Contents

• Applications
  • Amazon’s Dynamo continued
  • CDNs
  • Coral CDN
  • Internet Indirection Infrastructure
  • Planetlab
Amazon Dynamo

Aim is to store various kinds of data and have high availability

Build a distributed storage system that scales

Simple: **key-value**
- Highly available
- Guarantee **Service Level Agreements (SLA)**

Based on the SOSP 2007 presentation and paper: Dynamo: Amazon’s Highly Available Key-value Store
System Assumptions and Requirements

Query Model: simple read and write operations to a data item that is uniquely identified by a key

ACID Properties: Atomicity, Consistency, Isolation, Durability

Consistency is relaxed for efficient operation → noSQL data stores such as Dynamo

Efficiency: latency requirements which are in general measured at the 99.9th percentile of the distribution

Other Assumptions: operation environment is assumed to be non-hostile and 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

Multi-hop routing increases variability in routing
Dynamo is a one-hop DHT

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*
Dynamo Design Consideration

Sacrifice strong **consistency** for **availability**

Conflict resolution is executed during *read* instead of *write*
Use quorums and other techniques

Other principles:
- Incremental scalability
- Symmetry
- Decentralization
- Heterogeneity
Inconsistencies in distributed systems

Network partitions

Node is down

Process crashes before flushing

File corruption

...
CAP Theorem

CAP, first conceived in 2000 by Eric Brewer and formalized into a theorem in 2002 by Nancy Lynch

A useful model for describing the fundamental behavior of NoSQL systems

CAP is generally described as following:
Of three desirable properties you want in your system: consistency, availability and tolerance of network partitions,

you can only choose two.
## Summary of Techniques 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 (use another server for replica if proper one is not available)</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 (summarization of key ranges of virtual nodes)</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>
Dynamo Implementation

Data Stores
Nodes in the system are spread around a logical circle
Nodes are responsible for the region between it and its predecessor
Virtual nodes are evenly dispersed and appear to be regular nodes in the system, but in reality are just handled by the nodes of the system
Can be geographically distributed

Object Data
Uses hashing of an object’s key to determine where to store the object
Each object is replicated across N nodes (N-1 successor nodes to the coordinator node)
Consistent Hashing Revisited

Properties

**Smoothness** → addition of bucket does not cause movement between existing buckets

**Spread & Load** → small set of buckets that lie near object

**Balance** → no bucket is responsible for large number of objects

Moderate load imbalance is possible

Virtual nodes address this

**Log n replication** factor gives $O(\text{items}/n)$ balance with high probability for a high number of uniformly distributed items
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.

Virtual nodes are needed to address data/node imbalance problem.
Replication

Each data item is replicated at N hosts

“preference list”: The list of nodes that is responsible for storing a particular key (physical nodes)

Multiple data centers: Preference list is configured so that each object is replicated across multiple data centers

Nodes B, C and D store keys in range (A, B including K)
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

**Challenge:** an object having distinct version subhistories, which the system will need to reconcile in the future

**Solution:** uses **vector clocks** in order to capture causality between different versions of the same object
A vector clock is a list of \textbf{(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 counters in the second clock, then the first is an ancestor of the second and can be forgotten.

Figure from the Dynamo article.
Sloppy Quorum

The sloppy quorum technique is used to handle temporal faults.

Read/Write involve $N$ nodes (preference list) $R, W$ are 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.

$R$ and $W$ are usually configured to be less than $N$, to provide better latency.

Typical values $(3, 2, 2)$ $(N, R, W)$.
Dynamo Execution

**Writes**
- Requires generation of a new vector clock by coordinator
- Coordinator writes locally
- Forwards to N nodes, if W-1 respond then the write was successful

**Reads**
- Forwards to N nodes, if R-1 respond then forwards to user
- Only unique responses forwarded
- User handles merging if multiple versions exist
Gossip

A **gossip**-based protocol propagates membership changes and maintains an **eventually consistent** view of membership.

Each node contacts a peer chosen at **random every second**.

