UNIVERSITY of WASHINGTON

Dynamo: Amazon's Highly Available Key-value Store

Authors:

Guiseppe Decandia, Deniz Hastorun, Madan Jampani, Gunavardhan Kakulapati, Avinash Lakshman, Alex Pilchin, Swaminathan Sivasubramanian, Peter Vosshall and Werner Vogels.

Presented by:

Yashaswi Tamta

Jugal Gandhi

Jonathan McFadden

UNIVERSITY of WASHINGTON

Outline

- Overview
- Introduction
- Background
- Related work
- Key contributions
- Implementations
- Experiences
- Summary and Advantages
- Conclusion

Overview

- Failure in large scale servers and network components is inevitable.
- Hence, reliability at massive scale is one of the biggest challenges around.

This paper presents a solution:

DynamoDB - a highly available key-value store system

Overview

- For large companies like amazon, even a slightest outage has significant financial consequences.
- Highly reliable systems provide an ‘always-on’ experience, imperative in real time applications.

Introduction

To provide robust solutions for:

Load Balancing

Failure detection

Job scheduling

Overload handling

Failure recovery

Config management

Request routing

Request marshalling

Replica synchronization

Introduction

- In DynamoDB data is partitioned and replicated using consistent hashing, and consistency is facilitated by object versioning.
- The consistency among replicas during updates is maintained by a quorum-like technique and a decentralized replica synchronization protocol.

Introduction 1

- *Partitioning:*

  Dynamo's partitioning scheme completely relies on consistent hashing.

- *Replication:*

  Each data item is replicated on N hosts, where N is a parameter configured 'per instance'

Introduction 2

- *Data versioning:*

  Treats the result of each modification as a new and immutable version of the data.

- *Handling failures:*

  Dynamo does this by, using 'sloppy quorum' and 'hinted handoff'.

Background

- Traditional system use relational databases which retrieve and store state using a primary key.
- This excess functionality requires expensive hardware and skilled personnel for its operation.

Related work

- *Peer to Peer systems:*

  Here a search query is flooded through the network to find as many peers as possible that share the data.

- *Distributed file systems and Database:*

  Utilizes distributed stores of data items or chunks of data items across loosely coupled systems.

Related work

Dynamo differs from the previous works:

- Dynamo is targeted at applications that need an 'always writable' data store
- Dynamo is built for an infrastructure within a single admin domain
- Dynamo is built for latency sensitive applications

Related work

- ACID transactions are not supported compared to a less familiar but fully ACID compliant ConcourseDB [1]

- There's no multi-key index: you can either have one hash key or a hash-key + range key combination

Key contributions

- 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

  - Values of R and W impact object availability, durability, and consistency.

- Despite some design limitations dynamoDB offers zero maintenance and effortless scaling

Implementations - 1

Three main software components:

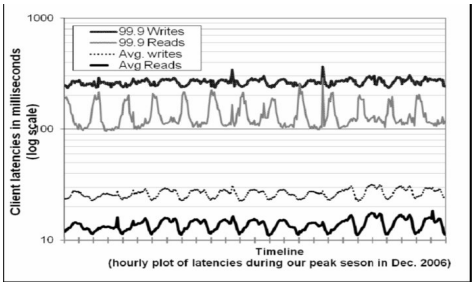
- Local persistent engine
  - plug-in architecture supports different storage engines
  - storage engine chosen based on application' object size distribution
  - BerkeleyDB: objects in the order of tens of kilobytes
  - MySQL: larger objects
- Request coordination
  - built on top of an event-driven messaging infrastructure
  - coordinator executes the read and write operations in behalf of client
- Membership and Failure Detection

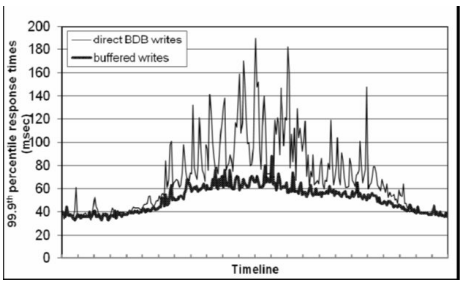
Implementations - 2

- Dynamo is used by a number of services with different usage patterns:
  - Business logic specific reconciliation: Many-node replication, with client handling reconciliation. e.g. shopping cart logic.
  - Timestamp based reconciliation: “Last write wins”. e.g. customer session service.
  - High-performance read engine: Services with a high read-request rate, small number of updates. e.g. product catalogs.
- Value in allowing applications to tune R and W (affecting consistency, durability, availability)
  - Common: (N:3, R:2, W:2)
  - High-performance read: (N:3, R:1, W:3)
  - High-performance write: (N:3, R:3, W:1)

