Distributed Hash Tables

Ioan Raicu
Computer Science Department
Illinois Institute of Technology

CS 595: Data-Intensive Computing
November 9th, 2011

Distributed Hash Tables

- Hash Table that spans multiple computers
- Basic operations
  - Insert, Remove, Find, and Update
- Goals
  - Scalability and reliability
- Some distributed hash table implementations
  - Chord, CAN, Pastry, Tapestry, Cycloid, Kademlia, Dynamo, ZHT, C-MPI, MemCache

Dynamo

- Dynamo is used to manage the state of services(eg. Shopping carts, wish list…)
- Very high availability
- High reliability
- Balance between availability, consistency, cost-effectiveness and performance.

Motivation

- Build a distributed storage system:
  - Scalable
  - Simple: key-value
  - **Highly available**
  - **Guarantee Service Level Agreements (SLA)**

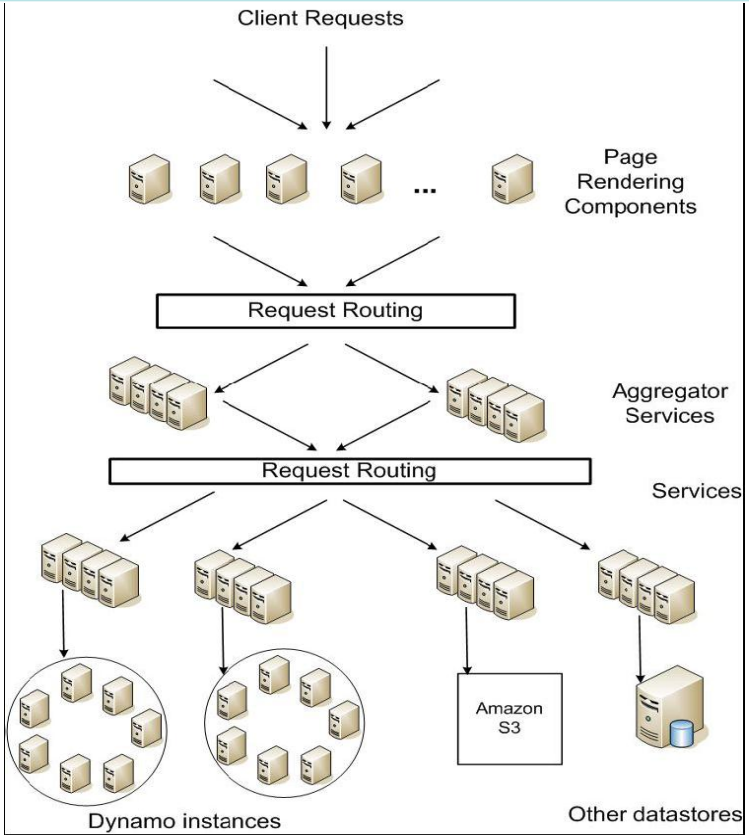
System Assumptions and Requirements

- Query Model: simple read and write operations to a data item that is uniquely identified by a key.
- ACID Properties: Atomicity, Consistency, Isolation, Durability.
- Efficiency: System must meet the strict Service Level Agreements requirement
- Other Assumptions: operation environment is safe, there are no security issue such as authentication and authorization.

Service Level Agreements (SLA)

- SLA: A contract where a client and a service agree on several system characteristics
  - The client's expected request rate distribution for a particular API
  - Expected service latency under certain conditions

