CS 582: Distributed Systems

Dynamo: Amazon's Highly Available Key-value Store

Dr. Zafar Ayyub Qazi

Fall 2024

Specific learning outcomes

By the end of today's lecture, you should be able to:

- ❑ Explain the core design of Amazon Dynamo
- ❑ Analyze how Dynamo implements data partitioning and replication strategies
- ❑ Evaluate the consistency guarantees provided by Dynamo
- ❑ Compare and contrast the consistency models of Dynamo and ZooKeeper
- ❑ Analyze the role of sloppy quorums & hinted handoffs in Dynamo design and their impact on system availability
- ❑ Analyze how Dynamo achieves high availability

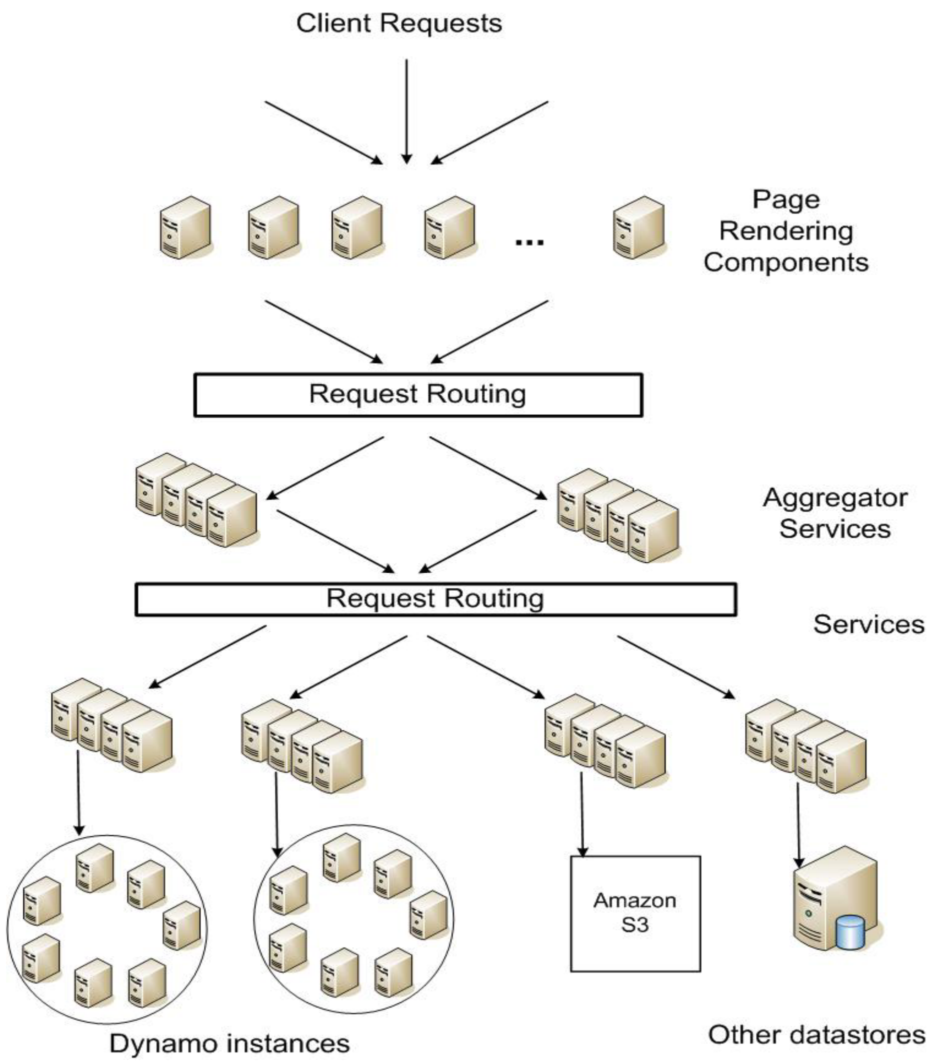
Great Success Story

- Many services at Amazon use it!
    - Shopping cart
    - Customer preferences
    - Session management
    - Sales rank
    - Product Catalog
    - …
- Now available as a cloud service on AWS
- Able to achieve high availability at Amazon scale
    - Applications using Dynamo have received successful responses for 99.9995% of their requests

Amazon Workload (in 2007)

- Peak load: Tens of millions of requests per day
- Tens of thousands of servers in globally distributed data centers

Architecture of Amazon's Platform



Tiered service-oriented architecture

- Stateless web page rendering servers
- Stateless aggregator servers
- Stateful data stores (e.g., Dynamo)

Dynamo Requirements

- Highly available despite failures being commonplace
  - "Even if a data center is destroyed by tornadoes"
  - "Always writeable"
- Low request-response latency
  - Focus on 99.9^{th} percentile of the distribution
  - Ensure it is less than a threshold, typically 300ms
- Incrementally scalable as servers grow to workloads
  - Adding "nodes" should be seamless
- Comprehensible conflict resolution
  - High availability in the above sense implies conflicts
  - Need a way to resolve conflicts

Operating Environment

- Dynamo is only used by Amazon's internal services
- Assumes its operating enviornment is non-hostile
  - No security related requirements such as authentication and authorization
- Each service uses its distinct instance of Dynamo

Some Key Design Questions

- How to partition and replicate data?
- How to route requests and handled them in a replicated system?
- What sort of consistency guarantees to provide?
- How to resolve conflicts?
- How to cope with node failures?
- How to detect failures?

Dynamo: Synthesis of Many Ideas

Consistent Hashing

Vector Clocks

Failure Detection

Virtual Nodes

Object versioning

Gossip-based membership protocol

Sloppy Quorums

Hinted Handoffs

Merkle Trees

Dynamo's System Interface

- Basic interface is a key-value store
  - get(k) and put(k, v)

How is data partitioned in Dynamo?

