Dynamo

Haibo Chen

Ack: Dynamo slides adapted from www.slideworld.com created by paper authors
Motivation

Even the slightest outage has significant financial consequences and impacts customer trust.

The platform is implemented on top of an infrastructure of tens of thousands of servers and network components located in many datacenters around the world.

Persistent state is managed in the face of these failures - drives the reliability and scalability of the software systems.
Build a distributed storage system:

- Scale
- Simple: key-value
- Highly available (sacrifice consistency)
- Guarantee Service Level Agreements (SLA)
System Assumptions and Requirements

Query Model

Simple read and write operations to a data item that is uniquely identified by a key.

Most of Amazon’s services can work with this simple query model and do not need any relational schema.

targeted applications - store objects that are relatively small (usually less than 1 MB)
Experience at Amazon has shown that data stores that provide ACID guarantees tend to have poor availability.

Dynamo targets applications that operate with weaker consistency (the “C” in ACID) if this results in high availability.
Efficiency

Latency requirements which are in general measured at the 99.9th percentile of the distribution

Average performance is not enough
Other Assumptions

Operation environment is assumed to be non-hostile

There are no security related requirements such as authentication and authorization
Service Level Agreements (SLA)

Application can deliver its functionality in bounded time

Every dependency in the platform needs to deliver its functionality with even tighter bounds

Example

service guaranteeing that it will provide a response within 300ms for 99.9% of its requests for a peak client load of 500 requests per second
Service-oriented architecture
Design Consideration

Sacrifice strong consistency for availability

Conflict resolution is executed during read instead of write, i.e. “always writeable”
Design Consideration (Cont’d)

Incremental scalability

Symmetry
Every node in Dynamo should have the same set of responsibilities as its peers

Decentralization
In the past, centralized control has resulted in outages and the goal is to avoid it as much as possible

Heterogeneity
This is essential in adding new nodes with higher capacity without having to upgrade all hosts at once
System architecture

Partitioning

High Availability for writes

Handling temporary failures

Recovering from permanent failures

Membership and failure detection
Partition Algorithm

Consistent hashing: the output range of a hash function is treated as a fixed circular space or “ring”.

“Virtual Nodes”: Each node can be responsible for more than one virtual node.
Advantages of using virtual nodes

If a node becomes unavailable, the load handled by this node is evenly dispersed across the remaining available nodes.

When a node becomes available again, or a new node is added to the system

- the newly available node accepts a roughly equivalent amount of load from each of the other available nodes.

The number of virtual nodes that a node is responsible can decided based on its capacity, accounting for heterogeneity in the physical infrastructure.
Replication

Each data item is replicated at N hosts.

“preference list”: The list of nodes that is responsible for storing a particular key.
Data Versioning

A `put()` call may return to its caller before the update has been applied at **all** the replicas.

A `get()` call may return **many** versions of the same object.

**Key Challenge:** distinct version sub-histories - need to be reconciled.

**Solution:** uses **vector clocks** in order to capture causality between different versions of the same object.
Vector Clock

A vector clock is a list of (node, counter) pairs

Every version of every object is associated with one vector clock

If the counters on the first object’s clock are less-than-or-equal to all of the nodes in the second clock

then the first is an ancestor of the second and can be forgotten
Vector clock example

D1 ([Sx,1])
  \[\text{write \ handled by Sx}\]
  \[\text{handled by Sx}\]

D2 ([Sx,2])

\[\text{write \ handled by Sy}\]
\[\text{write \ handled by Sz}\]

D3 ([Sx,2],[Sy,1])
D4 ([Sx,2],[Sz,1])

\[\text{reconciled \ and \ written by Sx}\]

D5 ([Sx,3],[Sy,1],[Sz,1])
Vector clock

In case of network partitions or multiple server failures, write requests may be handled by nodes that are not in the top N nodes in the preference list causing the size of vector clock to grow.

Dynamo stores a timestamp that indicates the last time the node updated the data item.

When the number of (node, counter) pairs in the vector clock reaches a threshold (say 10), the oldest pair is removed from the clock.
Execution of get () and put () ops

Two strategies to select a node:

- Route its request through a generic load balancer that will select a node based on load information.
- Use a partition-aware client library that routes requests directly to the appropriate coordinator nodes.
Execution of get () and put () ops (Cont’d)

The advantage of the first approach is that the client does not have to link any code specific to Dynamo in its application.

The second strategy can achieve lower latency because it skips a potential forwarding step.