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 of Amazon's platform

Design Consideration

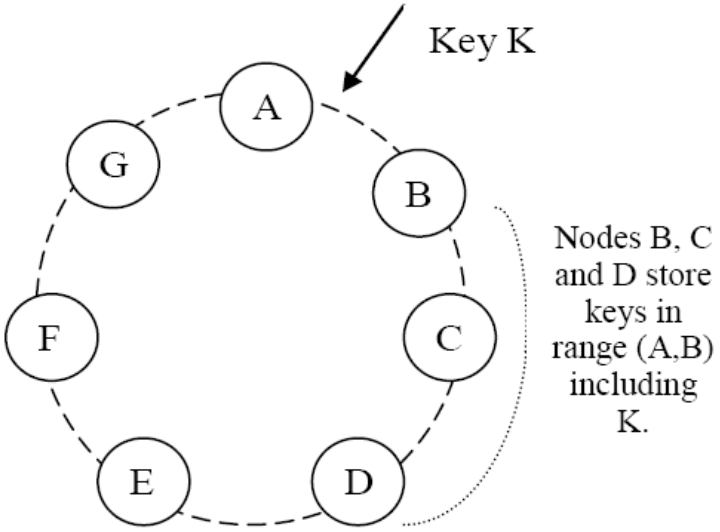
- Sacrifice strong consistency for availability
- Conflict resolution is executed during ***read*** instead of ***write*** : **always writeable**.
- Other principles:
  - Incremental scalability: scale node one by one
  - Symmetry: P2P character
  - Heterogeneity: nodes with different capacity

Summary of techniques used in *Dynamo* and their advantages

Problem	Technique	Advantage
Partitioning	Consistent Hashing	Incremental Scalability
High Availability for writes	Vector clocks with reconciliation during reads	Version size is decoupled from update rates.
Handling temporary failures	Sloppy Quorum and hinted handoff	Provides high availability and durability guarantee when some of the replicas are not available.
Membership and failure detection	Gossip-based membership protocol and failure detection.	Preserves symmetry and avoids having a centralized registry for storing membership and node liveness information.

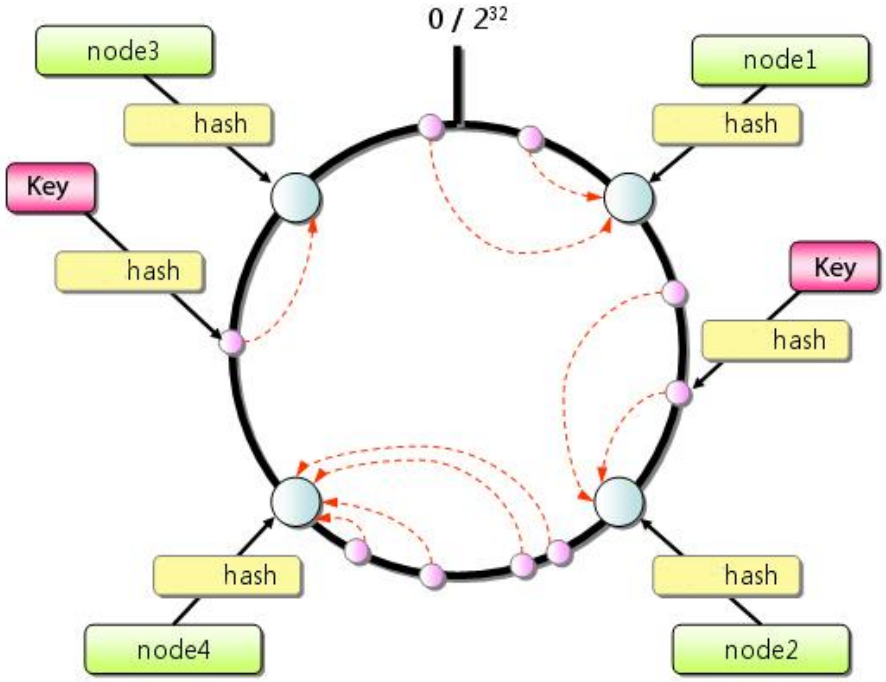
Partition Algorithm

- 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.

