Amazon Dynamo
Amazon’s Architecture

Client Requests

Page Rendering Components

Request Routing

Aggregator Services

Request Routing

Services

Dynamo instances

Amazon S3

Other datastores
Dynamo

- The platform for Amazon's e-commerce services: shopping chart, best seller list, produce catalog, promotional items etc.

- A highly available, distributed key-value storage system: put() & get() interfaces

- Aims to provide "always on" guarantee at the cost of losing some consistency
Design Considerations

- **Incremental scalability**
  - Minimal management overhead

- **Symmetry**
  - For ease of maintenance, no master/slave nodes

- **Decentralized control**
  - Centralized approach leads to outages

- **Heterogeneity**
  - Exploit capabilities of different nodes
CAP Theorem

Consistency
single up-to-date copy

Availability
for updates

Partition Tolerance

Pick 2 out of 3
What Does CAP Really Mean?

“No system where P is possible can at all times guarantee both C and A”

if your network is highly reliable (and fast), so that P is extremely rare, you can aim for both C and A

Google Spanner
Requirements

Reliability:
customers should be able to edit their shopping cart even when there are:
• disk failures
• network failures
• tornados!

Efficiency:
latenity measurement is done at 99.9th percentile of the distribution
ACID Properties

- Atomicity
- Consistency
- Isolation
- Durability
Requirements

Query model:
only simple read/write to small data (less than 1MB), uniquely identified by a key

ACID properties:
• atomicity: important!
• consistency: weak is sufficient
• isolation: not at all
• durability: important!
Consistency

• **BASE**
  • Basically Available, Soft State, Eventually Consistent

• **Always writable**
  • Can always write to shopping cart
  • Conflict resolution on reads

• **Application-driven conflict resolution**
  • Merge conflicting shopping carts to never lose “Add Cart”
  • push inconsistencies to Customer Service
Requirements

Service Level Agreement (SLA):

- client and server agree on several characteristics related to the system
  - e.g. 300 ms response time for peak load of 500 requests/second

- putting together a single webpage can require responses from 150 services for typical request
# Techniques Used by 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>
Consistent Hashing

Issues

- Unbalanced load distribution
- Nodes are heterogenous, uneven resources per node
Virtual Nodes

Hash key and place on ring

Key k

Next node clockwise is in charge

Each node has multiple points in ring: load more likely to even out

Different number of virtual nodes (depending on node’s resources)
Node Joining or Leaving
What If
Replication

Coordinator

Replicates data in a list of N nodes

Key k

N is per key

Part of longer preference list

Mindful of virtual nodes

A

G

B

F

C

E

D
<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>
Quorums

put():
- coordinator writes new data locally
- sends to next n-1 nodes
- when w-1 respond, considered successful

get():
- coordinator requests next n-1 nodes
- when r-1 respond, considered successful

• r + w > n
Data Versioning

• A put() call may return successfully to caller before the update has been applied to all the replications
  • If latest object version unavailable, apply put to earlier version, merge later

• A get() call may return many versions of the same object

• An object may have distinct version sub-histories that need to be reconciled
Delayed Reconciliation

The goal: "add to cart" operation should never be forgotten or rejected

- When a customer wants to add an item to (or remove from) a shopping cart and the latest version is not available, the item is added to (or removed from) the older version
- The divergent versions are reconciled later
Vector Clocks in Dynamo

- List of (node, counter) pairs
  Counters are logical integer timestamps assigned by coordinator

- Each object version has one vector clock

- Used to determine whether a version is subsumed

- Stored in “context” in get(), put()
Vector Clocks in Dynamo

Size of vector clock can get arbitrarily long
- bounded by N in practice
- drop oldest one when beyond certain threshold

Reconciliation
- Easy if values are causally ordered
- Otherwise, applications will handle
Reconciliation (Summary)

- An "add to cart" operation will never be lost,
- but removed items may reappear
Reconciliation in Practice

• In one 24 hour period, 99.94% of requests saw exactly one version

• Divergent version usually not caused by failure but concurrent writers....(rarely human beings, usually automated client programs)
## Techniques Used by 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></td>
<td>Vector clocks with reconciliation during reads</td>
<td>Version size is decoupled from update rates</td>
</tr>
<tr>
<td>High Availability for writes</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>Handling temporary failures</td>
<td>Anti-entropy using Merkle trees</td>
<td>Synchronizes divergent replicas in the background</td>
</tr>
<tr>
<td>Recovering from permanent</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>
<tr>
<td>failures</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Membership and failure detection</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Quasi-Quorums

- `get()` and `put()` driven by two parameters
  - `R`: minimum number of replicas to read from
  - `W`: minimum number of replicas to write to
- if `R+W > N`, we have a quorum system!
  ... but latency dictated by slowest replica

- What if we want to execute a `put()` but the number of available replicas in the next node is less than `w-1`?
Sloppy Quorum and Hinted Handoff

- Use next N healthy nodes for read/write
- Data tagged with the node it should go to (notified later if temporarily unavailable)
- Transfer the data to the node when it becomes available
- Always writeable!
<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>
Handling Permanent Failures

Replica synchronization:
- Node synchronizes with another node
- A Merkle Tree is used to detect inconsistency and minimize the data that needs to be transferred
  - leaves are hash of objects
  - parents are hash of children
Membership and Failure Detection

- Gossip-based protocol propagates membership change and maintains an eventually consistent view

- Seeds are used to prevent paritioned ring

- Use timeout to for failure discovery