The two nodes efficiently reconcile their persisted membership change histories.

Also reconcile position information on the ring (virtual buckets).
Hinted handoff

The hinted handoff is also used to handle temporal faults.

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.

As a result $A$ is always writable.
Eventual consistency

Given the **CAP impossibility result**, weaker consistency models were proposed to ensure availability and high performance.

**Eventual consistency**

All servers converge to the same state, but can be in differing states while converging.

A weak property

Works with network partitions

If a network partition happens the updates are eventually propagated to all servers.
Anti-entropy overview

To ensure eventual consistency, replicas exchange information about the data (writes) they have witnessed.

This information exchange is called **anti-entropy**
*after reversing entropy (thermodynamic randomness)*

Multiple ways to implement anti-entropy

- Should be an asynchronous process
- Broadcast, multicast, …
- A replica typically broadcasts a write to other replicas
- A logical clock is used to handle concurrent writes
Anti-entropy (replica synchronization)

A Merkle tree is a hash tree where leaves are hashes of the individual keys.

Can check each branch independently without downloading the full data.

Used to verify consistency of replicas (which replicas are out of sync).

One Merkle tree for a virtual node (key range).

Make divergent data sets consistent.
Merkle Tree

<table>
<thead>
<tr>
<th>Block B1</th>
<th>H(B1)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Block B2</td>
<td>H(B2)</td>
</tr>
<tr>
<td>Block B3</td>
<td>H(B3)</td>
</tr>
<tr>
<td>Block B4</td>
<td>H(H(B4), H(B5))</td>
</tr>
<tr>
<td>Block B5</td>
<td>H(B5)</td>
</tr>
</tbody>
</table>
Some benefits of anti-entropy

No need for complex failure code or master node election

Writes are fast, only local state is updated before returning

The replicas are updated with an asynchronous background process

Replicas can also be located in more distant data centers
Anti-entropy and guarantees

Typical requirements:
- **Safety**: nothing bad ever happens
- **Liveness**: something good eventually happens

Anti-entropy does not guarantee safety

But works reasonably well in practice

There are tools to assess the consistency of an anti-entropy system
Determine inconsistency window and then set the parameters
CALM theorem

CALM means *consistency as logical monotonicity*

Informally, programs that compute an ever-growing set of facts and do not retract emitted facts can be run safely on an eventually consistent store.

Example:

Stock trades $\rightarrow$ CALM

Latest stock trade $\rightarrow$ Not CALM

Reading:

Eventual Consistency Today: Limitations, Extensions, and Beyond by Peter Bailis, Ali Ghodsi

http://queue.acm.org/detail.cfm?id=2462076
Results

Average and 99.9 percentiles of latencies for read and write requests during peak request season of December 2006. The intervals between consecutive ticks in the x-axis correspond to 12 hours.

Latencies follow a diurnal pattern similar to the request rate and 99.9 percentile latencies are an order of magnitude higher than averages.
Dynamo Summary

“Eventually” consistent data store
Always writable

Decentralized

All nodes have the same responsibilities

Amazon.com’s Resolution
   Weakening consistency property in the system
   Increase the availability
Datastores

No-SQL
Dynamo, Apache Cassandra (column oriented as well)
BigTable (Google) (sparse, distributed multi-dimensional sorted map)
Hbase (Facebook)
Riak (Dynamo clone in Erlang)
Voldemort (LinkedIn)

P2P datastores
BitTorrent
Chord/DHT based
Freenet
PAST
GNUnet (not covered)
Content Delivery Networks (CDN)

Geographically distributed network of Web servers around the globe (by an individual provider, E.g. Akamai).

Improve the performance and scalability of content retrieval.

Allow several content providers to replicate their content in a network of servers.
Motivation

Network cost
Huge cost involved in setting up clusters of servers around the globe and corresponding increase in network traffic

Economic cost
Higher cost per service rate making them inaccessible to lower and medium level customers
Use cases

Static or slow changing content

Popular content shared by many users

Geographically distributed users

Ad-hoc or irregular usage that does not benefit from local caching

Expensive or saturated bandwidth connections

Flashcrowds
CDN Technology

Intelligent wide area traffic management
Direct clients’ requests to optimal site based on topological proximity

Two types of redirection: DNS redirection or URL rewriting

Cache
Saves useful contents in cache nodes.

