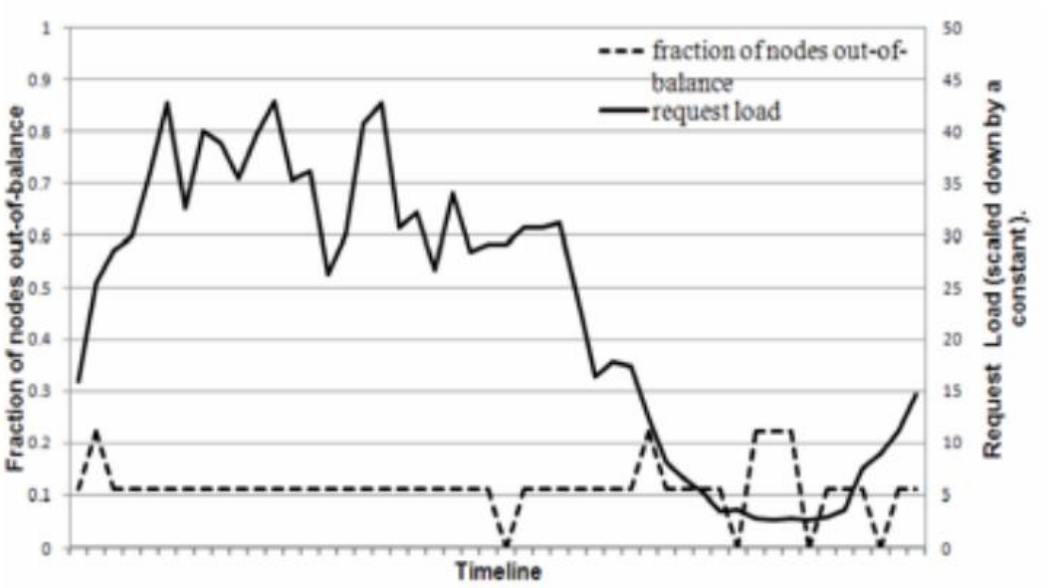
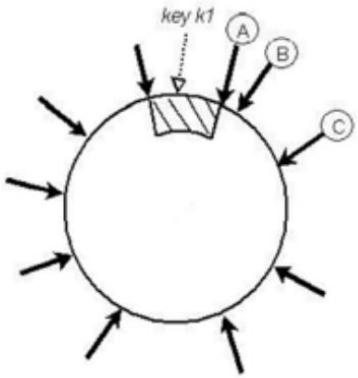


Figure 6: Fraction of nodes that are out-of-balance (i.e., nodes whose request load is above a certain threshold from the average system load) and their corresponding request load. The interval between ticks in x-axis corresponds to a time period of 30 minutes.

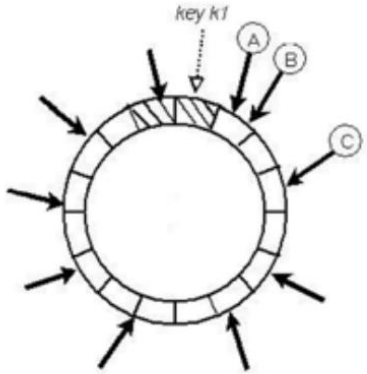
Partitioning strategies



Strategy 1

Random tokes and partitions by token value:

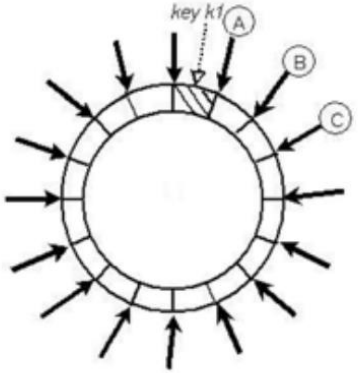
- Ranges vary
- Very resource intensive to add and remove nodes



Strategy 2

Random tokens but fixed partitions:

- Q equally sized partitions
- Consistent ranges
- Decoupling partitioning and partition placement

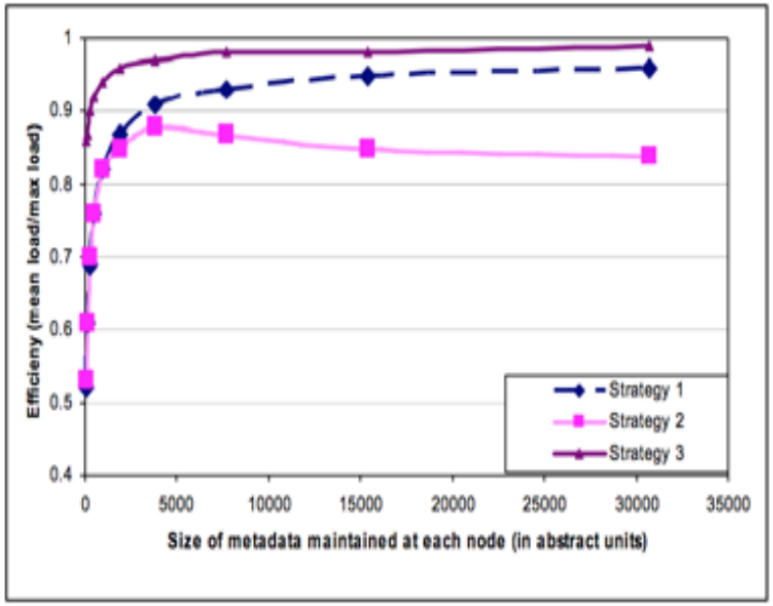


Strategy 3

Q/S tokens per node, fixed partitions:

- S is number of nodes in the system
- Decoupling partitioning and partition placement

Peformance and divergent versions



So how big of an issue is divergent versions?

24h shopping cart application:

- 99.94% of requests saw 1 version
- 0.00057% of requests saw 2 versions
- 0.00047% of requests saw 3 versions
- 0.00009% of requests saw 4 versions

This shows that divergent versions are created rarely! Experience also shows that it is increased numbers of concurrent writes that contributes to versioning. However, these are triggered by busy robots and not humans!

Client- vs server-driven Coordination

Client based:

- Move state machine to client nodes.
- Poll membership information from random node every 10 seconds
- Pull based approach scales better, but in worst case a node can be exposed to stale membership for 10 secs

	99.9th percentile read latency (ms)	99.9th percentile write latency (ms)	Average read latency (ms)	Average write latency (ms)
Server-driven	68.9	68.5	3.9	4.02
Client-driven	30.4	30.4	1.55	1.9

Dynamo’s conclusion

- Dynamo provides a highly available single-key storage system for smaller sized objects.
- Provides easy developer customization by needs in regards to availability, performance and durability.
- Does not scale into tens of thousands!

  each node has full list of routing info - lots of gossiping!
- May require application logic for data reconciliation
- Dynamo is the model for multiple NoSQL databases
- Available as a cloud service from Amazon