Dynamo : Amazon's Highly Available Key-value Store

잘따라가겠조 (Jiwoo Bang, Yejoon Sohn)
Distributed Computing Systems Laboratory
Department of Computer Science and Engineering
Seoul National University, Korea
5/29/2019

Index

- **Motivation**
- **Background**
- **Design Consideration**
- **System Architecture**
  - Partitioning algorithm
  - Data versioning
  - Hinted handoff
  - Replica synchronization
  - Membership and Failure detection
- **Implementation**
- **Evaluations**
- **Discussion**

Motivation

- **Amazon's platform**
  - Even the slightest outage has significant financial consequences and impacts customer trust
  - The platform is implemented on top of an infrastructure of tens of thousands of servers and network components located in many datacenters around the world
  - Persistent state is managed in the face of these failures - drives the reliability and scalability of the software systems
- **Dynamo**
  - Scalability
  - Simple data model (name, value)
  - Key-value store (big hashed table)
  - Highly available (sacrifice consistency)
  - Guarantee Service Level Agreements (SLA)

Background

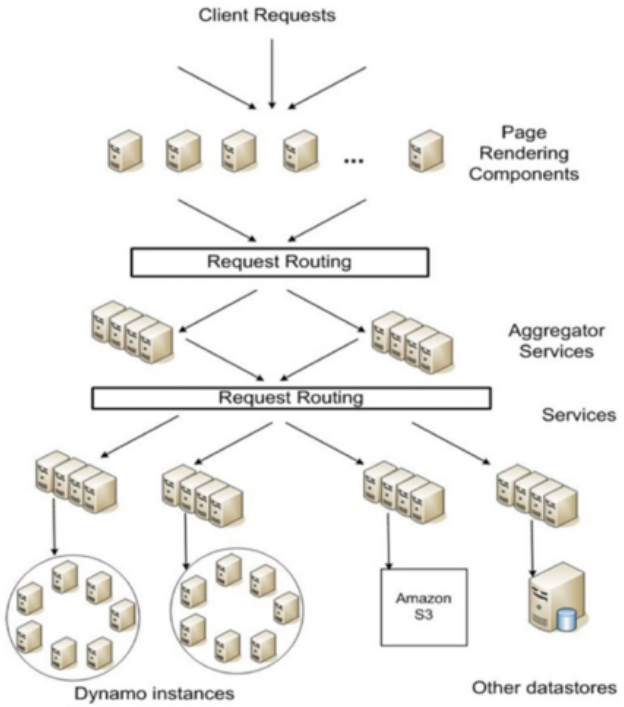
- **Query model**
  - Simple read and write operations to a data item that is uniquely identified by a key
  - Most of Amazon's services can work with this simple query model and do not need any relational schema
  - targeted applications - store objects that are relatively small (usually less than 1 MB)

Background

- **ACID properties**
  - Experience at Amazon has shown that data stores that provide ACID guarantees tend to have poor availability
  - Dynamo targets applications that operate with weaker consistency (the “C” in ACID) if this results in high availability.
- **Efficiency**
  - latency requirements which are in general measured at the 99.9th percentile of the distribution
  - Average performance is not enough

Background

- **Service Level Agreements (SLA)**
  - Application can deliver its functionality in bounded time
    - Every dependency in the platform needs to deliver its functionality with even tighter bounds
    - Service guaranteeing that it will provide a response within 300ms for 99.9% of its requests for a peak client load of 500 requests per second