Consistent Hashing
with Virtual Nodes

How is data replicated in Dynamo?

Consistent Hashing
& Preference lists

Consistent Hashing Review

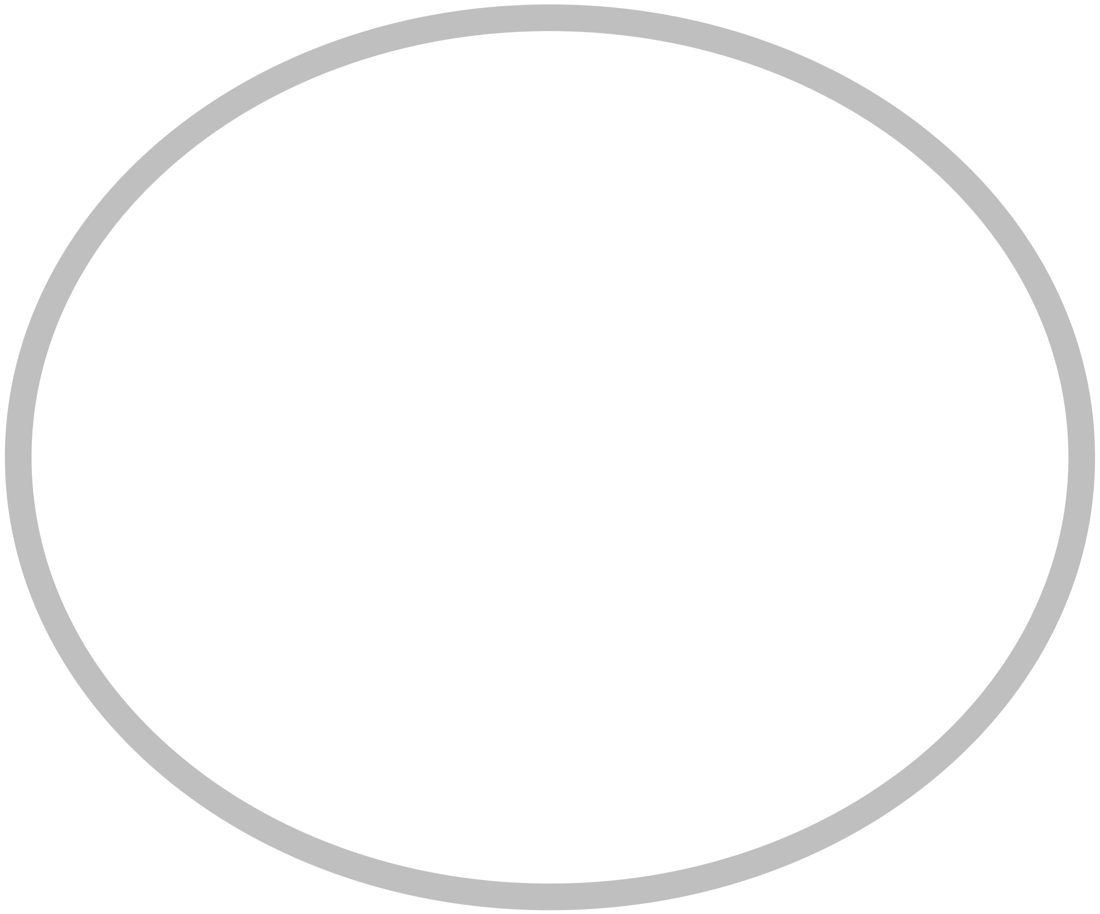
Consistent Hashing: Motivation

- Provides nice smoothness and data-balancing properties
- Widely used in industry:
  - Amazon's Dynamo data store
  - Facebook's Memcached system
  - Akamai's Content Delivery Network
  - Google's storage systems
  - Microsoft Azure's Storage System
  - Apache Casandra storage system
  - For in-network load balancing
    - Google's Maglev load balancer
  - …