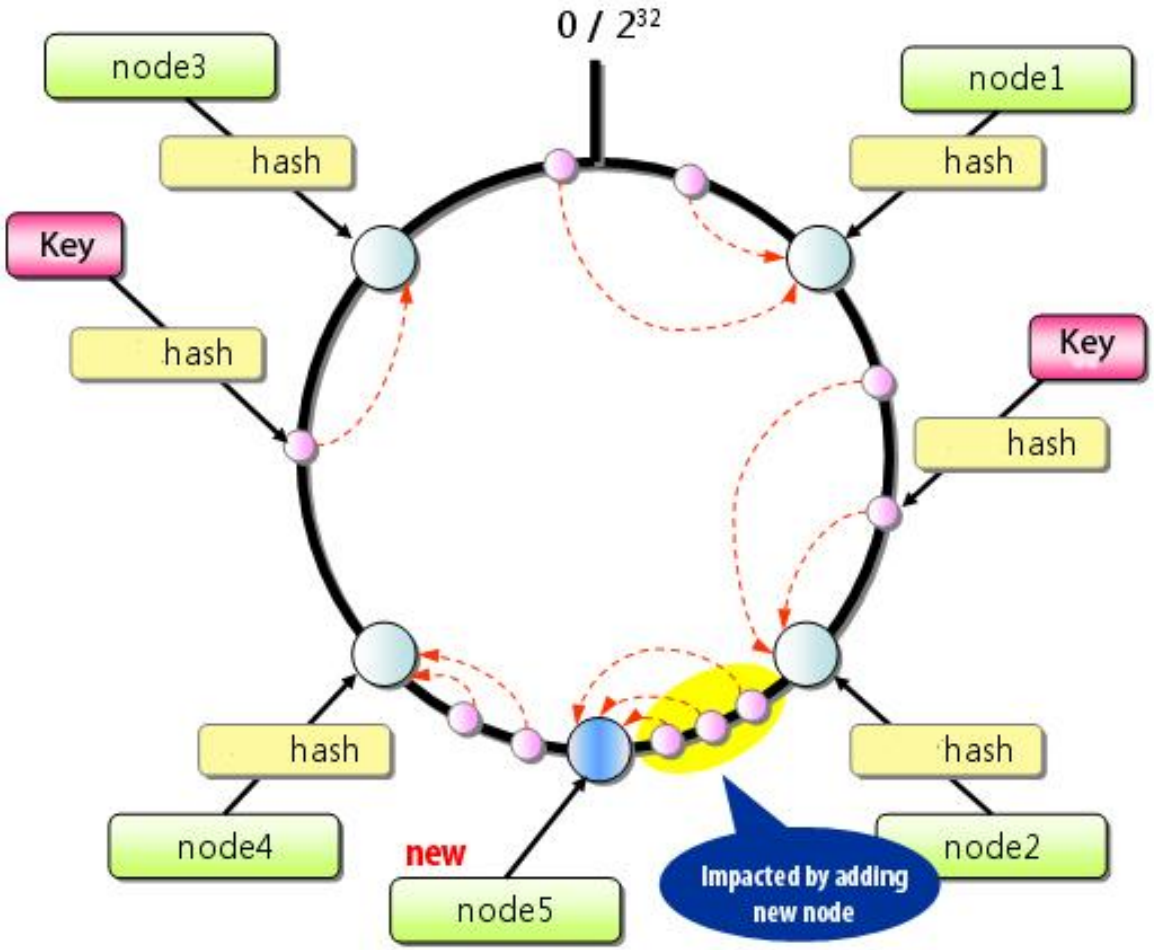


Consistent hashing

- Each node is assigned a random value represents its “position”
- Key need to search the first node along the circle