Design Consideration

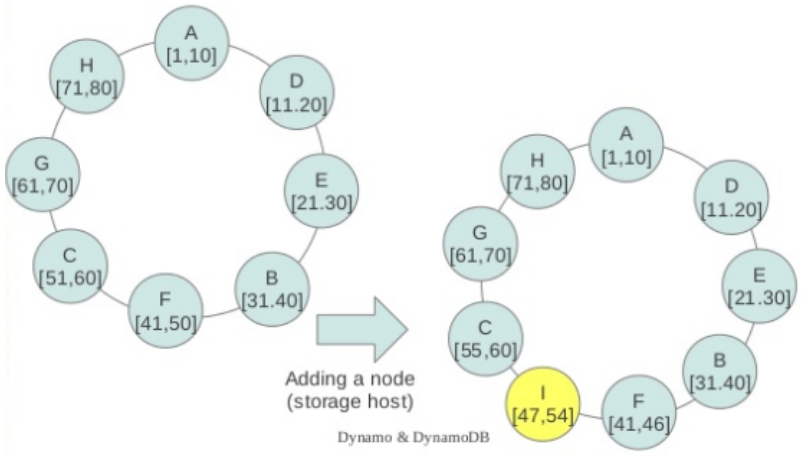
- **Sacrifice strong consistency for availability**
- **Conflict resolution**
  - executed during read instead of write : always writeable
- **Incremental scalability**
- **Symmetry**
  - Every node in Dynamo should have the same set of responsibilities as its peers
- **Decentralization**
  - In the past, centralized control has resulted in outages and the goal is to avoid it as much as possible
- **Heterogeneity**
  - This is essential in adding new nodes with higher capacity without having to upgrade all hosts at once

System Architecture

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.

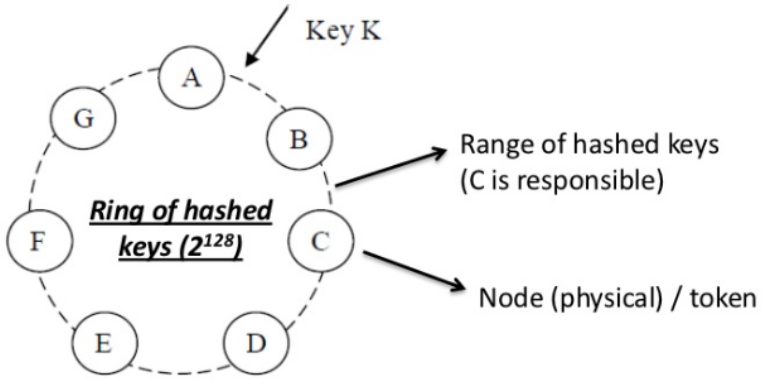
Partitioning Algorithm

- **Consistent hashing**
  - The output range of hash function is treated as a fixed circular space or *ring*
- **Replication**
  - An item can be stored in multiple consecutive nodes depending on the degree of replication N
  - Preference list : the list of (distinct physical) nodes that is responsible for storing a particular key



New nodes are randomly assigned "tokens".

All nodes have full membership info.



Partitioning Algorithm

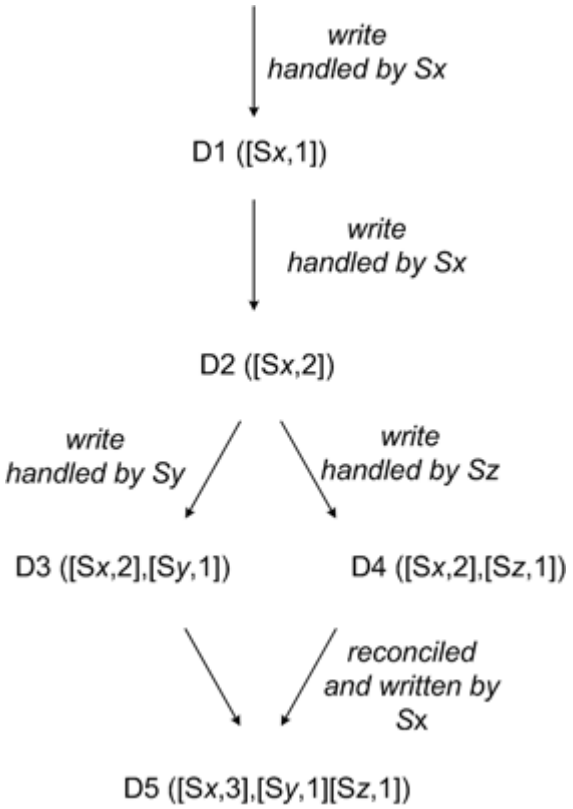
- **Virtual nodes**
  - Each node can be responsible for more than on virtual nodes (multiple random positions, tokens)
- **Advantages of using virtual nodes**
  - If a node becomes unavailable，the load handled by this node is evenly dispersed across the remaining available nodes
  - When a node becomes available again, or a new node is added to the system, the newly available node accepts a roughly equivalent amount of load from some of the other available nodes
  - The number of virtual nodes that a node is responsible can be decided based on its capacity, accounting for heterogeneity in the physical infrastructure

Data Versioning

- **Eventual consistency**
  - Updates are propagated to all replicas asynchronously
- **Put(key, context, value) call**
  - may return to its caller before the update has been applied at all the replicas
- **Get(key) call**
  - May return many versions of the same object
  - Distinct version sub-histories need to be reconciled.

