Dynamo: Amazon's Highly Available Key-value Store

Giuseppe DeCandia, Deniz Hastorun,
Madan Jampani, Gunavardhan Kakulapati,
Avinash Lakshman, Alex Pilchin, Swaminathan Sivasubramanian, Peter Vosshall
and Werner Vogels

Motivation

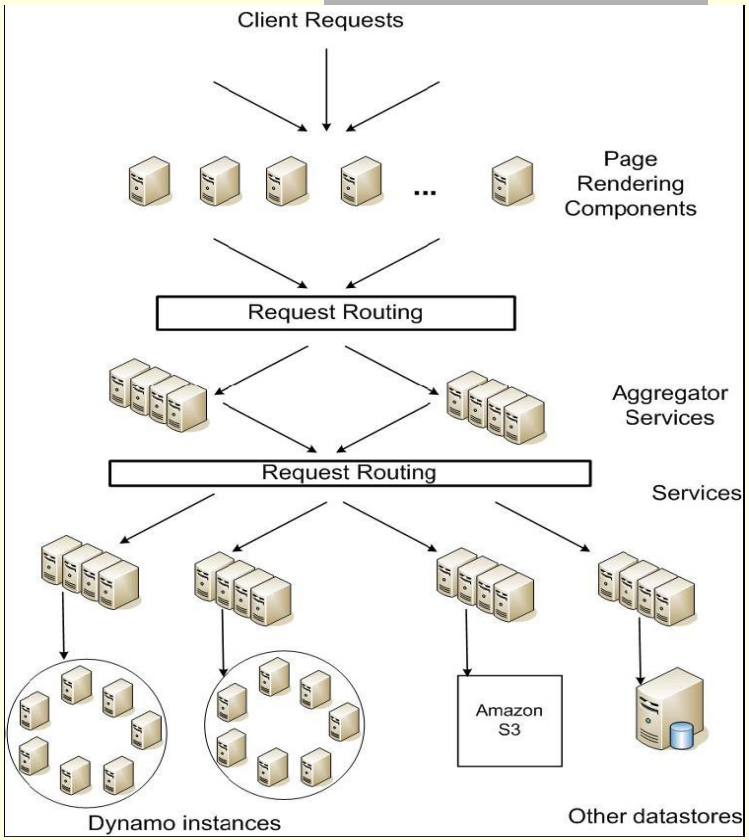
- Build a distributed storage system:
  - Scale
  - Simple: key-value
  - **Highly available**
  - **Guarantee Service Level Agreements (SLA)**

System Assumptions and Requirements

- *Query Model*: simple read and write operations to a data item that is uniquely identified by a key.
- *ACID Properties: Atomicity, Consistency, Isolation, Durability.*
- *Efficiency:* latency requirements which are in general measured at the 99.9th percentile of the distribution.
- Other Assumptions: operation environment is assumed to be non-hostile and there are no security related requirements such as authentication and authorization.

Service Level Agreements (SLA)

- Application can deliver its functionality in a bounded time: Every dependency in the platform needs to deliver its functionality with even tighter bounds.
- Example: 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.



Service-oriented architecture of Amazon's platform

Design Consideration

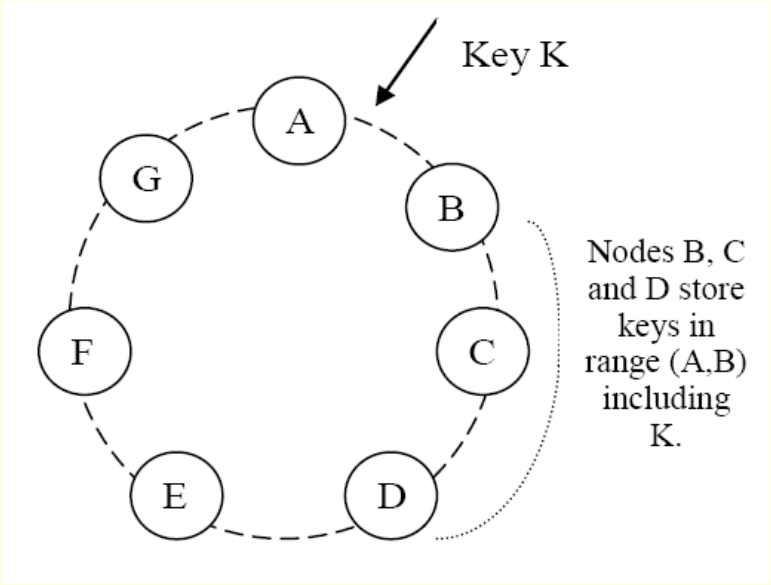
- Sacrifice strong consistency for availability
- Conflict resolution is executed during ***read*** instead of ***write***, i.e. “always writeable”.
- Other principles:
  - Incremental scalability.
  - Symmetry.
  - Decentralization.
  - Heterogeneity.

