# Dynamo

## Amazon’s Highly-Available Key-value Store

2007

Giuseppe DeCandia, Deniz Hastorun, Madan Jampani, Gunavardhan Kakulapati, Avinash Lakshman, Alex Pilchin, Swaminathan Sivasubramanian, Peter Vosshall and Werner Vogels

presented by Slavik Derevyanko

# Outline

- Dynamo overview and design considerations
- CAP: consistency vs availability trade-off
- Dynamo architecture
- Dynamo / Bigtable comparison

# Overview

- Dynamo is a highly-available large-scale distributed key-value datastore
- Used by core services powering Amazon's e-commerce platform - shopping carts, best seller lists, customer preferences, product catalog, etc.
- Completely decentralized architecture - no dedicated coordination servers
- Strong fault-tolerance to server and network failures - an "always-on" experience
- Uses eventual consistency model for object replicas - sacrifices strict consistency for availability

# Design considerations

- Most applications within Amazon only store and retrieve by primary keys - Dynamo offers a simple primary-key access interface - get(key), put(key, object)
- No support for advanced database features: transactions, joins, relational schema - dropping these features significantly improves scalability
- Weak support for ACID transactional guarantees: favors availability over consistency, no transaction isolation, etc.
- Stringent latency requirements (measured in 99.9th percentile of the distribution)
- Non-hostile environment - no authentication nor authorization

# Service-level agreements

- Amazon must deliver its functionality in strictly limited response time: every dependency in the platform needs to deliver its functionality within tight time 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.

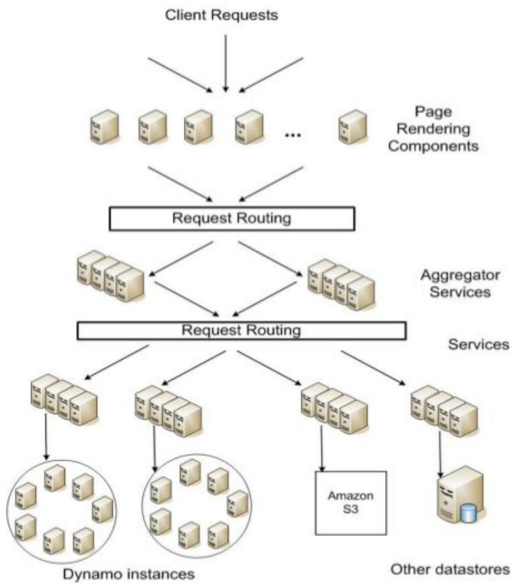



**Figure 1: Service-oriented architecture of Amazon's platform**

# CAP: consistency vs availability trade-off

# Eric Brewer and the CAP "theorem"

*A distributed system can have at most two of the three following properties: Consistency, Availability, and tolerance to network Partitions.*

Eric Brewer
Professor, University of California, Berkeley
VP Infrastructure, Google
2000

In 2002, Gilbert and Lynch converted "Brewer's conjecture" into a more formal definition with an informal proof.

# Understanding CAP

Example of an update operation in a partitioned DB

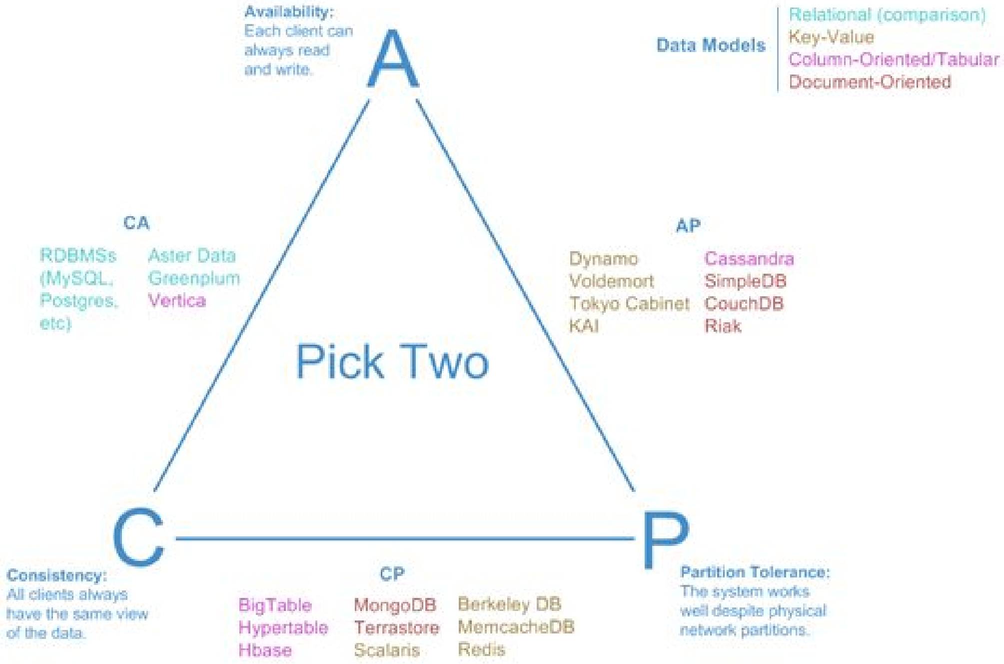
Two nodes on opposite sides of a partition yield a CAP C/A choice:

- Preserving availability: allowing at least one node to update state will cause the nodes to become inconsistent, thus forfeiting C.
- Preserving consistency: one side of the partition must act as if it is unavailable, thus forfeiting A.
- Preserving both C and A: only when nodes communicate, thereby forfeiting P.

# Dynamo’s consistency guarantees

- “From the very early replicated database works, it is well known that when dealing with the possibility of network failures, strong consistency and high data availability cannot be achieved simultaneously [2, 11].” (1984, 1979).
- Availability is increased by using optimistic replication techniques - i.e. changes are propagating to replicates in the background - **eventual consistency**.
- Conflict resolution considerations:
  - when to resolve: Dynamo delays conflicts resolution until the data is read (always writable)
  - who resolves: database engine (tactics like “last write wins”), or the client app (merging carts, etc)

# Distributed databases and CAP



# Replica consistency with HBase

## 72. Timeline-consistent High Available Reads

### 72.1. Introduction

HBase, architecturally, always had the strong consistency guarantee from the start. All reads and writes are routed through a single region server, which guarantees that all writes happen in an order, and all reads are seeing the most recent committed data.

### 72.2. Timeline Consistency

With this feature, HBase introduces a Consistency definition, which can be provided per read operation (get or scan).

```
public enum Consistency {
    STRONG,
    TIMELINE
}
```

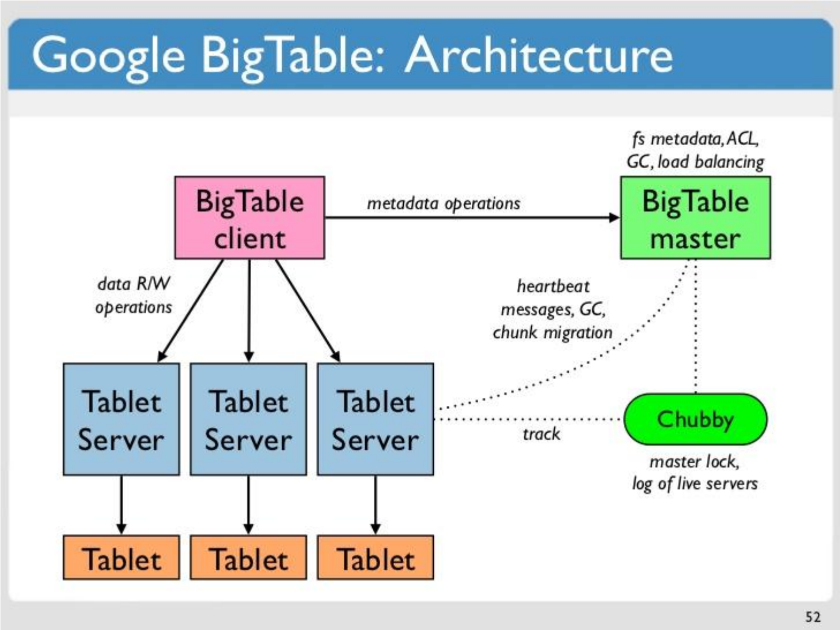
`Consistency.STRONG` is the default consistency model provided by HBase. In case the table has region replication = 1, or in a table with region replicas but the reads are done with this consistency, the read is always performed

