Dynamo: Amazon’s Highly Available Key-Value Store

Amazon.com, SOSP 2007

Presented by: Marc Leef
Problem

• Lack of a distributed data store that:
  • Prioritizes availability and performance over consistency
  • Allows for application-specific control of performance, durability, and consistency
  • Can operate at Amazon scale (3 million checkouts/day, hundreds of thousands of concurrent users)
Key Ideas

• Always writeable
• Incrementally scalable
• Node equality
• Application-level conflict resolution
• Configurable performance and consistency tuning
Architecture

• Partitioning
• Replication
• Versioning
• Operations
• Failures
Partitioning: Consistent Hashing

- Hash function output range is ring
- Hash of key is a location in the ring
- Walk clockwise to find containing node
- Problems:
  - Non-uniform data/load distribution
  - Cannot leverage more powerful nodes

Key: “COS 518”
Value: “Best class I’ve ever taken”

Hash(“COS 518”) = 4
Partitioning: Virtual Nodes

- Nodes assigned multiple points
- Advantages:
  - Efficient redistribution of load
  - More power = more points on circle
Replication

- N represents degree of replication
- Responsible through Nth predecessor
- Each node contains preference list
- Preference list:
  - Contains nodes responsible for given key
  - Contains only physical nodes
  - More on this later
Versioning

- Object updates are propagated asynchronously
- Eventual consistency
- Object versions are tracked through vector clocks
- Vector clock:
  - List of (node, counter) pairs
  - One per object version
- Version conflicts are resolved by applications
Versioning: Vector Clock Evolution Example

Figure 3: Version evolution of an object over time
Operations

• Simple get/put interface
• Two routing strategies for client requests:
  • Load Balancer: No Dynamo specific code in application
  • Partition-Aware Library: Lower latency
• Configurable values:
  • R: Minimum # of nodes that must participate in read to be considered successful
  • W: Minimum # of nodes that must participate in write to be considered successful
• R + W > N similar to quorum system
• R + W < N provides better latency

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

<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>
Failures: Hinted Handoff/Sloppy Quroum

• If a node becomes unreachable, the replicas its responsible for will be diverted to another node

• Message metadata contains hint of identity of the originally intended recipient

• Replicas are stored in separate database and restored once original recipient becomes reachable
Failures: Replica Synchronization

- Merkle trees are used to reconcile discrepancies in data.
- Separate Merkle tree kept for each key range.
- Trees are compared via traversal to identify out of sync keys.
Failures: Membership and Failure Detection

• Nodes “gossip” to reconcile differences in preference lists, membership, and health status

• Nodes contact one another at random every second to perform this reconciliation step

Preference List:
• Keys 1-100:
  1. A
  2. B
  3. C
• Keys 101-200:
  1. D
  2. E
  3. A
Table 1: Summary of techniques used in Dynamo and their advantages.

<table>
<thead>
<tr>
<th>Problem</th>
<th>Technique</th>
<th>Advantage</th>
</tr>
</thead>
<tbody>
<tr>
<td>Partitioning</td>
<td>Consistent Hashing</td>
<td>Incremental Scalability</td>
</tr>
<tr>
<td>High Availability for writes</td>
<td>Vector clocks with reconciliation during reads</td>
<td>Version size is decoupled from update rates.</td>
</tr>
<tr>
<td>Handling temporary failures</td>
<td>Sloppy Quorum and hinted handoff</td>
<td>Provides high availability and durability guarantee when some of the replicas are not available.</td>
</tr>
<tr>
<td>Recovering from permanent failures</td>
<td>Anti-entropy using Merkle trees</td>
<td>Synchronizes divergent replicas in the background.</td>
</tr>
<tr>
<td>Membership and failure detection</td>
<td>Gossip-based membership protocol and failure detection.</td>
<td>Preserves symmetry and avoids having a centralized registry for storing membership and node liveness information.</td>
</tr>
</tbody>
</table>
Performance: Read/Write

Dynamo performance during critical holiday period
Performance: Buffered Writes

Object updates stored in buffer, routinely written to disk by writer thread
**Partitioning Strategies**

**Strategy 1**
Random tokens

**Strategy 2**
Random tokens + Equal sized partitions

**Strategy 3**
Q/S tokens + Equal sized partitions
Partitioning Strategies: Results
Closing Thoughts

• Strengths:
  • Configurable
  • Performant
  • Available

• Weaknesses:
  • Too much work for applications?
  • Node symmetry scalable?
  • Paper clarity