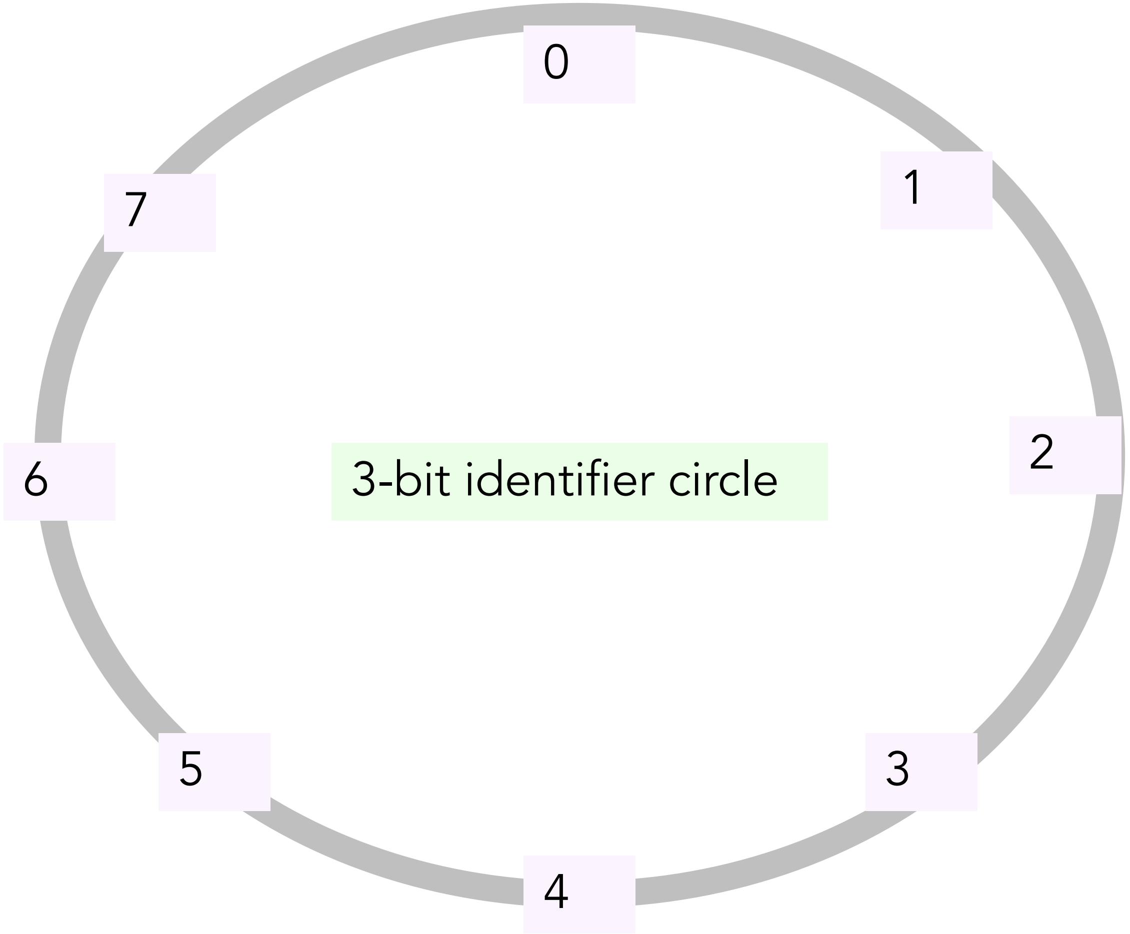
Consistent Hashing

- Servers and keys are mapped to an abstract circle called a hash ring or an identifier circle
- Construction
  - Assign each server and key an n-bit identifier using a base hash function such as SHA-1
  - Servers and keys get mapped to a mod 2^n circle
  - A key (and corresponding data) gets stored on the closest clockwise server

Consistent Hashing

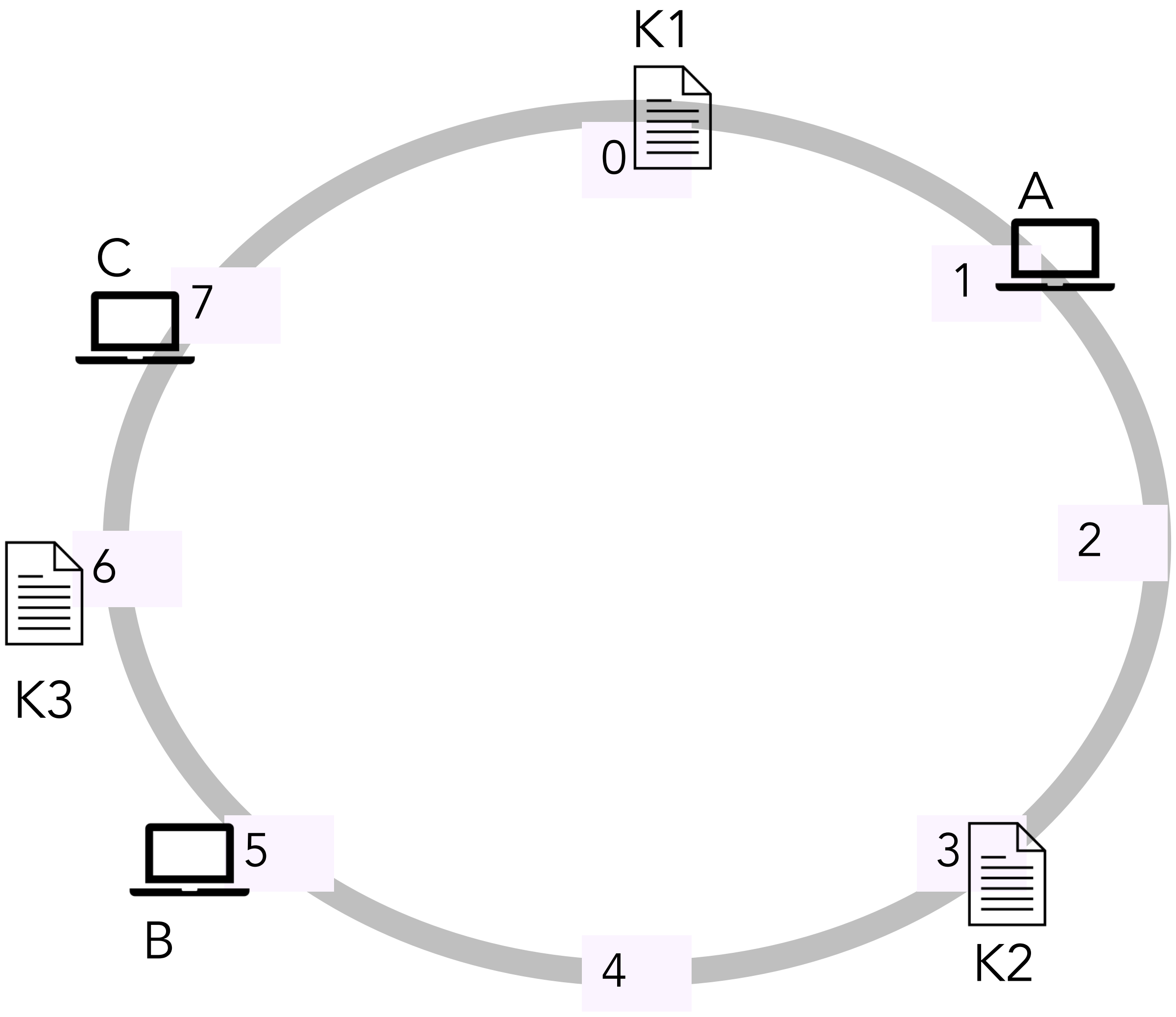


Consistent Hashing



3-bit identifiers

Consistent Hashing



3-bit identifiers

hash(node-IP)

hash(filename)

