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

amazon.com

October 2007
SOSP '07: Proceedings of twenty-first ACM SIGOPS symposium on Operating systems principles
Publisher: ACM

Apr. 1, 2009
OUTLINE

• INTRODUCTION
• BACKGROUND
• SYSTEM ARCHITECTURE
• IMPLEMENTATION
• EXPERIENCES
• CONCLUSIONS
INTRODUCTION

- Amazon uses a highly decentralized, loosely coupled, service oriented architecture consisting of hundreds of services.
- There are always a small but significant number of server and network components that are failing at any time.
- Dynamo is another highly available and scalable distributed data store like S3.
BACKGROUND

- System Assumptions and Requirements:
  - Query Model, ACID, Efficiency, Others
- Service Level Agreements (SLA)
- Design Considerations:
  - Conflict resolution
  - Incremental scalability
  - Symmetry
  - Decentralization
  - Heterogeneity
SYSTEM ARCHITECTURE

- System Interface
- Partitioning Algorithm
- Replication and Data Versioning
- Execution of get() and put() operations
- Handling Failures
- Membership and Failure Detection
- Adding/Removing Storage Nodes
System Interface

- Dynamo stores objects associated with a key through a simple interface:
  - `get(key)`
  - `put(key, context, object)`

- The `context` encodes system metadata about the object that is opaque to the caller and includes information.
Partitioning Algorithm

- Dynamo’s partitioning scheme relies on consistent hashing to distribute the load across multiple storage hosts.
- The output range of a hash function is treated as a fixed circular space or “ring”.
- Each node in the system is assigned to a random value within this space which represents its “position” on the ring.
SYSTEM ARCHITECTURE

Partitioning Algorithm

Hash Function

Virtual Nodes

Data

Coordinator

Consistent Hashing in Dynamo
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 which is in charge of the replication of the data items
- The list of nodes that is responsible for storing a particular key called the preference list
SYSTEM ARCHITECTURE

Replication

Replication of keys in Dynamo

N = 3

Coordinator

Data

Data

Data
Data Versioning

• Dynamo provides eventual consistency, which allows for updates to be propagated to all replicas asynchronously.

• Dynamo uses *vector clocks* in order to capture causality between different versions of the same object.

• The client needs to perform the reconciliation if the system can’t...
A client writes a new object

The client updates the object

Update the object by different servers and different clients

The client performs the reconciliation

Version evolution of an object over time
Execution of `get()` and `put()`

- Any storage node in Dynamo is eligible to receive client `get` and `put` operations for any key.

- There are two strategies that a client can use to select a node:
  - route its request through a load balancer
  - use a partition-aware client library
Execution of get() and put()  

- To maintain consistency among replicas, Dynamo uses a consistency protocol which has two key configurable values:
  
  - **R**: the minimum # of nodes that must participate in a successful read operation
  
  - **W**: the minimum # of nodes that must participate in a successful write operation
Handling Failures: Hinted Handoff

- All read and write operations are performed on the first N *healthy* nodes from the preference list which may not always be the first N nodes encountered while walking the consistency hashing ring.

- Ensures that the read and write operations are not failed due to temporary node or network failures.
SYSTEM ARCHITECTURE

Handling Failures: Hinted Handoff

Hinted Handoff during a Write Operation

Detect A periodically

Data with a hint in its metadata
Handling Permanent Failures: Replica synchronization

- To detect the inconsistencies between replicas faster and to minimize the amount of transferred data

- *Merkle trees*: a hash tree where leaves are hashes of the values of the individual keys. Parent nodes higher in the tree are hashes of their respective children
SYSTEM ARCHITECTURE

Membership and Failure Detection

- Ring Membership
  - Explicit mechanism to initiate the addition and removal of nodes from ring
- External Discovery
  - To prevent logical partitions, some Dynamo nodes play the role seeds
- Failure Detection
  - Avoid attempts to communicate with unreachable peers
SYSTEM ARCHITECTURE

Adding/Removing Storage Nodes

A bootstrapping scenario when adding a node

Transfer keys
IMPLEMENTATION

• Each storage node in Dynamo has three main software components:
  • request coordination
    • using Java NIO channels
  • membership and failure detection
  • local persistence engine
    • allows for different storage engines
EXPERIENCES

• Three main patterns used in Dynamo:
  • Business logic specific reconciliation
  • Timestamp based reconciliation
  • High performance read engine

• The common (N, R, W) configuration used by several instances of Dynamo is (3, 2, 2)
EXPERIENCES

Balancing Performance and Durability

Average and 99.9 percentiles of latencies for read and write requests during Amazon’s peak request season of December 2006

(hourly plot of latencies during our peak seson in Dec. 2006)
EXPERIENCES

Balancing Performance and Durability

Comparison of performance of 99.9th percentile latencies for buffered vs. non-buffered writes over a period of 24 hours
EXPERIENCES

Ensuring Uniform Load Distribution

Fraction of nodes that are out-of-balance and their corresponding request load
Ensuring Uniform Load Distribution

EXPERIENCES

Strategy 1
- T random tokens per node and partition by token value

Strategy 2
- T random tokens per node and equal sized partitions

Strategy 3
- Q/S tokens per node, equal-sized partitions

Partitioning and placement of keys in the three strategies. A, B, and C depict the three unique nodes that form the preference list for the key k1 on the consistent hashing ring (N=3)
EXPERIENCES

Ensuring Uniform Load Distribution

Comparison of the load distribution 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?

• Occurs when the system is facing failure scenarios or handling a large number of concurrent writes to a single data

• In experiment, the number of versions returned to the shopping cart service:
  • 99.94% of requests saw one version
  • 0.00057% of requests saw 2 versions
  • 0.00047% of requests saw 3 versions
  • 0.00009% of requests saw 4 versions
**EXPERIENCES**

Client-driven or Server-driven Coordination

<table>
<thead>
<tr>
<th></th>
<th>99.9th percentile read latency (ms)</th>
<th>99.9th percentile write latency (ms)</th>
<th>Average read latency (ms)</th>
<th>Average write latency (ms)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Server-driven</td>
<td>68.9</td>
<td>68.5</td>
<td>3.9</td>
<td>4.02</td>
</tr>
<tr>
<td>Client-driven</td>
<td>30.4</td>
<td>30.4</td>
<td>1.55</td>
<td>1.9</td>
</tr>
</tbody>
</table>

Performance of client-driven and server-driven coordination approaches
CONCLUSIONS

• The production use of Dynamo for the past year demonstrates that decentralized techniques can be combined to provide a single highly-available system

• Its success in one of the most challenging application environments shows that an eventual consistent storage system can be a building block for highly-available applications