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.

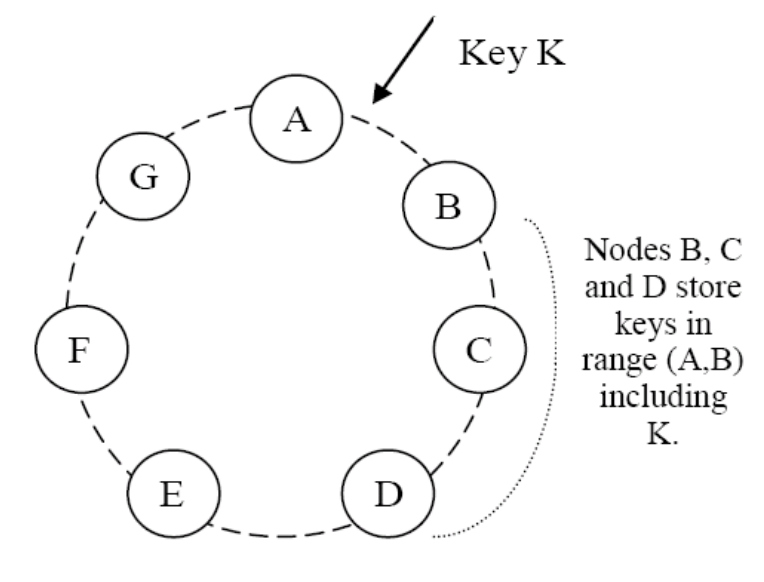
Partition Algorithm

- Consistent hashing: the output range of a hash function is treated as a fixed circular space or “ring”.
- ”Virtual Nodes”: Each node can be responsible for more than one virtual node.



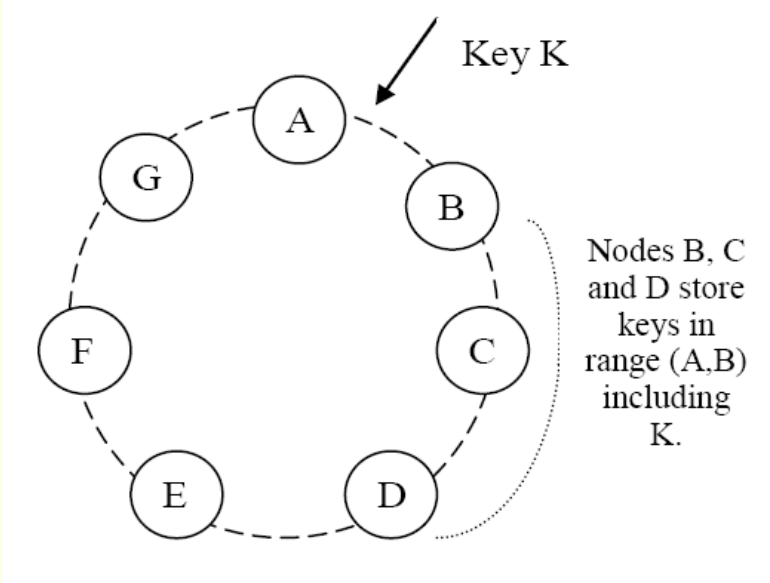
Advantages of using virtual nodes

- 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, the newly available node accepts a roughly equivalent amount of load from each of the other available nodes.
- The number of virtual nodes that a node is responsible can decided based on its capacity, accounting for heterogeneity in the physical infrastructure.



Replication

- Each data item is replicated at N hosts.
- "*preference list*": The list of nodes that is responsible for storing a particular key.