Consistent Hashing’s Load Balancing

Consistent Hashing's Load Balancing

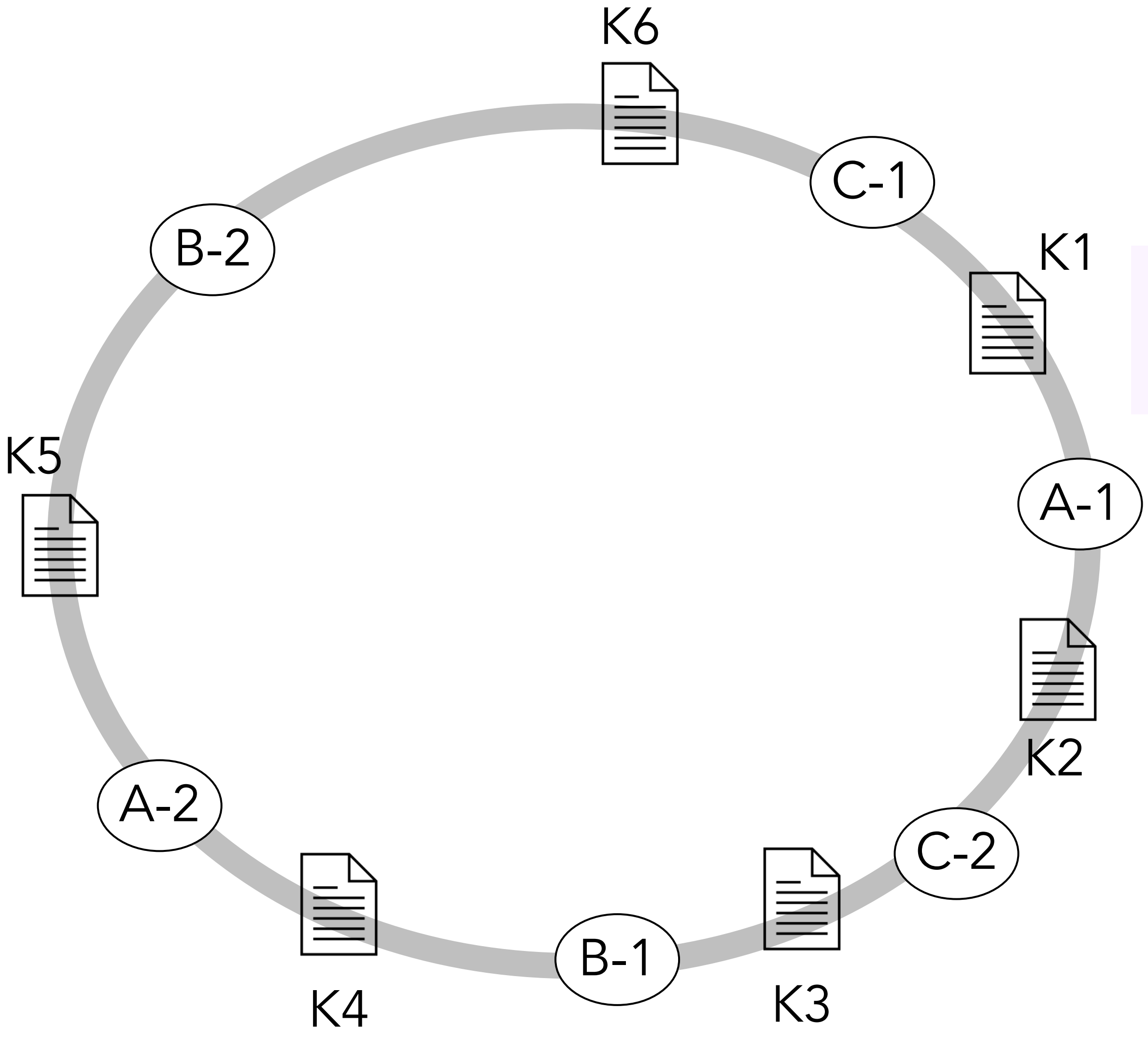
- Each server owns $1/N^{th}$ of the ID space in expectation
  - Where N is the number of servers

- What happens if a server fails?
  - If a server fails, its successor takes over the space
  - Smoothness goal : only the failed server's keys get relocated
  - But now successor owns $2/N^{th}$ of the key space
    - Failures can upset the load balance

- What if servers have different capacities?
  - The basic algorithm is oblivious to node heterogeneity

Virtual Nodes

Virtual Nodes



A,B,C,D represent different servers
& Y-X represents Y's virtual node

Virtual Nodes

- Idea: Each physical node now maintains V > 1 tokens
  - Each token corresponds to a *virtual node*
- Each virtual node owns an expected $1/(VN)^{th}$ of ID space
- Upon a physical node's failure, V successors take over
  - Result: Better load balance with larger V
- The number of virtual nodes that a node is responsible for can be decided based on its capacity

Theoretical Results

- For any set N of nodes, and K keys, with high probability:

- Each node is responsible for at most $(1+\epsilon)\frac{K}{N}$ keys
- When an $(N+1)^{st}$ node joins or leaves the network, responsibility for $O\left(\frac{K}{N}\right)$ keys changes hands (and only to and from the joining or leaving node)
- ϵ can be reduced to an arbitrarily small constant by having each node run $O(\log N)$ virtual nodes