Two cache policies: least frequently used standard and least recently used standard.
CDN Types (*Skeletal*)

- **CDNs**
  - **Hosting CDN**
  - **Relaying CDN**
    - **Partial Site Content Delivery**
    - **Full Site Content Delivery**
      - *Request Routing Techniques*
        - **DNS based**
        - **URL Rewriting**
CDN

Replicate content on many servers

Challenges

How to replicate content
Where to replicate content
How to find replicated content
How to choose among known replicas

How to direct clients towards replica
   DNS, HTTP redirect, anycast, etc.

Akamai
Server Selection

Service and content is replicated in many places in network

How to direct clients to a particular server?
As part of routing → anycast, cluster load balancing
As part of application → HTTP redirect
As part of naming → DNS

Which server to use?
Best performance → to improve client performance
   Based on Geography? RTT? Throughput? Load?
Lowest load → to balance load on servers
Any active node → to provide availability
Client ISP

Clients

Client DNS

(Local DNS server for client)

redirection

CDN

CDN DNS

Content Provider

DNS

1

2

3

4

5

6
Surrogate Server placement problem

Given N possible locations at edge of the Internet, we are able to place K (K<N) surrogate servers, how to place them to minimize the total cost?

This is the: **Minimum K-Median Problem**

Given N points, we must select K (centers), and then assign each input point j to the selected center that is closest to it. The goal is to select K centers so as to minimize the sum of the assignment costs.

This is NP-Hard

Note: many cost functions can be used, can take into account clients that generate greatest load

Many heuristics algorithms to solve this (tree, greedy, hot spot...
<table>
<thead>
<tr>
<th>CDN</th>
<th>Type</th>
<th>Coverage</th>
<th>Solutions</th>
</tr>
</thead>
<tbody>
<tr>
<td>Akamai</td>
<td>Commercial</td>
<td>Market leader</td>
<td>Edge platform for handling static and dynamic content, DNS-based request-routing</td>
</tr>
<tr>
<td></td>
<td>CDN service including streaming data</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Limelight Networks</td>
<td>Commercial</td>
<td>Surrogate servers in over 70 locations in the world</td>
<td>Edge-based solutions for content delivery, streaming support, custom CDN for custom delivery solutions, DNS-based request-routing</td>
</tr>
<tr>
<td></td>
<td>On-demand distribution, live video, music, games, ...</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Coral</td>
<td>Academic</td>
<td>Experimental, hosted on PlanetLab</td>
<td>Uses a DHT algorithm (Kademlia), support for static content, DNS-based request-routing</td>
</tr>
<tr>
<td></td>
<td>Content replication based on popularity (on demand), addresses flash crowds</td>
<td></td>
<td></td>
</tr>
<tr>
<td>CoDeeN</td>
<td>Academic testbed</td>
<td>Experimental, hosted on PlanetLab, collaborative CDN</td>
<td>Support for static content, HTTP direction</td>
</tr>
<tr>
<td></td>
<td>Caching of content and redirection of HTTP requests</td>
<td></td>
<td>Consistent hashing for mapping data to servers</td>
</tr>
<tr>
<td>Globule</td>
<td>Academic</td>
<td>Apache extension, Open Source collaborative CDN</td>
<td>Support for static content, monitoring services, DNS-based request-routing</td>
</tr>
<tr>
<td></td>
<td>Replication of content, server monitoring, redirection to available replicas</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Akamai

Clients fetch html document from primary server
URLs for replicated content are replaced in html

Client resolves aXYZ.g.akamaitech.net hostname

Akamai.net name server returns NS record for g.akamaitech.net
G.akamaitech.net nameserver choses server in region

Should try to choose server that has file in cache - How to choose?

