DYNAMO

Amazon's Highly Available Key-value Store

PRESENTED BY

GAURAV VAIDYA

Some of the slides in this presentation have been taken from http://cs.nyu.edu/srg/talks/Dynamo.ppt

Introduction

Need for a highly available Distributed Data Store

- During the holiday shopping season, the service that maintains Amazon's shopping cart (Shopping Cart Service) served tens of millions requests that resulted in well over **3 million checkouts** in a single day and the service that manages session state handled hundreds of thousands of concurrently active sessions.
- Most of Amazon's services need to handle failures and inconsistencies

Motivation

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

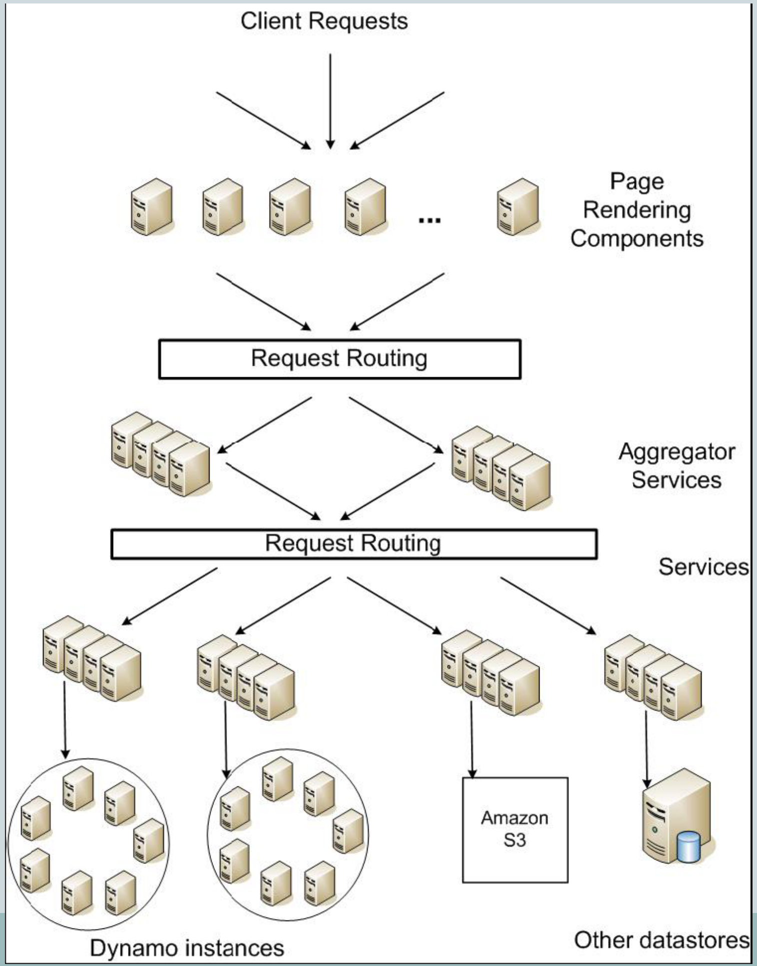
System Assumptions and Requirements

- **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 abounded 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.

Related Work

- **Peer to Peer Systems**
  - Freenet and Gnutella
  - Storage systems: Oceanstore and PAST
    - Conflict resolution for resolving updates
- **Distributed File Systems and Databases**
  - Ficus and Coda
  - Farsite
  - Google File System

Comparison

Dynamo

- (a) it is intended to store relatively small objects (size < 1M) and
- (b) key-value stores are easier to configure on a per-application basis.

Antiquity

- Uses a techniques to preserve data integrity and to ensure data consistency
- Dynamo does not focus on the problem of data integrity and security - built for a trusted environment

Bigtable

- distributed storage system for managing structured data
- allows applications to access their data using multiple attributes
- Dynamo targets applications that require only key/value access
- primary focus on high availability
- updates are not rejected even in the wake of failure.

Traditional Replicated Relational Database Systems

- focus on the problem of guaranteeing strong consistency to replicated data.
- limited in scalability and availability.
- not capable of handling network partitions

Dynamo

- Dynamo is targeted mainly at applications that need an "always writeable" data store where no updates are rejected
- Dynamo is built for an infrastructure within a single administrative domain where all nodes are assumed to be trusted
- Applications do not require support for hierarchical namespaces (a norm in many file systems) or complex relational schema (supported by traditional databases)
- Dynamo is built for latency sensitive applications that require at least 99.9% of read and write operations to be performed within a few hundred milliseconds
- zero-hop **DHT**, where each node maintains enough routing information locally to route a request to the appropriate node directly.