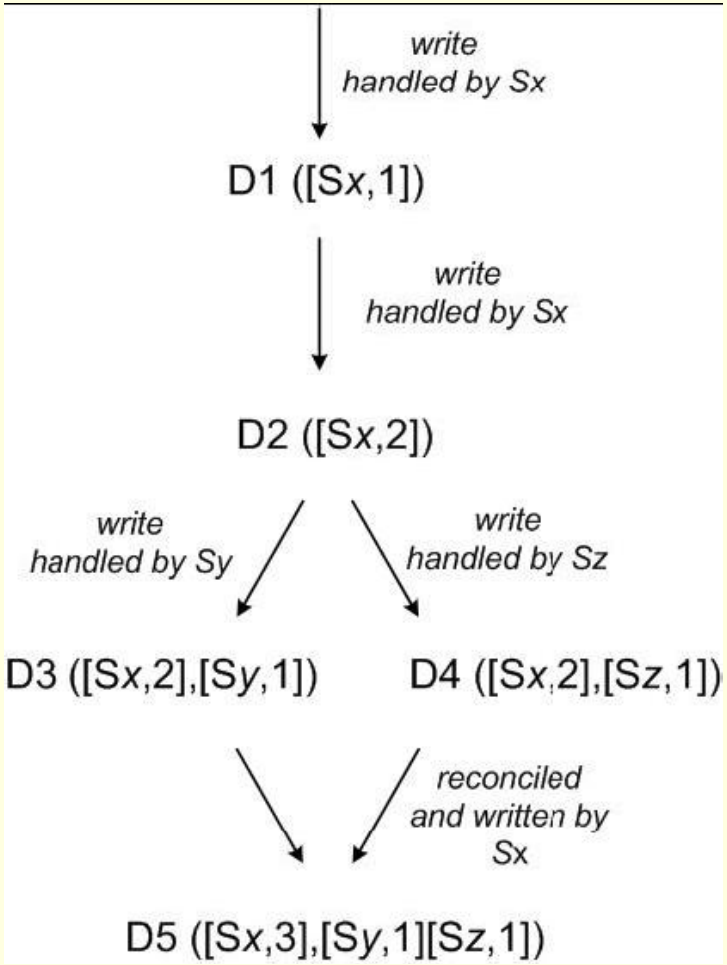
Data Versioning

- A put() call may return to its caller before the update has been applied at all the replicas
- A get() call may return many versions of the same object.
- Challenge: an object having distinct version sub-histories, which the system will need to reconcile in the future.
- Solution: uses vector clocks in order to capture causality between different versions of the same object.

Vector Clock

- A vector clock is a list of (node, counter) pairs.
- Every version of every object is associated with one vector clock.
- *If the counters on the first object's clock are less-than-or-equal to all of the nodes in the second clock, then the first is an ancestor of the second and can be forgotten.*

Vector clock example



Execution of get () and put () operations

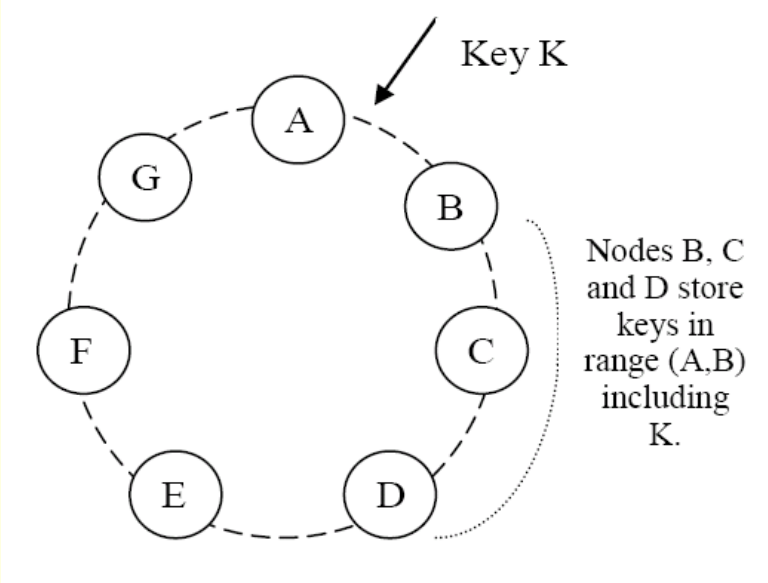
1. Route its request through a generic load balancer that will select a node based on load information.
2. Use a partition-aware client library that routes requests directly to the appropriate coordinator nodes.