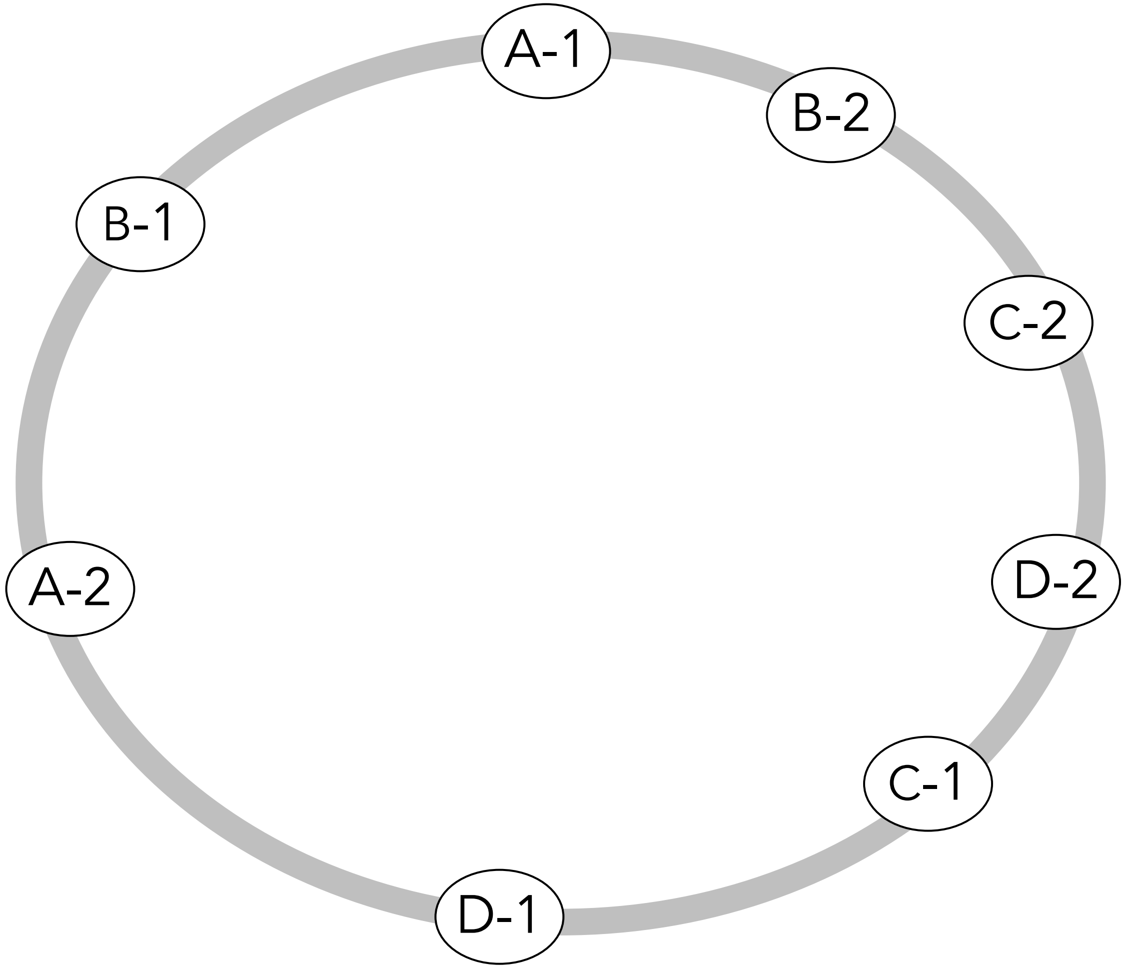
Proven by D. Lewin in his work "Consistent hashing and random trees", Master thesis, MIT, 1998