System Architecture

System Architecture

- System Interface
- Partitioning Algorithm
- Replication
- Data Versioning
- Execution of get () and put () operations
- Handling Failures: Hinted Handoff
- Handling permanent failures: Replica synchronization
- Membership and Failure Detection
- Adding/Removing Storage Nodes

System Interface

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

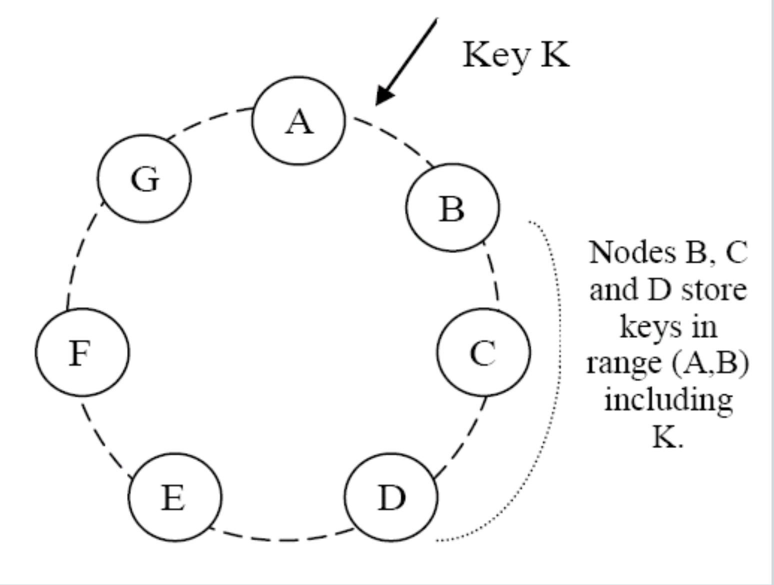
- MD5 (Key) = 128 bit identifier

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.

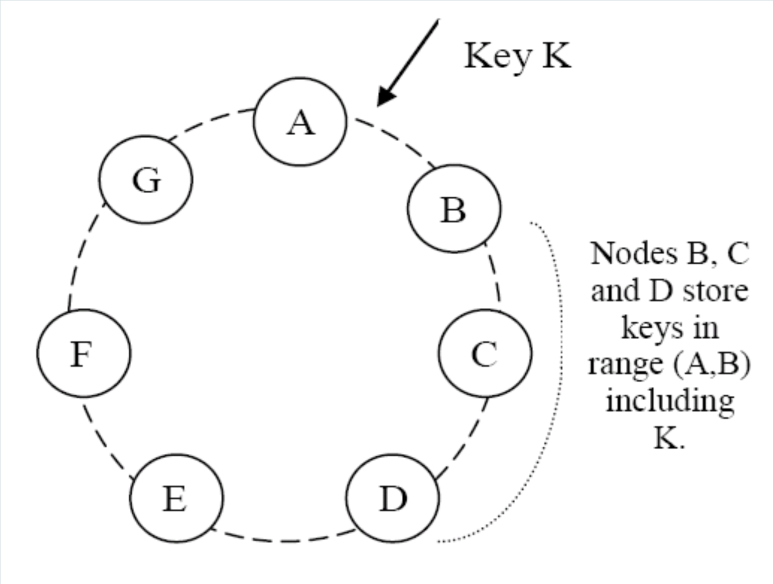
Partition Algorithm

- **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.