# Dynamo architecture

# Architecture comparison

## Amazon Dynamo:

- **Incremental scalability**: automatic scaling out one host at a time.
- **Symmetry**: Every node has the same set of responsibilities as its peers.
- **Decentralization**: Design favors decentralized peer-to-peer techniques over centralized control. This leads to a simpler, more scalable, and more available system.
- **Heterogeneity**: work distribution is proportional to the capabilities of the individual servers. This is essential when adding new nodes with higher capacity



# Nodes partitioning

- Dynamically partitions data over the set of nodes
- **Consistent hashing:** the output range of a hash function is treated as a fixed circular space or “ring”.
- Each node in the system is assigned a random value within this space which represents its “position” on the ring.
- Each data item identified by a key is assigned to a node by hashing the data item’s key to yield its position on the ring.
- **Virtual nodes**: Each node can be responsible for more than one virtual node.

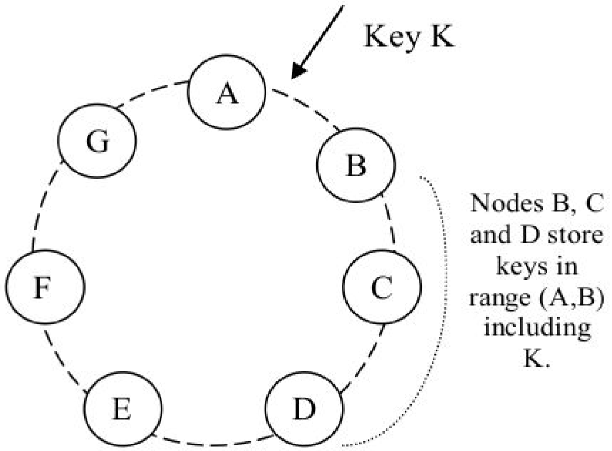



**Figure 2: Partitioning and replication of keys in Dynamo ring.**

# Object 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.
- Both "add to cart" and "delete item from cart" are put() requests in Dynamo
- Uses vector clocks in order to capture causality between different versions of the same object.
- A vector clock is a list of (node, counter) pairs
- **Every version of every object is associated with one vector clock**

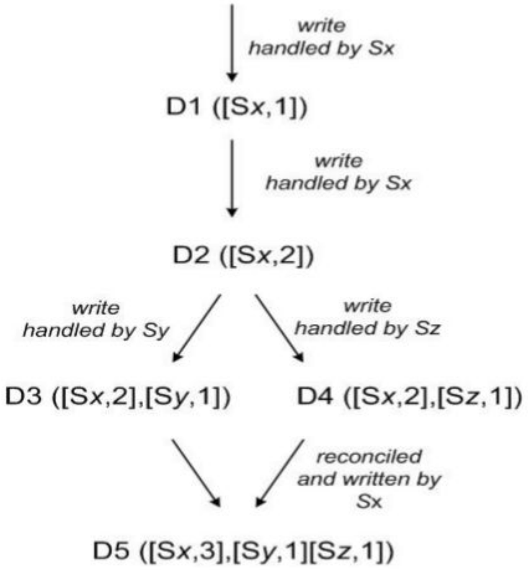



**Figure 3: Version evolution of an object over time.**

# Divergent versions: when and how many?

- The number of object versions returned to the shopping cart service was profiled for a period of 24 hours
- During this period, 99.94% of requests saw exactly one version; 0.00057% of requests saw 2 versions; 0.00047% of requests saw 3 versions and 0.00009% of requests saw 4 versions
- The increase in the number of concurrent writes is usually triggered by busy robots (automated client programs) and rarely by humans

# Execution of get() and put() operations

- Any storage node is eligible to receive client get and put operations for any key.
- To maintain consistency among its replicas, **a quorum protocol** is used.
- This protocol has two key configurable values: R and W.
  - R is the minimum number of nodes that must participate in a successful read operation.
  - W is the minimum number of nodes that must participate in a successful write operation.
- Setting R and W such that R + W > N yields a quorum-like system.
- R and W are usually configured to be less than N, to provide better latency.

# Conclusions

# Conclusions

## Dynamo vs. BigTable

| | Dynamo | BigTable |
|---|---|---|
| **data model** | key-value | multidimensional map |
| **operations** | by key | by key range |
| **partition** | random | ordered |
| **replication** | sloppy quorum | only in GFS |
| **architecture** | decentralized | hierarchical |
| **consistency** | eventual | strong (*) |
| **access control** | no | column family |

# Thank you!

UNIVERSITY OF
WATERLOO