Sloppy Quorum

- R/W is the minimum number of nodes that must participate in a successful read/write operation.
- Setting R + W > N yields a quorum-like system.
- In this model, the latency of a get (or put) operation is dictated by the slowest of the R (or W) replicas. For this reason, R and W are usually configured to be less than N, to provide better latency.

Hinted handoff

- Assume N = 3. When A is temporarily down or unreachable during a write, send replica to D.
- D is hinted that the replica is belong to A and it will deliver to A when A is recovered.
- Again: “always writeable”



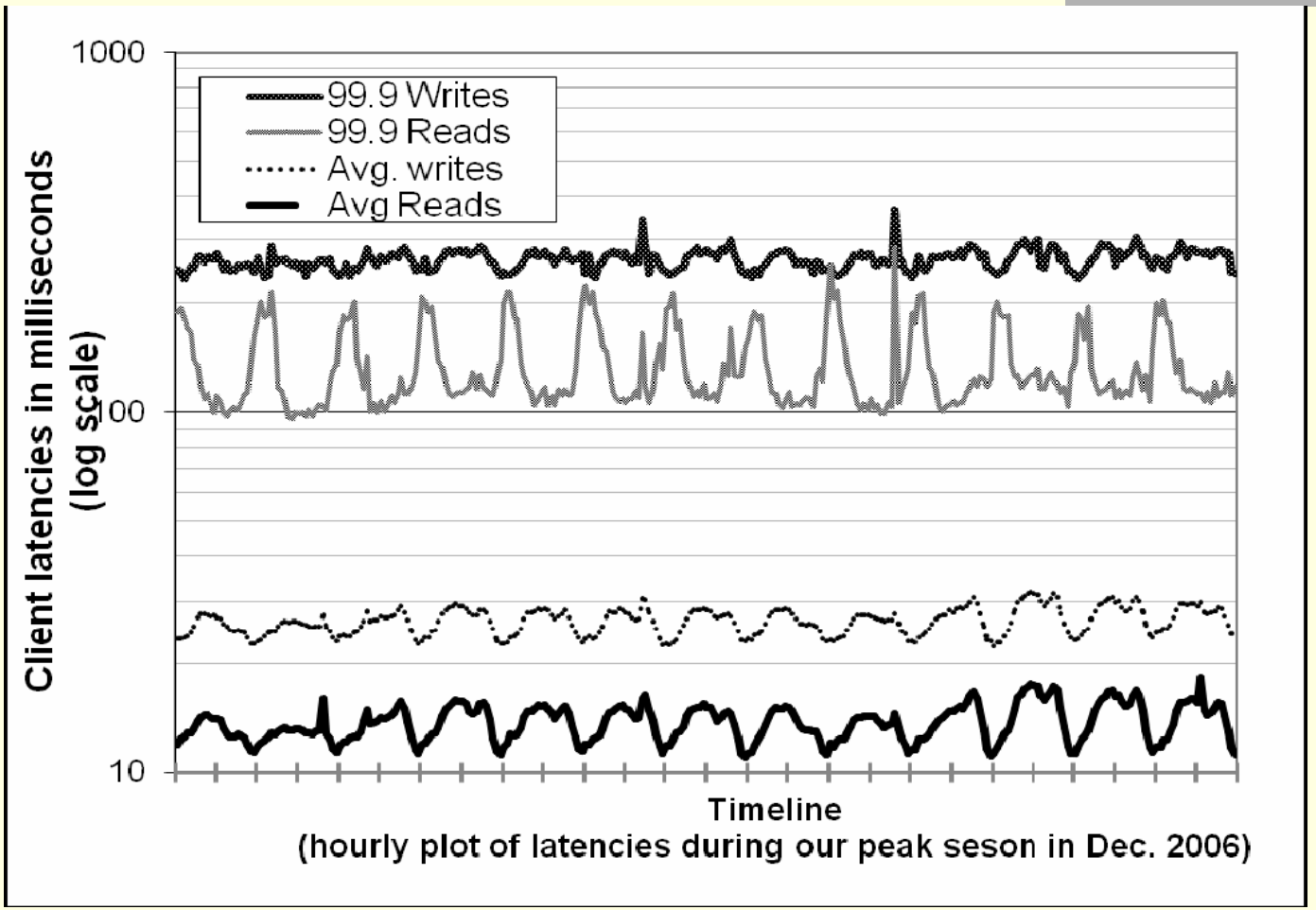
Other techniques

- **Replica synchronization:**
  - **Merkle hash tree.**
- **Membership and Failure Detection:**
  - **Gossip**