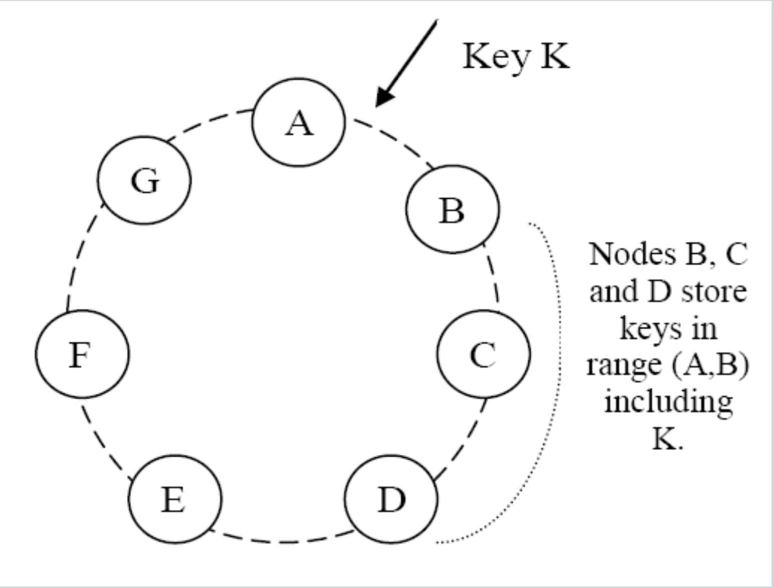
Advantages of using virtual nodes

- 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 be 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.



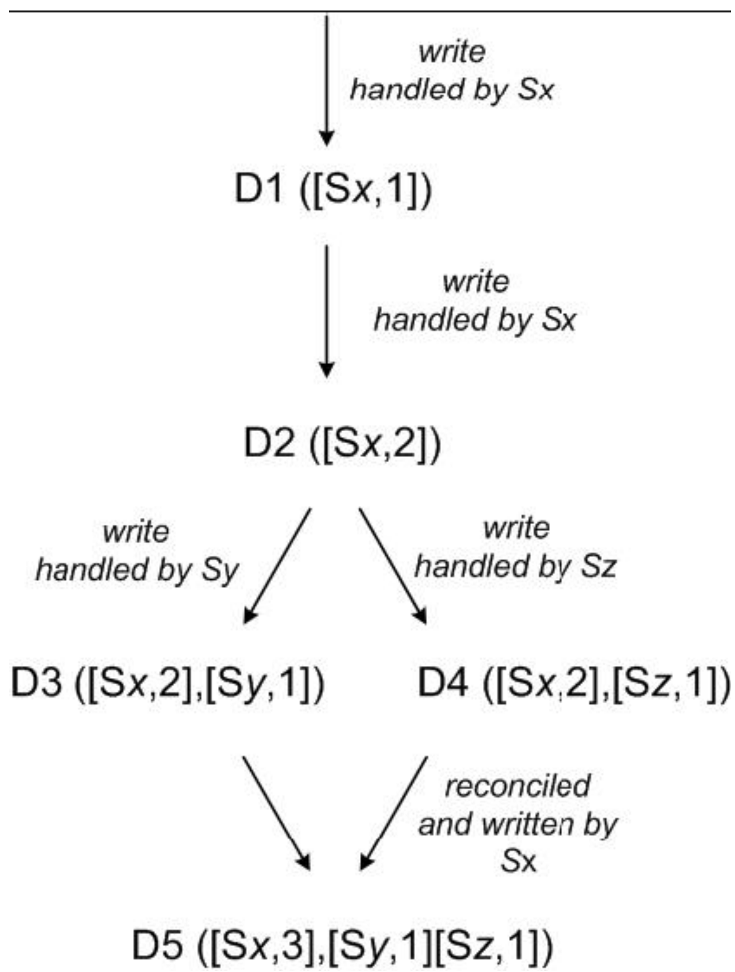
Data Versioning

- 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

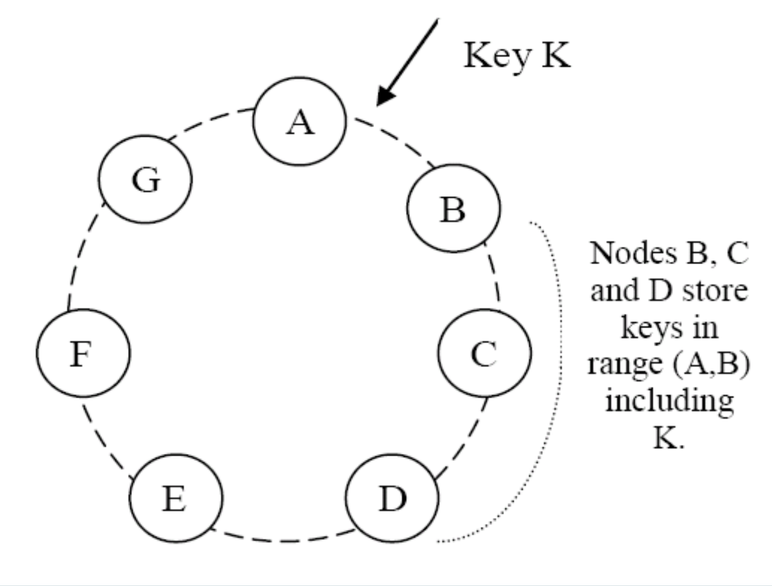
1. Route its request through a generic load balancer that will select a node based on load information.
2. Use a partition-aware client library that routes requests directly to the appropriate coordinator nodes.