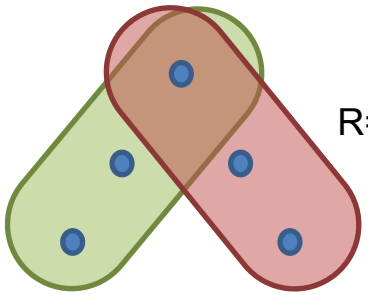
Data Versioning

- **Vector clock**
  - A vector clock is a list of (node, counter) pairs
  - Every version of every object is associated with one vector clock
  - Captures causality between different versions
  - Syntactic reconciliation: performed by system
  - Semantic reconciliation: performed by client

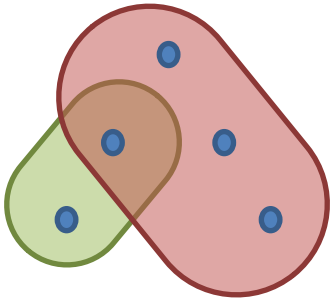


Execution of get() and put() operation

- **The user is able to issue commands with either of the following scenarios:**
  - A generic load balancer is invoked to direct the user's requests to the least utilization
  - Use a partition-aware library to direct the request to one of the data owners directly
- **The system requires two configurable values**
  - R: the number of available healthy nodes required for a successful reads
  - W: the number of available healthy nodes required for a successful writes