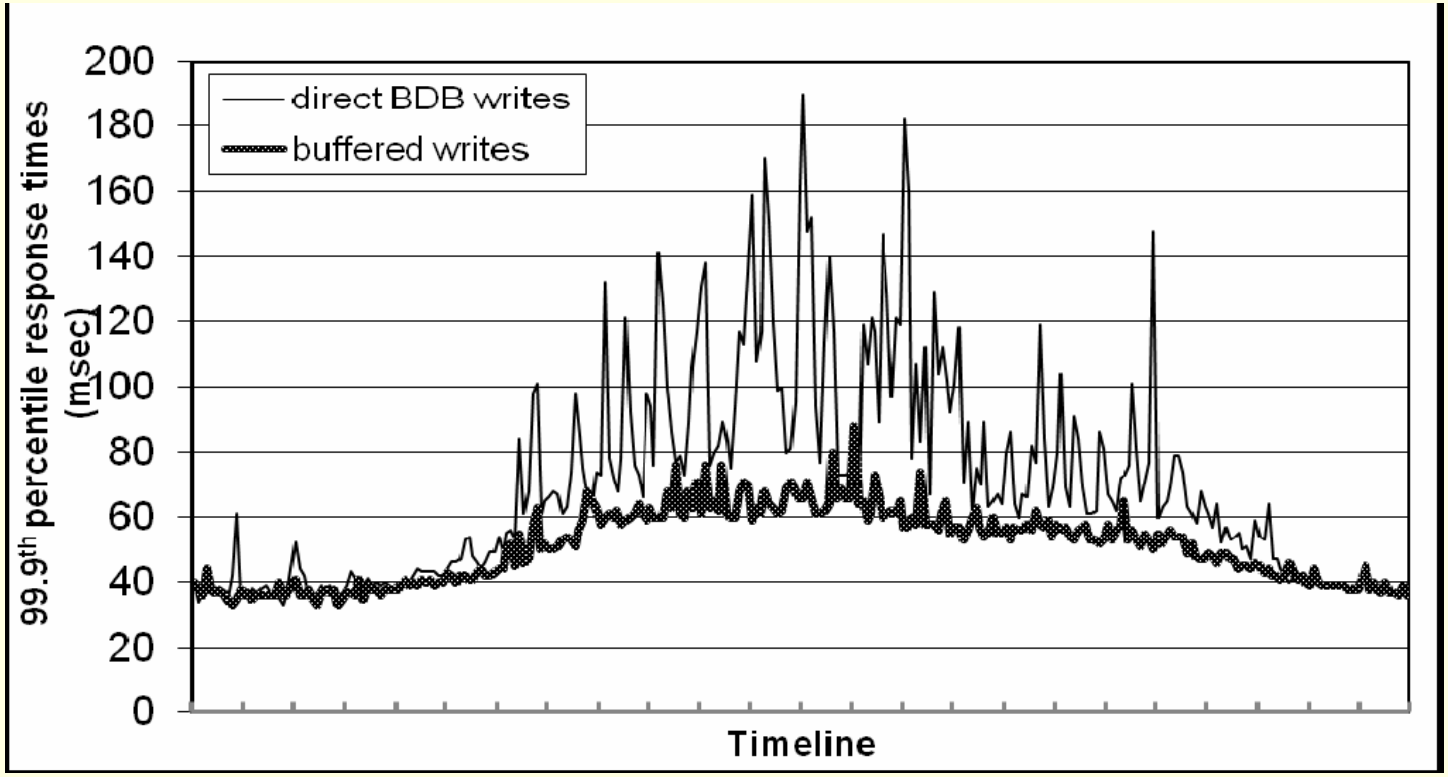
Implementation

- Java
- Local persistence component allows for different storage engines to be plugged in:
  - Berkeley Database (BDB) Transactional Data Store: object of tens of kilobytes
  - MySQL: object of > tens of kilobytes
  - BDB Java Edition, etc.

Evaluation


1000
100
10
99.9 Writes
99.9 Reads
Avg. writes
Avg Reads
Client latencies in milliseconds
(log scale)
Timeline
(hourly plot of latencies during our peak seson in Dec. 2006)

Evaluation


direct BDB writes
buffered writes
99.9th percentile response times (msec)
200
180
160
140
120
100
80
60
40
20
0
Timeline