Summary so far …

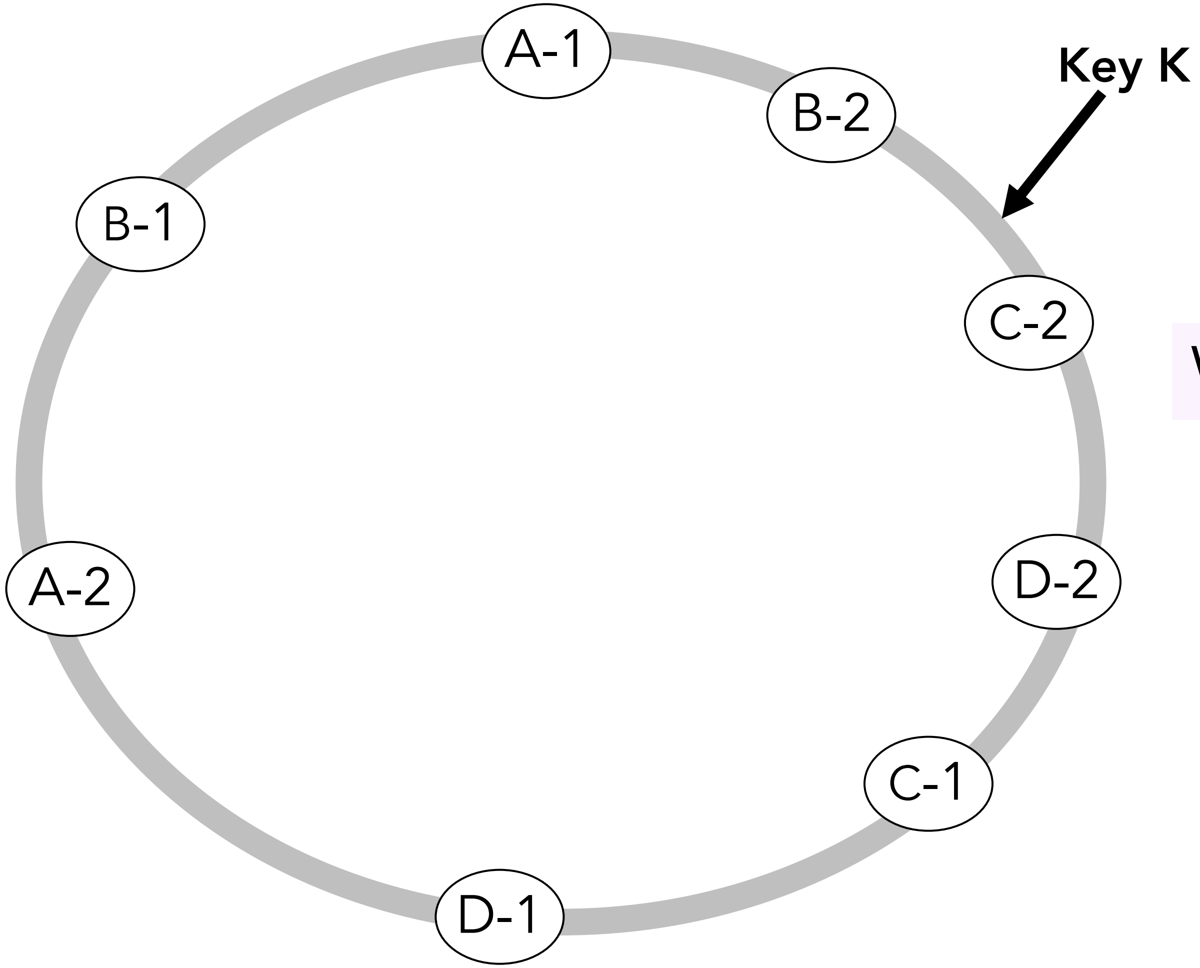
- Consistent hashing is widely used to partition data
  - Provides smoothness
  - However, load balancing can be impacted under node removal or addition
- Virtual nodes
  - Can help with load imbalance in case
    - Failures and different server capacities

Now let's come back to data partitioning and replication in Dynamo

Data Partitioning



Data Partitioning

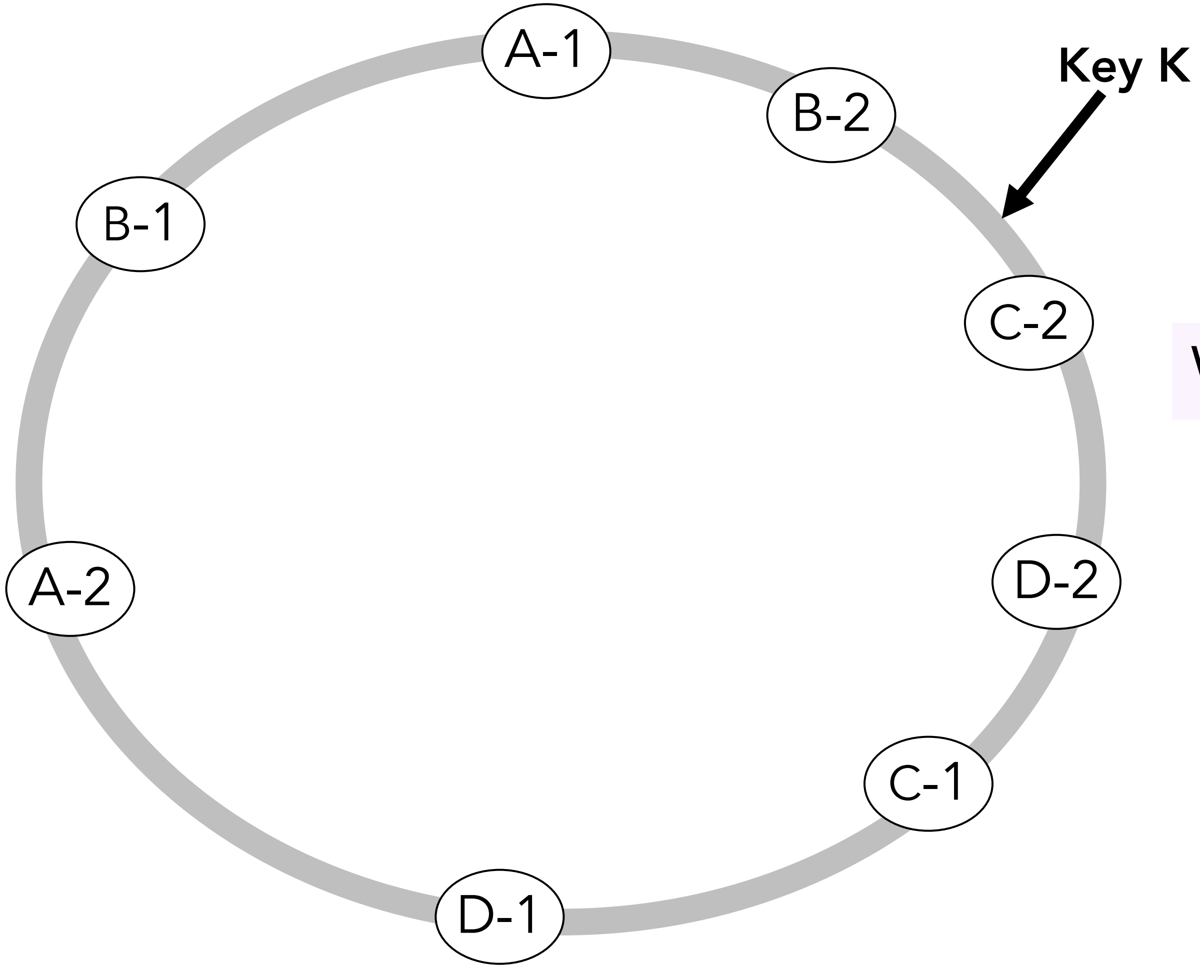


Which nodes would store key K?