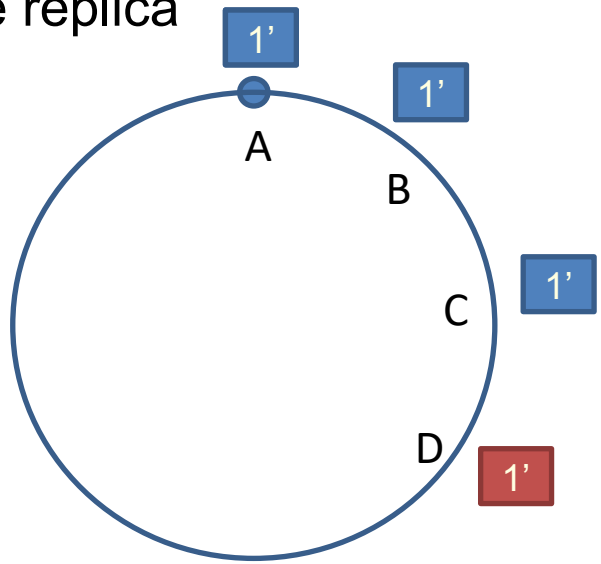
R=3, W=3, N=5



R=4, W=2, N=5

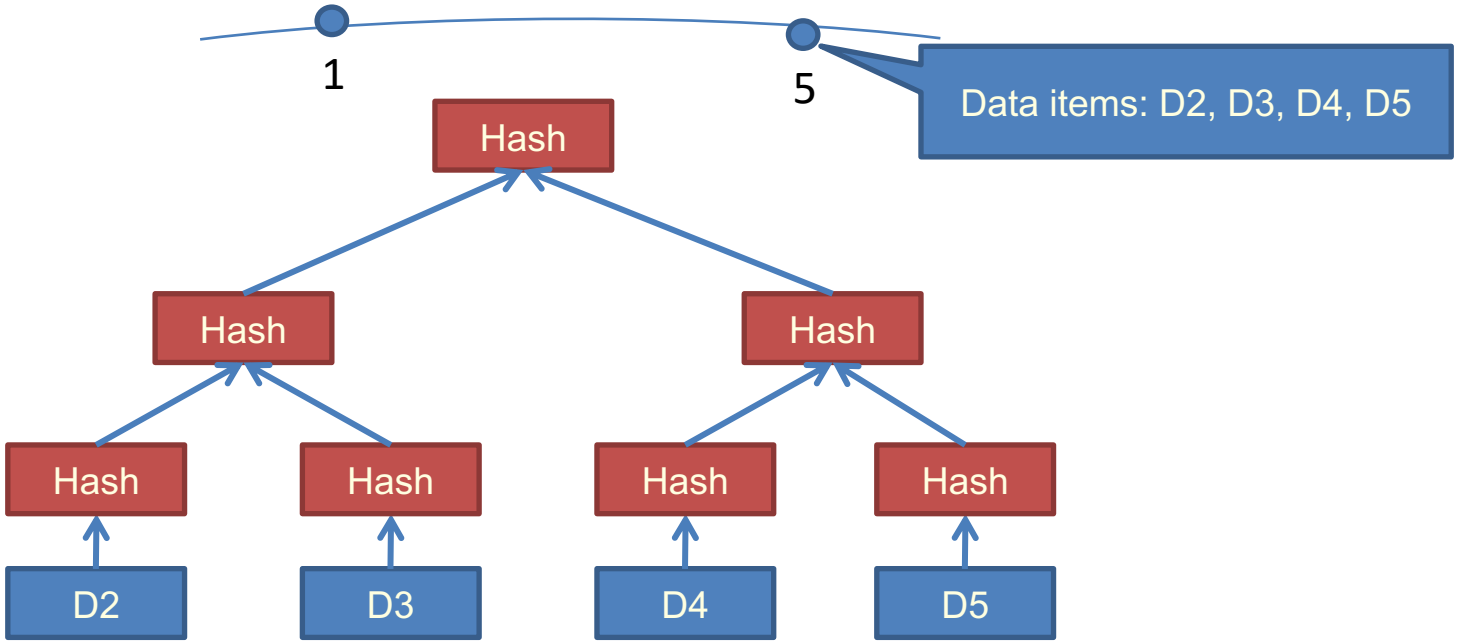
Hinted Handoff

- **Handling transient failures**
  - Sloppy quorum
    - All read/write operations are performed on the first N healthy nodes from the preference list
  - When A is unreachable, 'put' key will use D
  - When D detects A is alive
    - Send the replica to A
    - Remove the replica



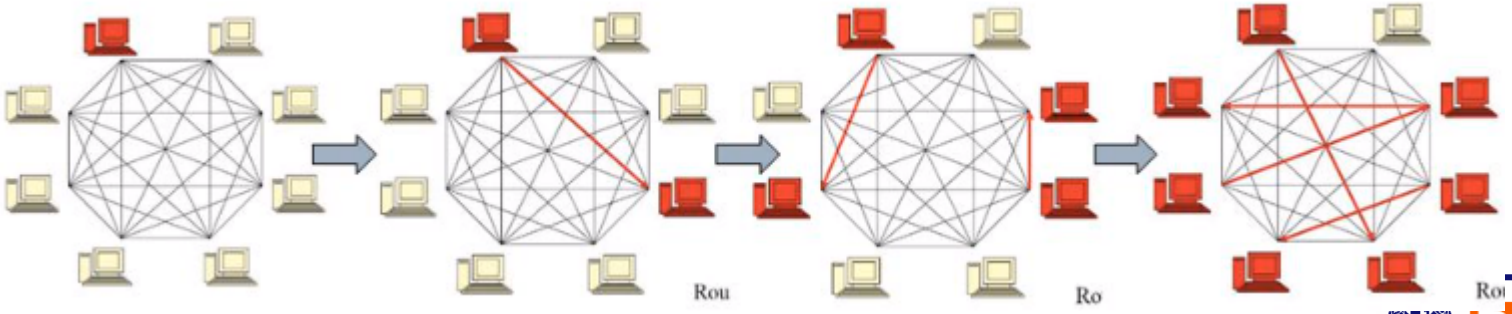
Replica Synchronization

- **Handling permanent failure**
  - Anti-entropy for replica synchronization
  - Use Merkle trees for fast inconsistency detection and minimum transfer of data
  - Exchange root of Merkle tree to check if the key ranges are up-to-date



Membership and Failure Detection

- **Membership**
  - A managed system
    - Administrator explicitly adds and removes nodes
  - Receiving node stores changes with time stamp
  - Gossiping to propagate membership changes
    - Eventual consistent view
- **Failure detection**
  - Passive failure detection
    - Use pings only for detection from failed to alive
    - A detects B as failed if it doesn't respond to a message
    - A periodically checks if B is alive again