First node routed to is “coordinator” node
- Generates vector clock for put and gives data to N highest healthy nodes
- Queries N highest nodes for all versions, returned all versions found
Temporary Failures
Sloppy Quorum

R/W is the minimum number of nodes that must participate in a successful read/write operation.

Setting R + W > N yields a quorum-like system.

In this model, the latency of a get (or put) operation is dictated by the slowest of the R (or W) replicas.

For this reason, R and W are usually configured to be less than N, to provide better latency.
Hinted handoff

Assume $N = 3$.

When A is temporarily down or unreachable during a write, send replica to D.

D is hinted that the replica belongs to A and it will deliver to A when A is recovered.

Again: “always writeable”
Replica synchronization

Merkle tree:

- a hash tree where leaves are hashes of the values of individual keys.

- Parent nodes higher in the tree are hashes of their respective children.
Advantage of Merkle tree:

Each branch of the tree can be checked independently without requiring nodes to download the entire tree.

Help in reducing the amount of data that needs to be transferred while checking for inconsistencies among replicas.
### 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</td>
<td>Version size is decoupled from update rates.</td>
</tr>
<tr>
<td></td>
<td>during reads</td>
<td></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</td>
</tr>
<tr>
<td></td>
<td></td>
<td>replicas are not available.</td>
</tr>
<tr>
<td>Recovering from permanent</td>
<td>Anti-entropy using Merkle trees</td>
<td>Synchronizes divergent replicas in the background.</td>
</tr>
<tr>
<td>failures</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Membership and failure</td>
<td>Gossip-based membership protocol</td>
<td>Preserves symmetry and avoids having a centralized registry for storing</td>
</tr>
<tr>
<td>detection</td>
<td>and failure detection.</td>
<td>membership and node liveness information.</td>
</tr>
</tbody>
</table>
Implementation

Java

Local persistence component allows for different storage engines to be plugged in:

- Berkeley Database (BDB) Transactional Data Store: object of tens of kilobytes
- MySQL: object of > tens of kilobytes
- BDB Java Edition, etc.
Performance

Guarantee Service Level Agreements (SLA)

the latencies exhibit a clear diurnal pattern (incoming request rate)

write operations always results in disk access

affected by several factors such as variability in request load, object sizes, and locality patterns
Improvement

A few customer-facing services required higher levels of performance.

Each storage node maintains an object buffer in its main memory.

Each write operation is stored in the buffer and gets periodically written to storage by a writer thread.

Read operations first check if the requested key is present in the buffer.
Improvement (Cont’d)

lowering the 99.9th percentile latency by a factor of 5 during peak traffic

write buffering smooths out higher percentile latencies
Improvement (Cont’d)

A server crash can result in missing writes that were queued up in the buffer.

To reduce the durability risk, the write operation is refined to have the coordinator choose one out of the N replicas to perform a “durable write”

Since the coordinator waits only for W responses, the performance of the write operation is not affected by the performance of the durable write operation.
Out-of-balance

If the node’s request load deviates from the average load by a value more than a certain threshold (here 15%) Imbalance ratio decreases with increasing load under high loads, a large number of popular keys are accessed and the load is evenly distributed
Strategy 3 achieves better efficiency

*Faster bootstrapping/recovery:*

Since partition ranges are fixed, they can be stored in separate files, meaning a partition can be relocated as a unit by simply transferring the file (avoiding random accesses needed to locate specific items).

*Ease of archival*

Periodical archiving of the dataset is a mandatory requirement for most of Amazon storage services.

Archiving the entire dataset stored by Dynamo is simpler in strategy 3 because the partition files can be archived separately.
Coordination

Dynamo has a request coordination component that uses a state machine to handle incoming requests.

Client requests are uniformly assigned to nodes in the ring by a load balancer.

An alternative approach to request coordination is to move the state machine to the client nodes.

In this scheme client applications use a library to perform request coordination locally.
Coordination

The latency improvement is because the client-driven approach eliminates the overhead of the load balancer and the extra network hop that may be incurred when a request is assigned to a random node.
Conclusion

Dynamo is a highly available and scalable data store for Amazon.com’s e-commerce platform.

Dynamo has been successful in handling server failures, data center failures and network partitions.
Conclusion (Cont’d)

Dynamo is incrementally **scalable** and allows service owners to scale up and down based on their current request load.

Dynamo allows service owners to **customize** their storage system by allowing them to tune the parameters N, R, and W.