Data Replication

- Data is replicated on N healthy successors
- Dynamo maintains a preferences list for each key
  - The preference list contains the nodes responsible for storing a particular key
- For robustness, the preference list is constructed by skipping virtual nodes to ensure distinct physical nodes
  - First N successor positions for a key may be owned by less than N distinct physical nodes
- To account for node failures the list contains more than N nodes

Data Replication



Which nodes would store key K?

Wide-area replication

- Preference lists always contain nodes from more than one data center

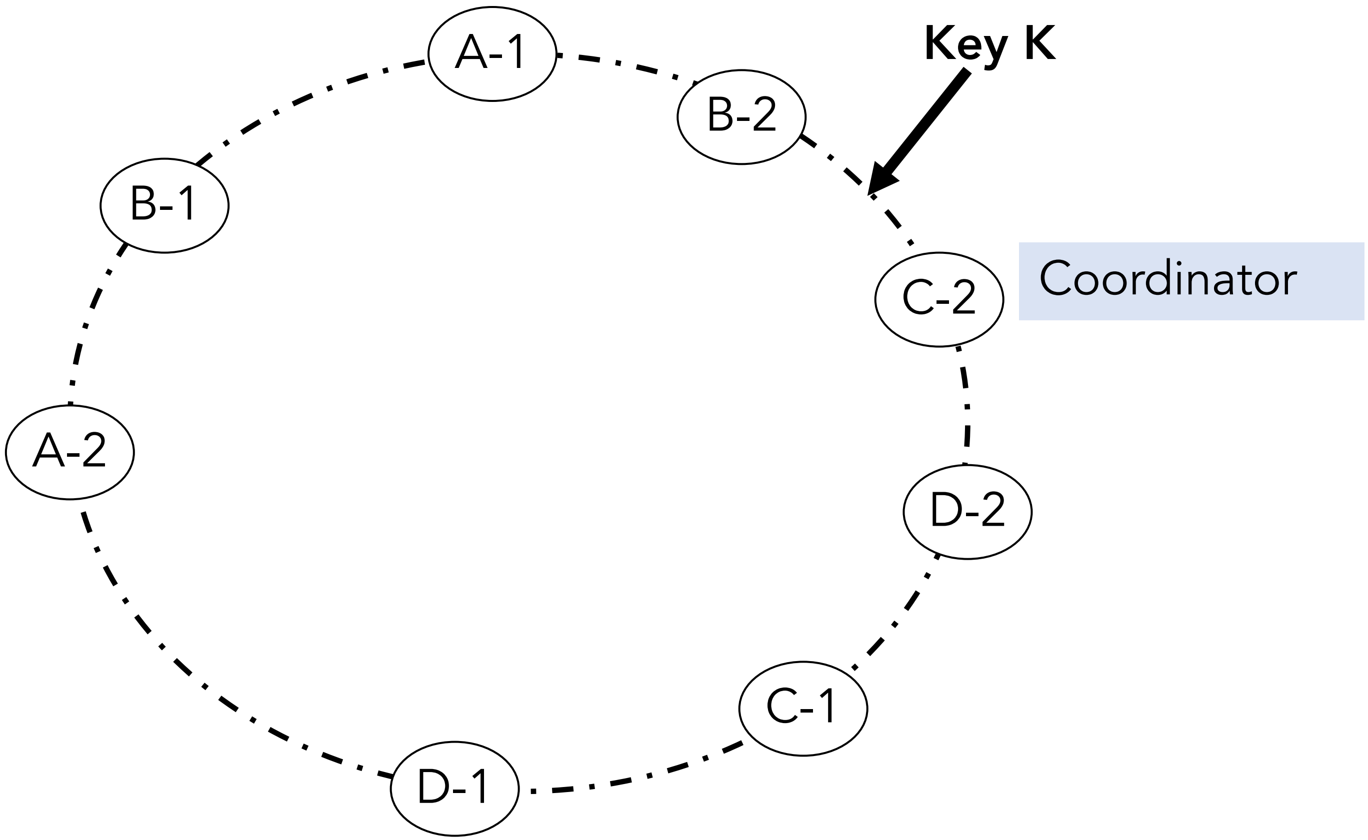
- Consequence: Data likely to survive failure of entire data center

Why Consistent hashing + Virtual nodes?

- Incremental scalability
  - If load increases and you want to add servers, you can do it dynamically while incurring minimal key reallocations
  - The newly available node accepts roughly similar load from other available nodes
- Heterogeneity in physical infrastructure
  - The number of virtual nodes that a node is responsible for can be decided based on its capacity
- Even load distribution under failure
  - If a node fails, the load handled by this node is evenly dispersed across different nodes

How are data requests routed?

Data Request Routing



Data Request Routing

- Coordinator node receives the request for a put(key ..)
- Coordinator then sends the request to first N healthy nodes in the preference list

How does the Coordinator know which other nodes to send a request to?

Gossip and Lookup

- Gossip: Once per second, each node contacts a randomly chosen other node
- They exchange their list of known nodes (including virtual node IDs)
- Each node eventually learns about the key ranges other nodes handle

When does a Coordinator decide to send the response back to the client?