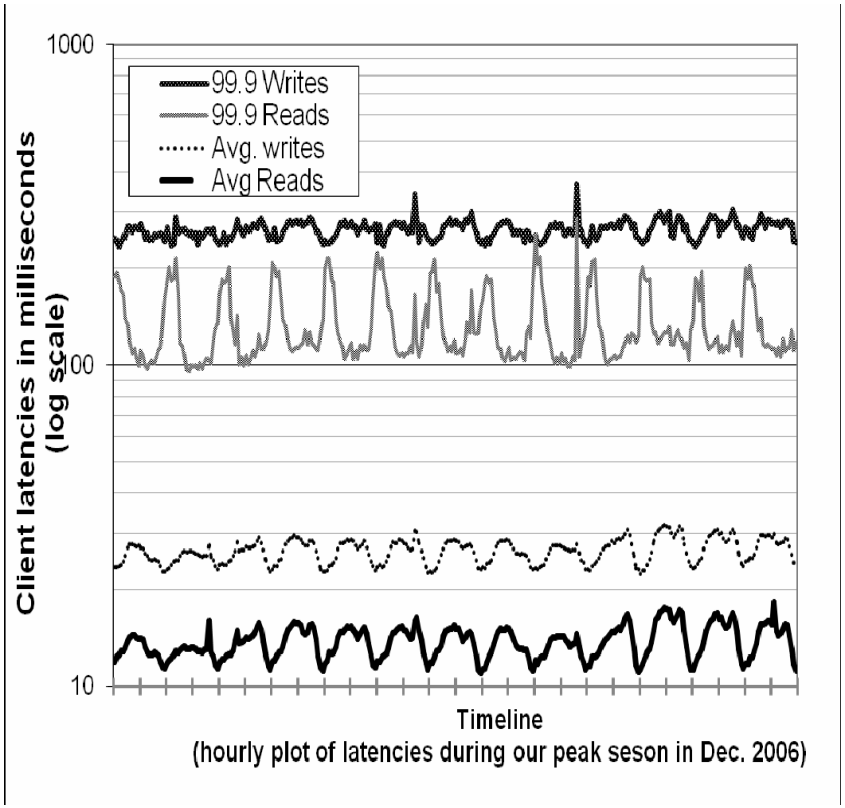


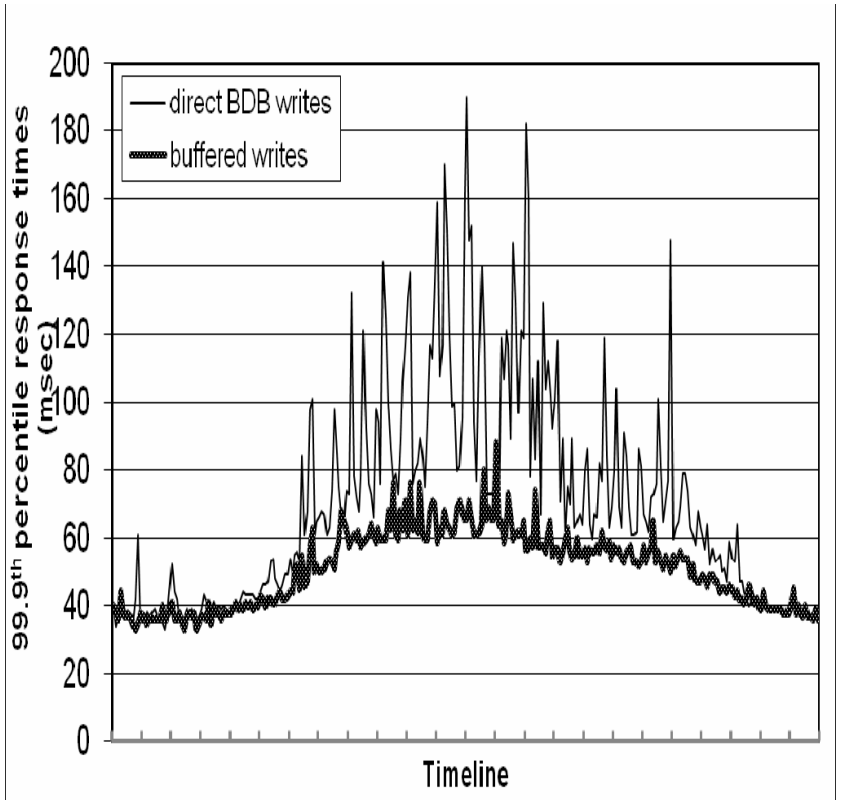
Implementation

- **Three main software components In Dynamo**
  - Membership & failure detection
  - Local persistence engine
    - Berkeley DB, MySQL
  - Request coordination
    - Read & Write
    - Read repair
- **Several services with different configurations**
  - Business logic specific reconciliation