Uses aXYZ name and consistent hash
How Akamai Works

1. End-user requests "index.html" from "cnn.com".
2. "cnn.com" forwards the request to the DNS root server.
3. The DNS root server queries an Akamai high-level DNS server.
4. The Akamai high-level DNS server forwards the request to an Akamai low-level DNS server.
5. The Akamai low-level DNS server forwards the request to an Akamai server.
6. The Akamai server identifies the IP address of the "cnn.com" content provider.
7. The Akamai server requests "foo.jpg" from "cnn.com".
8. "cnn.com" forwards the request to the DNS root server.
10. The Akamai high-level DNS server forwards the request to an Akamai low-level DNS server.
11. The Akamai low-level DNS server forwards the request to an Akamai server.
12. The Akamai server sends the "foo.jpg" to the End-user.

Source: www.cs.cmu.edu/~srini/15-744/S08/lectures/17-DNS.ppt
Coral: An Open CDN

Pool resources to dissipate flash crowds

Implement an open CDN
Allow anybody to contribute
Works with unmodified clients
CDN only fetches once from origin server

Runs in PlanetLab
Based on NSDI 2004 presentation and paper
Using CoralCDN

Rewrite URLs into “Coralized” URLs


Coral distributes the load

Who might “Coralize” URLs?
Web server operators Coralize URLs
Coralized URLs posted to portals, mailing lists
Users explicitly Coralize URLs
Coral Server Discovery

1. Each Coral server inserts its IP network prefix as key, its IP address as value
2. DNS server does DHT lookup on client IP prefix to find nearby Coral server

Details (multiple prefixes):

Each Coral server uses traceroute to find nearby routers
Registers itself under IP of each nearby router

Coral DNS server traceroutes to client
  Looks up each router IP address in mapping
DNS Redirection
Return proxy, preferably one near client
Hierarchical DHT

A hierarchy of DHTs, with clustering at lower levels
DHT based on XOR metric

Nearby (< 20 ms) Coral nodes form an L2 DHT
  L1: 60 ms
  L0: global
Search in L2 DHT first
  If nearby copy exists, will find it first
Only search L1, L0 if miss in lower level
Finding URLs

Look up the URL in a DHT
key=URL, value=IP addr of Coral cache that has the URL

Coral cache fetches the page from that other cache

If DHT had more than one value for key, fetch page from more than one
In case one is down or slow
Challenges for DNS Redirection

Coral lacks…

Central management

*A priori* knowledge of network topology

Anybody can join system

Any special tools (e.g., BGP feeds)

Coral has…

Large number of vantage points to probe topology

Distributed index in which to store network hints

Each Coral node maps nearby networks to self
Internet Indirection Infrastructure (i3)

- A DHT-based overlay network
  - Based on Chord
- Aims to provide more flexible communication model than current IP addressing
- Also a forwarding infrastructure
  - i3 packets are sent to identifiers
  - each identifier is routed to the i3 node responsible for that identifier
  - the node maintains triggers that are installed by receivers
  - when a matching trigger is found the packet is forwarded to the receiver
i3 II

- An i3 identifier may be bound to a host, object, or a session
- i3 has been extended with ROAM
  - Robust Overlay Architecture for Mobility
  - Allows end hosts to control the placement of rendezvous-points (indirection points) for efficient routing and handovers
  - Legacy application support
    - user level proxy for encapsulating IP packets to i3 packets
R inserts a trigger (id, R) and receives all packets with identifier id.

Mobility is transparent for the sender.

the host changes its address from R1 to R2, it updates its trigger from (id, R1) to (id, R2).

Source: http://i3.cs.berkeley.edu/
A multicast tree using a hierarchy of triggers
Anycast using the longest matching prefix rule.
Sender-driven service composition using a stack of identifiers

Receiver-driven service composition using a stack of identifiers

Source: http://i3.cs.berkeley.edu/
PlanetLab (www.planet-lab.org)

Global research network that support the development of network services
As of Feb 2014, PlanetLab has 1181 nodes at 567 sites