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**

Membership and Failure Detection

- *Ring Membership*
  - explicit mechanism to initiate the addition and removal of nodes from a Dynamo ring
- *External Discovery*
- *Failure Detection*

Adding/Removing Storage Nodes

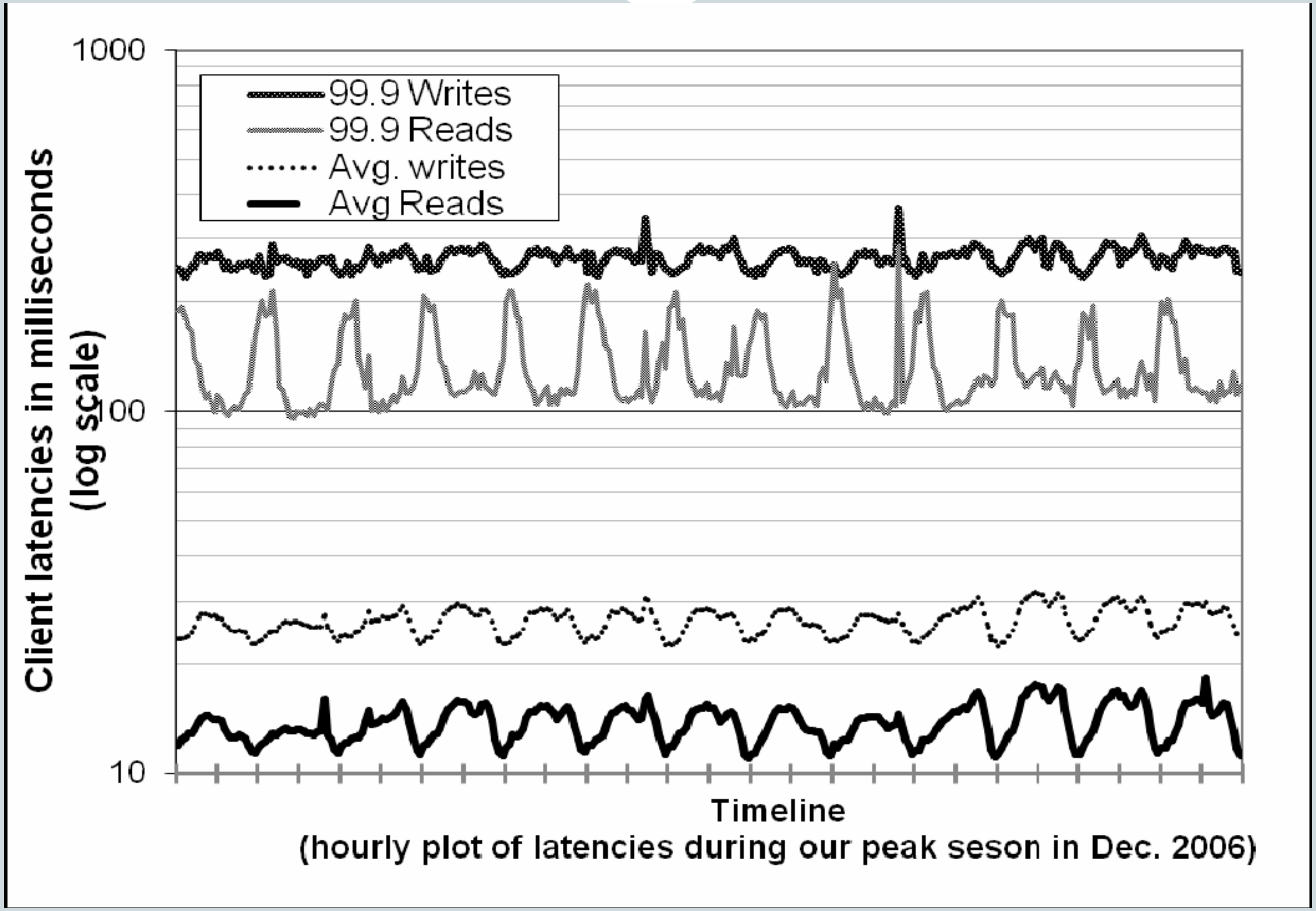
- A new node (say X) is added into the system
- It gets assigned a number of tokens (key range)
- Some existing nodes no longer have to some of their keys and these nodes transfer those keys to X
- Operational experience has shown that this approach distributes the load of key distribution uniformly across the storage nodes

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.