  - Timestamp based reconciliation
  - High performance read engine
- **Client can tune the value (N, R, W)**
  - Value of W and R impact object
    - Availability, durability, consistency

Evaluation

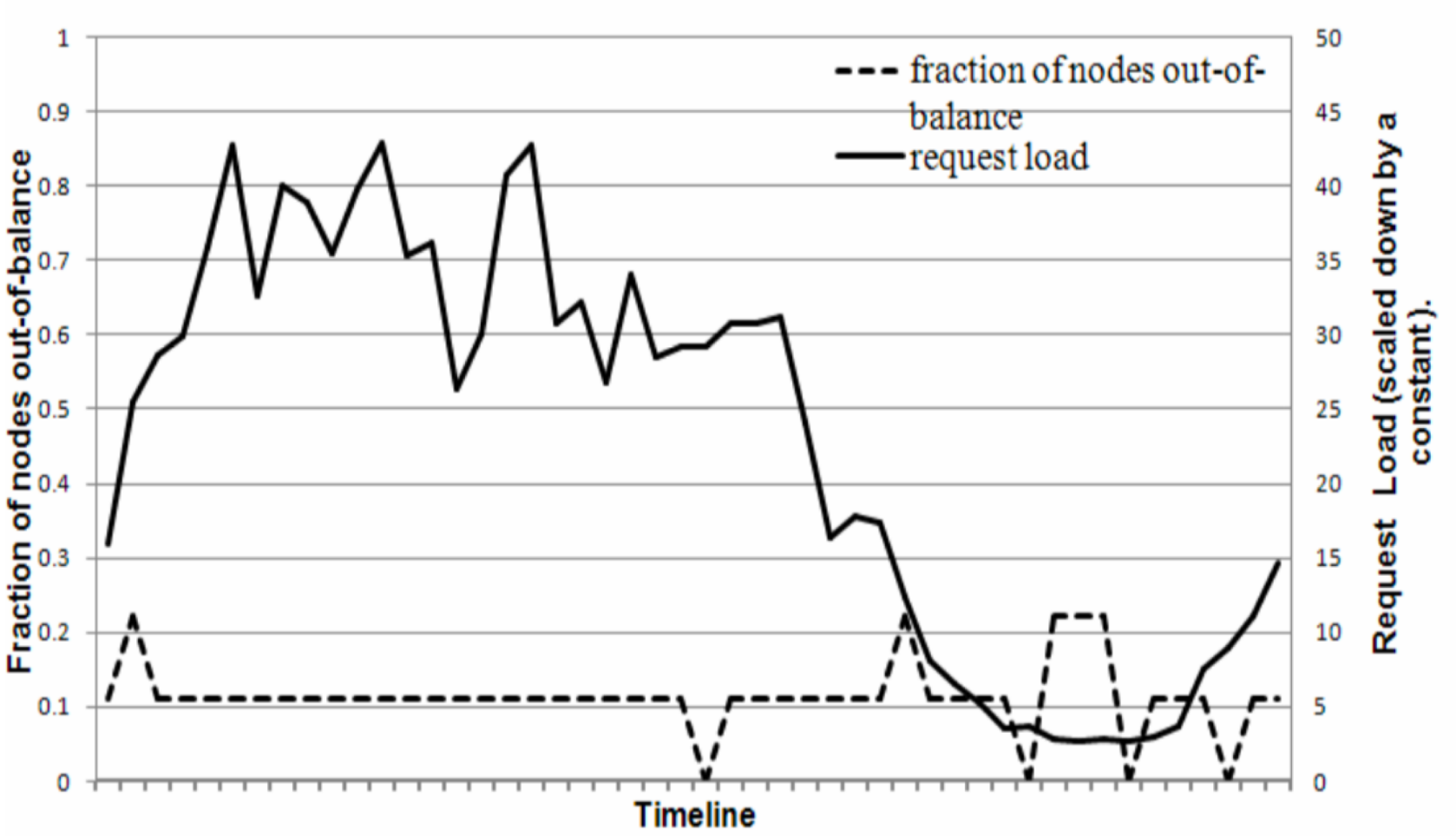
- **Balancing Performance Durability**
  - DynamoDB : Highly available data store (under 300ms)
  - Using memory buffer