Consistent hashing: add new node



Consistent hashing

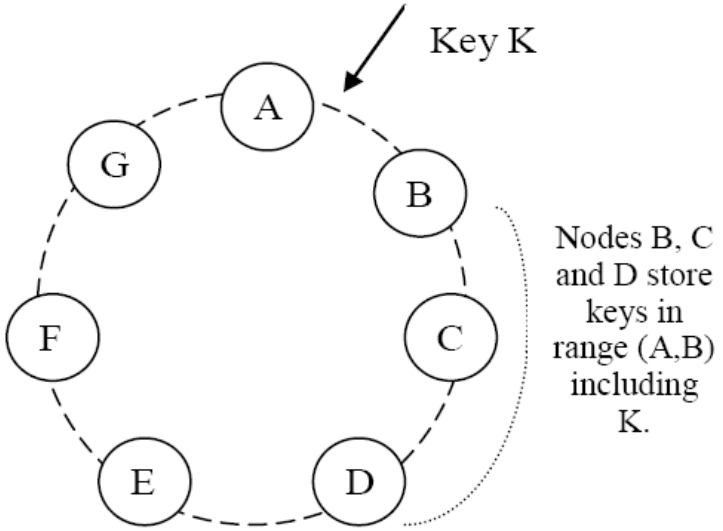
- Good:
  - Impact of adding and removing nodes over system is small
- Bad:
  - Non-uniform data and load distribution
  - Ignorant of diversity in the performance of nodes

Virtual nodes

- Each node gets assigned to multiple points in the ring.
- Good:
  - If a node fails the load on this node will spread across the remaining available nodes.
  - When a node come back again, the new node accepts a same amount of load from each of the other nodes.
  - aware of diversity in performance of nodes

Replication

- Each data item is replicated at N hosts.
- Preference list: The list of nodes for storing replica.



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, they need to be handled in the future.
- 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 object has a vector clock.
- If the counters on the first object's clock are smaller than that in the second clock, then the first is old and can be dismissed.

Execution of get () and put ()

- Dynamic: Load balancer selects a node based on load information then forward the requests
  - Client doesn't have to link any code to Dynamo
- Static: Use a partition-aware client library , send requests directly to the correct prime nodes.
  - Fast, don't need to get load info, don't need to forward anything

Failure tolerance: Sloppy Quorum

- Flex membership

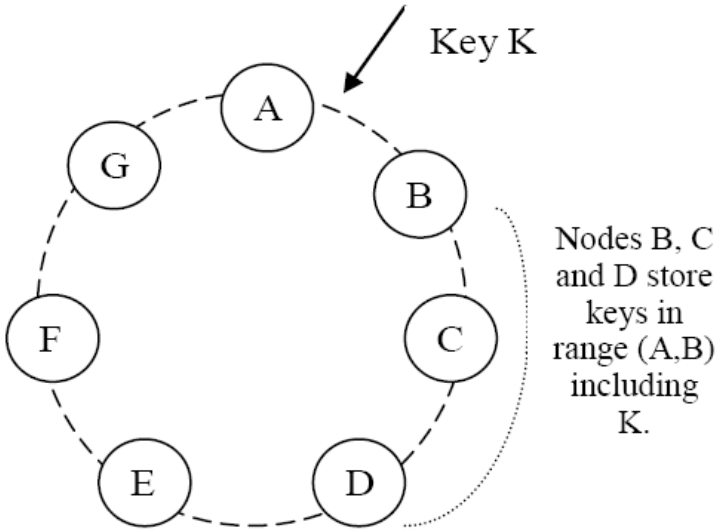
- R/W is the minimum number of nodes that must participate in a successful read/write operation.
- Setting R + W > N yields a quorum-like system.
- R/W number is bigger, then the reliability is better but availability is worse

If replication node fail…

- Preference list: physical node list for storing replica
- Choose first N “healthy” nodes in the list to read/write replica

Failure handling

- Assume N = 3. When A is temporarily down during a write, send replica to D.
- D is hinted that the replica is belong to A and it will send to A when A is recovered.
- Again: “always writeable”



Other techniques

- **Membership**
  - **Gossip protocol**
- **Failure Detection:**
  - **Message response instead of regular heart beat**

Implementation

- Java
- Local persistence component
  - Berkeley Database (BDB) Transactional Data Store
  - MySQL
  - BDB Java Edition

Questions