Experiences - 1

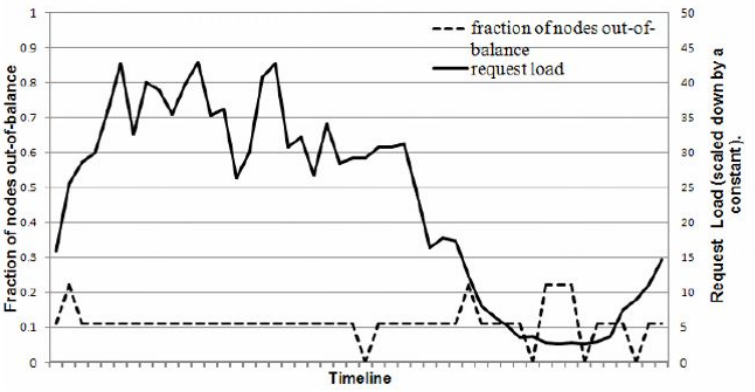
- Average and 99.9th percentile latencies
  - latencies exhibit a diurnal pattern
  - write latencies higher than read because they results in disk access
  - affected by several factors such as variability in request load, object sizes, and locality patterns
  - To achieve higher-performance on writes, an optional writer thread can be used to buffer writes.





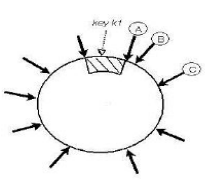
Experiences - 2

- Fraction of Out-of-Balance Nodes

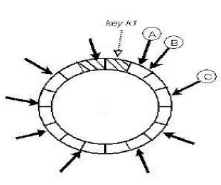


Experiences - 3

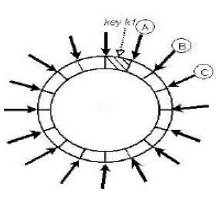
- Load Distribution Strategies



Strategy 1



Strategy 2

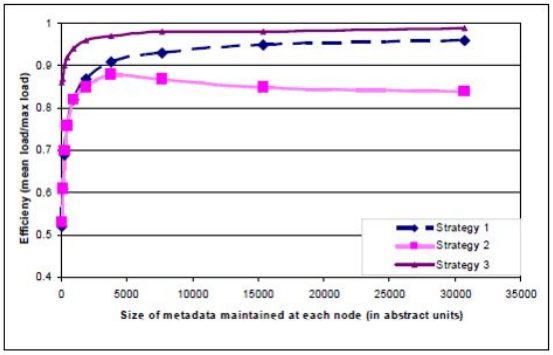


Strategy 3

T random tokens per node and partition by token value	T random tokens per node and equal sized partitions	Q/S tokens per node, equal-sized partitions
• New nodes need to "steal" key ranges from exiting nodes. • Data partitioning and Data placement schemes intertwined.	• Decoupling of partitioning and partition placement • Enabling the possibility of changing the placement scheme at runtime	• When a node leaves, its tokens are randomly distributed to remaining. • Similar strategy followed when a new node is added.

Experiences - 4

- Load Balancing Efficiency



Experiences - 5

- Coordination
  - There are two ways of locating a node to service a request:
    - A load-balancer can determine the node for a given key/ request. The burden is on the load-balancer/system
    - The client can periodically sample a random node (every 10 seconds), grab its membership state and use that to query nodes directly.

Table 2: Performance of client-driven and server-driven coordination approaches.

	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

Summary and Advantages

Problem	Technique	Advantages
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.

Conclusions

- Dynamo is a highly available and scalable data store, that provides customers with the ability to customize their storage system to meet their desired performance, durability and consistency SLAs.
- Dynamo allows service owners to customize their storage system by allowing them to tune the parameters N, R,and W.
- It has been proven as a durable, robust and scalable solution to delivering at massive-scale. Its success demonstrates that “an eventual- consistent storage system can be a building block for highly- available applications“.

Questions

???

THANK YOU