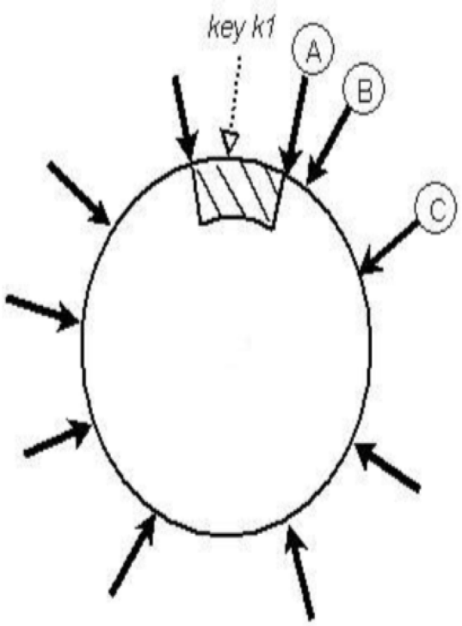
Evaluation

- **Ensuring Uniform Load distribution**

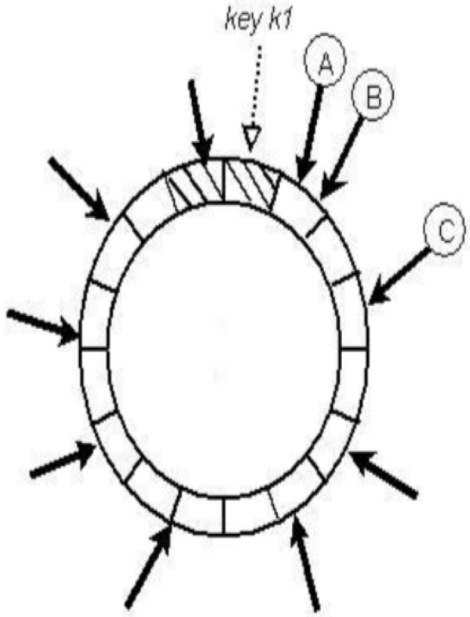


Evaluation

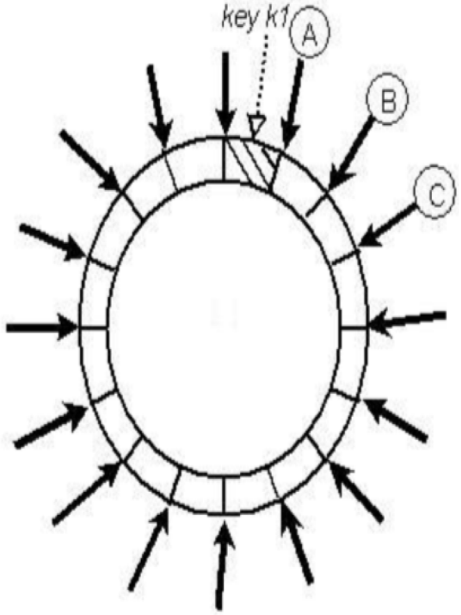
- **Dynamo Partitioning**
  - Three Strategies