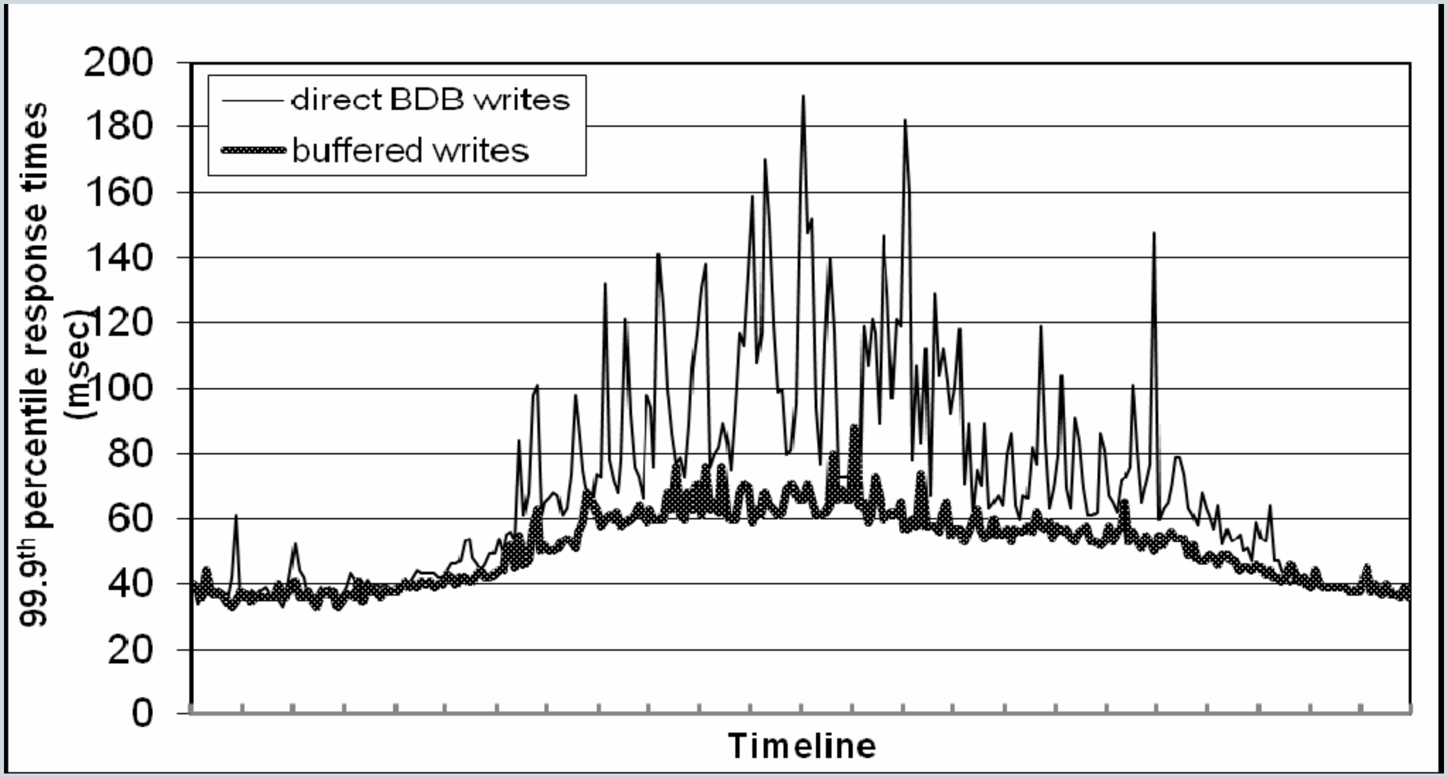
EVALUATION

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

EXPERIENCES & LESSONS LEARNED

Usage patterns

- Business logic specific reconciliation
  - Client has reconciliation logic in case of divergent versions
- Timestamp based reconciliation
  - Last write wins
- High performance read engine
  - Large number of read requests
  - R=1, W=N