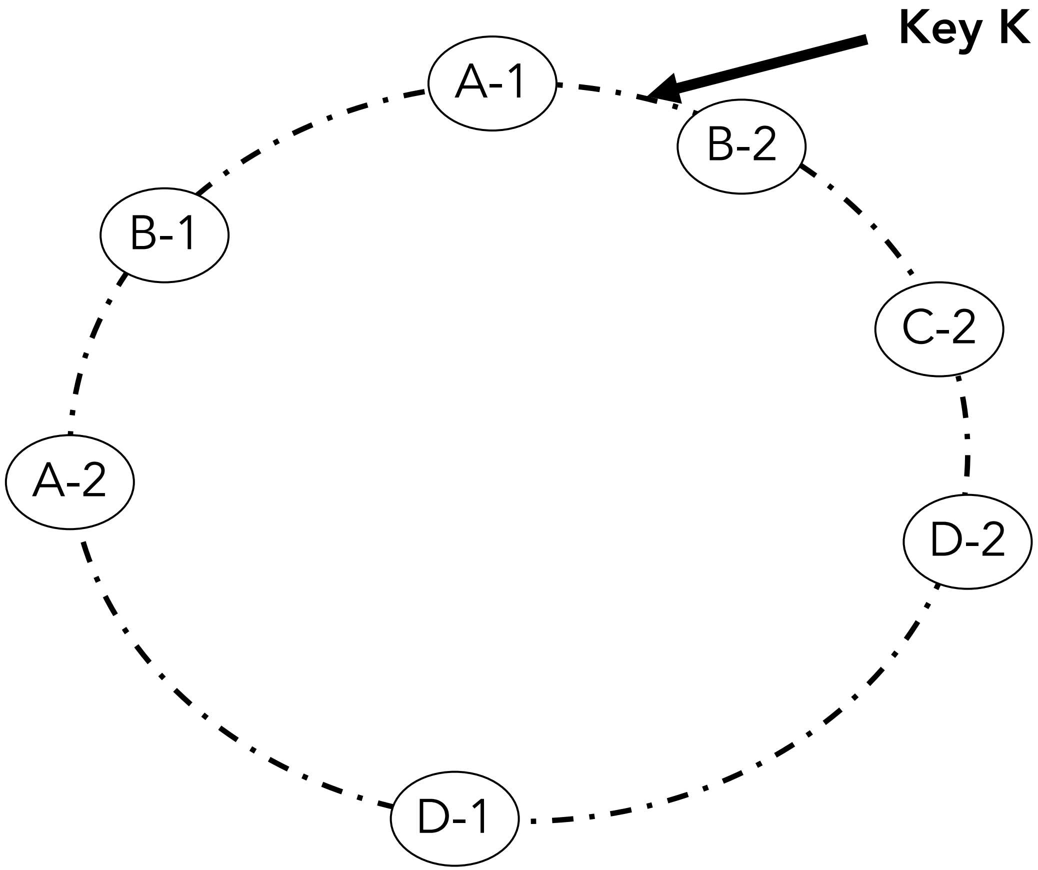
Quorum

- Goals:
    1. Do not block waiting for unreachable nodes
    2. put should always succeed
- Sends put/get to first N reachable nodes, in parallel
- put: waits for W replies (Write Quorums)
- get: waits for R replies (Read Quorums)
- Sloppy quorums: If the coordinator does not get the required replies, it tries successors in the preference list (beyond N)
- Hinted Handoffs
    - Indicates the intended replica node to the recipient
    - The recipient will periodically try to forward to the intended replica node

Sloppy Quorum and Hinted Handoff

- Suppose coordinator doesn't receive *W* replies when replicating a put()
  - Could return failure, but remember goal of high availability for writes...
- Coordinator tries next successors in preference list (beyond first *N*) if necessary
  - Indicates the intended replica node to recipient
  - Recipient will periodically try to forward to the intended replica node

Hinted Handoff Example



N = 3

Varying N, R, W

- Allows configuration of quorums
- Tuning of consistency, availability, performance
    - <u>And durability</u>

Varying N, R, W

- If R + W > N then writes intersect with future reads
  - If the set of nodes N is fixed
- If W > N/2 then two concurrent writes will intersect
  - If the set of nodes N is fixed

Consistency

- Common case given in paper: N = 3; R = W = 2
- What sort of consistency guarantees can be provided by Dynamo with this configuration?
- With these values, does Dynamo guarantee a get() sees all prior put()s?

What the Dynamo paper says …

“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 “

Conflicts

- Suppose N = 3, W = R = 2, nodes A, B, C
  - 1st put(k, …) completes on A and B
  - 2nd put(k, …) completes on B and C
  - Now get(k) arrives, completes first at A and C

- Conflicting results from A and C
  - Each has seen a different put(k, …)

- How is this conflict handled?

Conflicts vs. Applications

- Shopping Cart
  - Could take the union of two shopping carts

Conflicts vs. Applications

- Shopping Cart
  - Could take the union of two shopping carts
- What if the second put() was the result of user deleting item from cart stored in first put()
    - Result: “resurrection” of deleted items
- Can we do better? Can Dynamo resolve cases when multiple values are found?
  - Sometimes. If it can’t, application must do so

Dynamo summary of key design qns & ideas

1. How to partition and replicate data?
2. How to route and handle requests ?
3. How to cope with node failures?
4. What sort of consistency guarantees to provide?
5. How to resolve conflicts between replicas?
6. How to detect failures?

Consistent hashing + virtual nodes + preference lists

Coordinator nodes + Quorum + Gossip

Sloppy quorum + hinted handoff

Eventuall Consistency

Homework: Varying N, R, W

N	R	W
3	2	2
3	3	1
3	1	3
3	3	3
3	1	1

Next Lecture

- Wrap up our discussion on Dynamo
- Scaling Memcache at Facebook