Big Data Systems

Djellel Eddine Difallah

Spring 2023

Lecture 5 – NOSQL

So far ..

- The relational model is very good for abstracting real world objects and their relationships
- SQL is powerful at declaratively querying such data representation
- RDBMSs take care of data storage, query execution and optimization etc., and are fully ACID compliant
- Some challenges: OLAP vs OLTP workloads
  - Mostly solved at the storage level: column vs row stores
- Distributed DBMSs (new challenges, new techniques etc)

Outline

- Introduction
- Dynamo
- Key distributed systems principles:
  - Consistent Hashing
  - Virtual Nodes
  - Vector clocks
  - Data Versioning

NOSQL Motivation

- Originally motivated by Web 2.0 applications
  - Lots of traffic
  - Users are doing both updates and reads
- The goal is to scale simple OLTP-style workloads to thousands or millions of users
  - Not feasible on a single machine (bottleneck)
  - Need to scale-out
  - RDBMs is not designed for large clusters of machines
- NOSQL has emerged from companies trying to solve critical business problems

“Not Only SQL” or “Not Relational”

Distinctive features:

1. Scale horizontally (many machines)
2. Replicate/distribute data over the server nodes
3. Simple call level interface (contrast w/ SQL)
4. Weaker consistency model than ACID
5. Efficient use of distributed indices and RAM
6. Flexible schema

NOSQL Tradeoffs

- The system is willing to give up
  - Complex queries: e.g., give up on joins
  - Multi-object transactions
  - ACID guarantees:
    - **Eventual consistency** is OK
    - If updates stop, all replicas will converge to the same state and all reads will return the same value

* Not all NOSQL system give all of these

Data Models

In RDBMS

- Record/Tuple = row in a relational db

In NOSQL

- Extensible record = families of attributes have a schema, but new attributes may be added
- Document = nested values, extensible records (think XML, JSON, attribute-value pairs)
- Object = like in a programming language, but without methods

A simple data model stems from simple and very specific sub-application e.g., a shopping cart.

- Joins between customer/item/stock/quantity
- vs a single representation

Different Types of NoSQL

Taxonomy based on data models [Rick Cattell, SIGMOD Record 2011]

- Key-value stores
  - Value is arbitrary (think of it as a scalable hashmap)
  - e.g., Project Voldemort, Memcached, **Dynamo**
- Column Family (a.k.a Extensible Record Stores)
  - Some schema structure
  - e.g., HBase, **Cassandra**, PNUTS
- Document stores
  - Complex data structure (JSON seems to be the standard but can be XML)
  - e.g., SimpleDB, CouchDB, **MongoDB**

NOSQL KEY-VALUE STORES

DYNAMO

Not to be confused with Amazon DynamoDB

Dynamo

- Dynamo: Amazon's Highly Available Key-value Store.
  - By Giuseppe DeCandia et. al. SOSP 2007.
  - *Note: the paper depicts the design principles but not the actual implementation of Amazon DynamoDB*
- Main observations
  - "There are many services on Amazon's platform that only need primary-key access to a data store"
    - Best seller lists, shopping carts, customer preferences, session management, sales rank, product catalog etc.
- Motivation is to build a distributed storage system
  - Scalable
  - Simple: key-value
  - Highly available
  - Guarantee Service Level Agreements (e.g., 99.9% of requests are processed under 300ms, at peak workload)

What is Dynamo?

- A distributed Key-Value store service built on a ring topology **(decentralized architecture)**
  - High availability for Writes
    - Goal to NEVER reject any writes (bad for business)
      - That's why conflict resolution is pushed to reads
  - Eventual Consistency
- Data model: simple (key,value) pairs
  - "Items" are uniquely identified using a key
  - Values are binary objects (blobs) ~ 1MB
  - No further schema
- Operations
  - Insert, delete, and lookup operations on keys
  - No operations across multiple data items

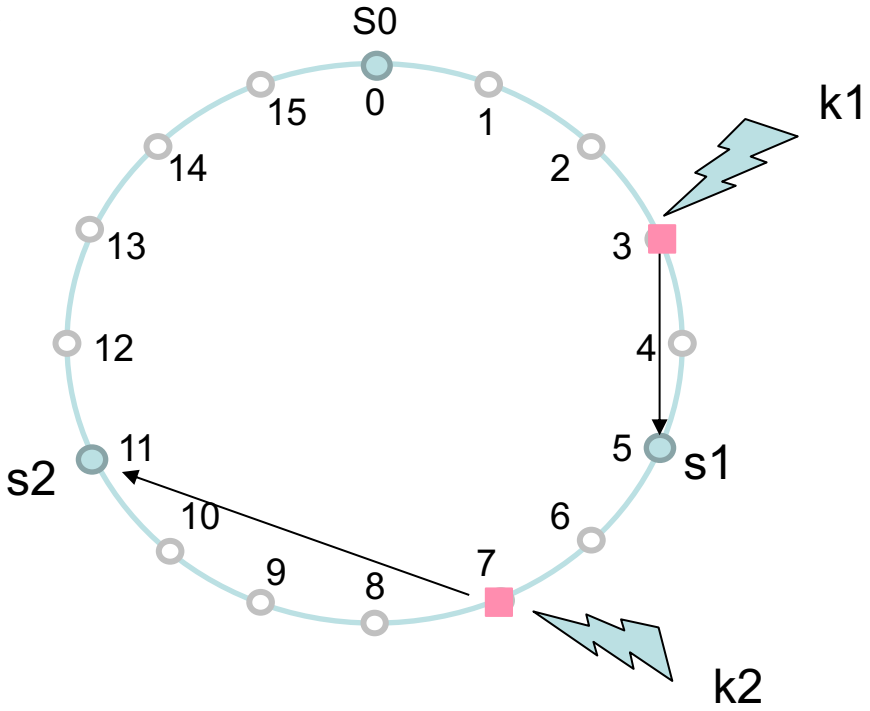
Partitioning: The naïve approach

- On which server shall we put a given key?
  - Select a hash function ***h()***
  - ***server = h(k) % N***
  - Works well, if the number of servers (N) doesn't change