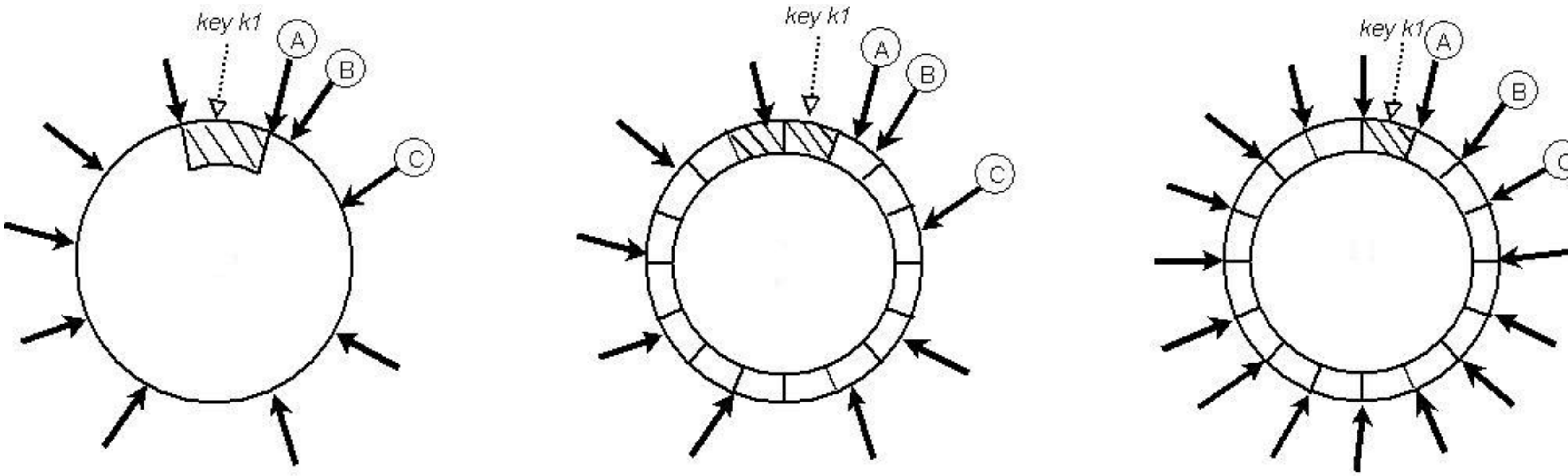
Balancing Performance and Durability

- **Typical SLA**: 99.9% of the read and write requests execute within 300ms
- Dynamo provides the ability to trade-off durability guarantees for performance
- Buffering write and read operations
- A server crash can result in missing writes that were queued up in the buffer
- One of the N replicas can perform a **durable write** without affecting performance