Strategy 1



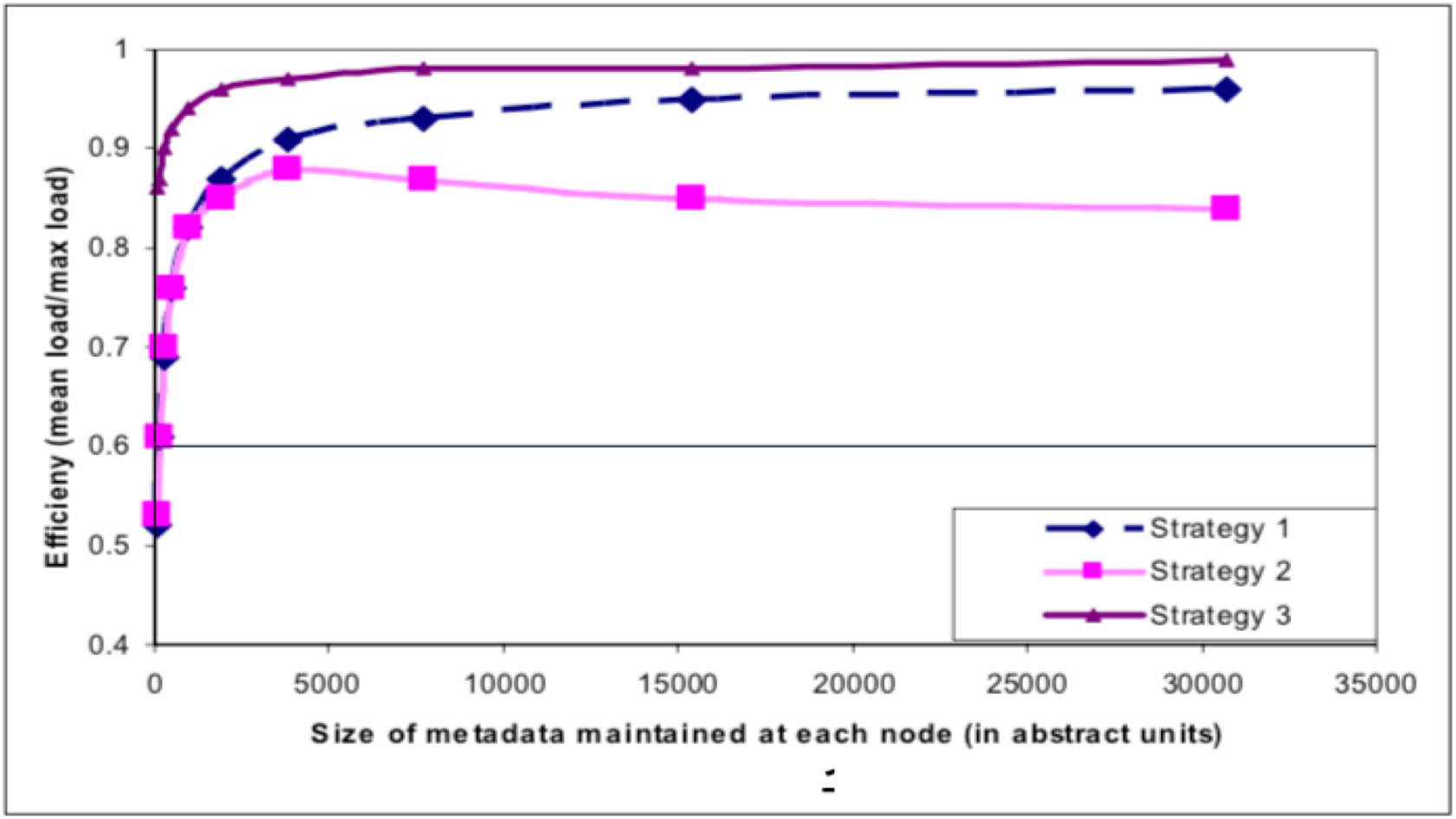
Strategy 2



Strategy 3

Evaluation

- **Dynamo Partitioning**
  - Three Strategies



Evaluation

- **Divergent Versions**
  - How & When
  - System Failure
  - Handling a large number of concurrent writers to a single data item
- **Client-driven or Server-driven**
  - Dynamo node for a read request and a write request

	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

Related Work

- **Apache Cassandra**
  - Column-based data model
    - Support more extensive data types
  - Distributed Hash Table
  - Tunable tradeoff
    - Consistency vs. Latency
  - No single point of Failure
  - Linearly scalable
  - Flexible partitioning, replica placement
  - Highly availability (eventual consistency)

Conclusion

- **Advantages**
  - Scalability
  - Simple data model
  - Highly available
  - Guarantee Service Level Agreements
- **Limitations**
  - Low consistent data
  - Simple query model compared to relational db
  - Difficulty in handling hot keys