Amazon Dynamo

distributed key-value storage

Michal Oniszczuk

October 10, 2012
Vast Distributed System
- Tens of millions of customers
- Tens of thousands of servers
- Failure is a normal case

Outage means
- Lost Customer Trust
- Financial loses

Goal: great customer experience
- Always Available
- Fast
- Reliable
- Scalable
SOA - *(Service Oriented Architecture)*
- Simple put(key, object) and get(key) operations
- Targets small objects (< 1MB)
- No operations on multiple data
- Relational databases are not needed, they do not scale
ACID Properties

(Atomicity, Consistency, Isolation, Durability)

- Properties that guarantee that database transactions are processed reliably
- Atomicity (“A”) does not apply
- Weaker Consistency (“C”)
- No Isolation (“I”)
- Dynamo is configurable per application. Tradeoffs between: Durability (“D”), Availability, Performance, Cost Efficiency
**Example:** *Service A is demanding key-value storage for objects that in 99.9% of cases can be read in 300 ms*

Just one number is describing the agreement - the latency of 99.9% percent of cases. Each application in Amazon’s architecture must obey the performance contract.

In Amazon it turns out that more sophisticated ways of describing SLAs such as mean, median, average and variance are not good enough.
- No security guaranteed
- Each service runs its own instance of Dynamo
- Design targets hundreds of hosts
Usually: strong consistency

Data is unavailable until all storage replicas (copies) are the same.
Other way: optimistic replication

Changes are allowed to propagate. This causes conflicts.

When to resolve conflicts?

- Each write must be successful.
- The conflict resolution is pushed to read operations.

Who performs the process of conflict resolution?

- the client application
- data store itself
Design principles

- Incremental scalability
- Symmetry
- Decentralization
- Heterogeneity
- \textit{put}({\textit{key}, \textit{context}, \textit{object}}) - stores replicas of \textit{object} under the \textit{key}. \textit{context} is used to store the metadata used by Dynamo to resolve conflicting versions.

- \textit{get}({\textit{key}}) - returns the object and its context. It may return multiple results.
Consistent Hashing

- Each key is assigned to *coordinator node* (first clockwise encountered node on the ring from key’s position)
- Virtual nodes (tokens)
Coordinator stores object both locally and also at $N - 1$ clockwise successor nodes in the ring.
- Vector clocks - a list of tuples \([\text{node}, \text{counter}]\)
- Syntactic reconciliation
- Semantic reconciliation

```
write
handled by Sx

D1 ([Sx,1])

write
handled by Sx

D2 ([Sx,2])

write
dhandled by Sy

write
dhandled by Sz

D3 ([Sx,2],[Sy,1])

D4 ([Sx,2],[Sz,1])

reconciled
and written by
Sx

D5 ([Sx,3],[Sy,1][Sz,1])
```
Key properties

- get and put are invoked over HTTP (Amazon’s internal request processing framework)
- it is possible to use generic load balancer - then *internal forwarding*
- or else the client may use the library that routes requests directly to the appropriate coordinator nodes
Two configureable values $R$ and $W$

- $W$ - minimum number of nodes that must participate in successful *write* operation
- $R$ - minimum number of nodes that must participate in successful *read* operation
Hinted handoff

- concept of *healthy nodes*
- connected with *only distinct physical nodes* approach
## Summary of techniques used in Dynamo.

<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>
- \( N \) - number of tokens that are responsible for storing data from particular range in hash space
- \( W \) - minimum number of nodes that must participate in successful \textit{write} operation
- \( R \) - minimum number of nodes that must participate in successful \textit{read} operation

The most common configuration of this values in production environment was \((N, W, R) = (3, 2, 2)\).

For the massively read storage it could be set to \((N, W, R) = (3, 1, 1)\).
- Berkeley DB
- MySQL
- tailored *in-memory buffer* with persistent backing store
99.9995% of applications’ calls had been returned successfully without timing out

**no data loss** have occurred during measurements

if client’s application perform using libraries some of the request coordination it reduces latencies by 50%

it turns out that: 99.94% of requests saw exactly one version of object, 0.00057% saw two versions, ...
Further problems

Balancing Performance & Durability Ensuring Uniform Load Distribution
(hourly plot of latencies during our peak season in Dec. 2006)
- P2P systems (Gnutella)
- Distributed File Systems and Databases (GFS, BigTable)
**Dynamo**

- highly available and scalable
- configurable
Some slides are used from presentation by Marcin Walas.

Based on the article Dynamo: Amazons Highly Available Key-value Store by Giuseppe DeCandia, Deniz Hastorun, Madan Jampani, Gunavardhan Kakulapati, Avinash Lakshman, Alex Pilchin, Swaminathan Sivasubramanian, Peter Vosshall and Werner Vogels.