Ensuring Uniform Load distribution

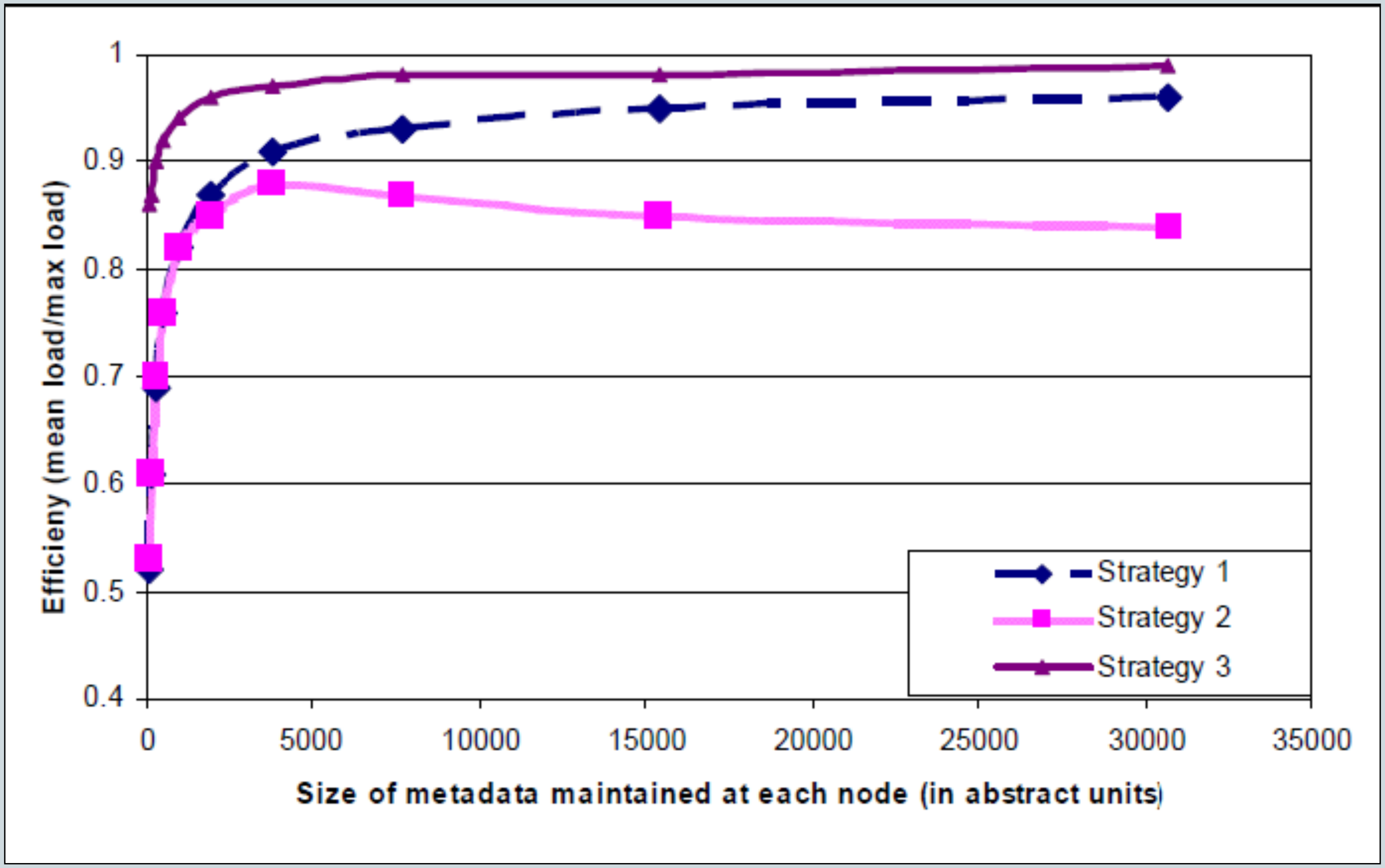
- Strategy 1:
  - T random tokens per node and partition by token value
    - Random sized hash space partitions
    - When a new node joins the system, it needs to "steal" its key ranges
- Strategy 2:
  - T random tokens per node and equal sized partitions
    - Fixed size hash space partitions, T tokens, S nodes, Q>>S*T
- Strategy 3:
  - Q/S tokens per node, equal-sized partitions
    - When a new node joins the system, it needs to "steal" its key ranges

Uniform Load Distribution Strategies



Strategy 1 Strategy 2 Strategy 3

Comparison of efficiency of different strategies



for system with 30 nodes and N=3 with equal amount of metadata maintained at each node

Divergent Versions: When and How Many?

- Metric: The number of divergent versions
- Experiment: The number of versions returned to the shopping cart service over a period of 24 hours.

Percentage of requests	No. of versions
99.94%	1
0.00057%	2
0.00047%	3
0.00009%	4

- This shows that divergent versions are created rarely.

Client-driven or Server-driven Coordination

	99.9th percentile write 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

CONCLUSIONS

Summary

- Successful responses (without timing out) for 99.9995% of its requests
- No data loss event has occurred to date
- Allows configuring (N,R,W) to tune the instance as per needs
- Exposes data consistency and reconciliation logic issues to the developers
  - Complex application logic
  - Easy to migrate pre-existing Amazon applications
- Dynamo is incrementally scalable
- Full membership model:
  - Each node actively gossips the full routing table
  - Overhead caused while scaling

Conclusions

PNUTS	Dynamo
• Hashed / Ordered tables • Hosted service • Generation based versioning • Communication through Pub / Sub YMB infrastructure (optimized for geographically separated replicas) • Partitioning into tablets • Timeline based consistency	• Key – value pairs • Internal use • Vector clocks used • Gossip based • Partitioning tokens • Eventual consistency and reconciliation