- Problem 1: if a server goes down, we loose data
- Problem 2. if new server joins, then N → N+1, and the entire hash table needs to be reorganized
- Problem 3: we want replication, i.e. store the key/value at more than one server, also retrieve it from arbitrary replica

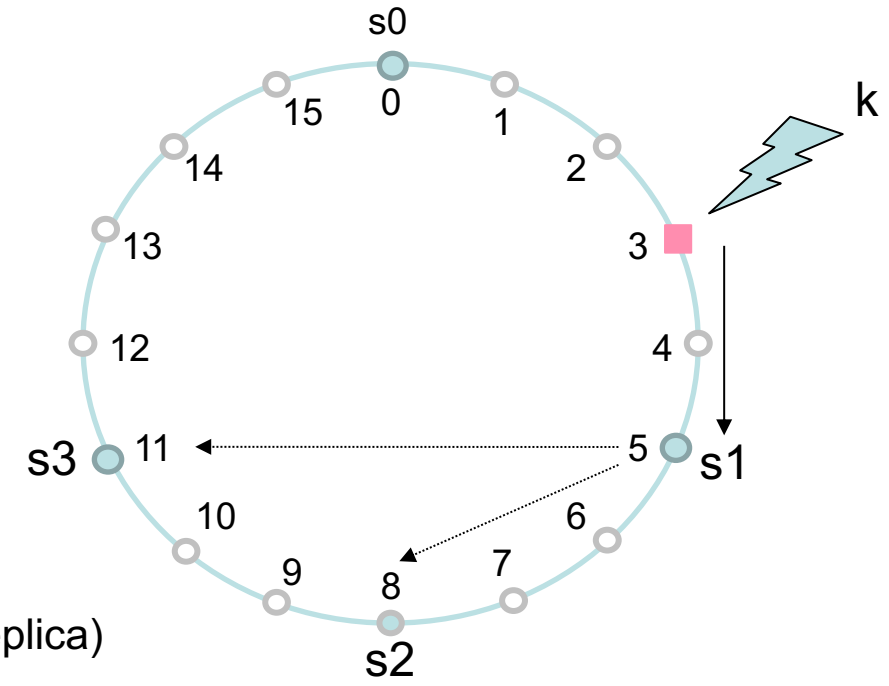
Data Partitioning & Replication

- Use consistent hashing
  - Each node gets an ID from a **key space** (much larger than possible number of nodes)
  - Nodes are arranged in a ring
  - Data stored on the first node clockwise of the current placement of the data key



Data Partitioning & Replication

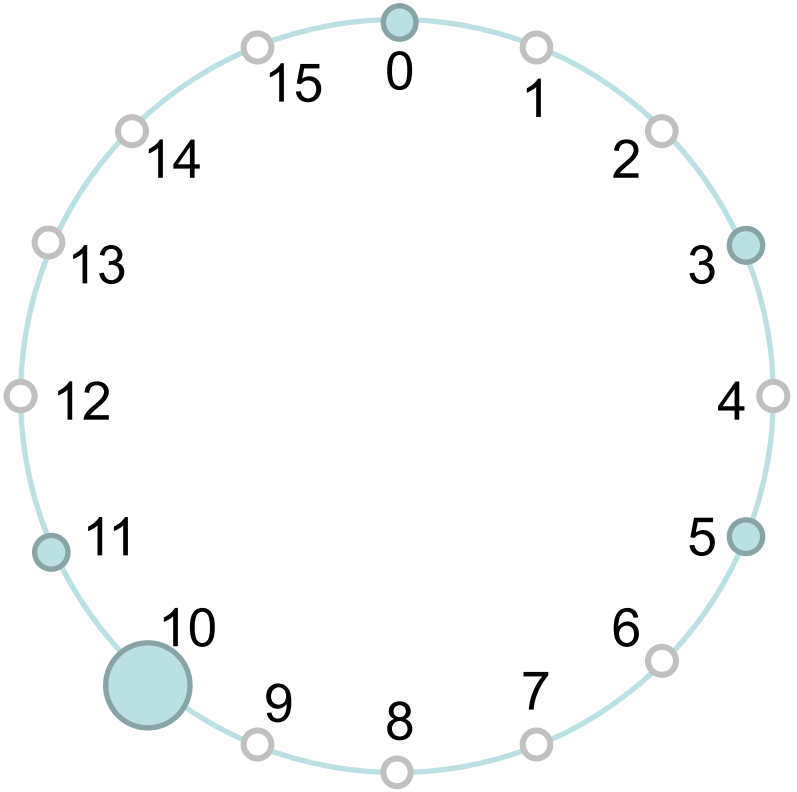
- Replication
  - Set N = desired degree of replication
  - Copy object with key k to *h(k), h(k)+1, ..., h(k)+N-1* along the ring



Example with N = 3
(S1,S2,S3 will store the replica)

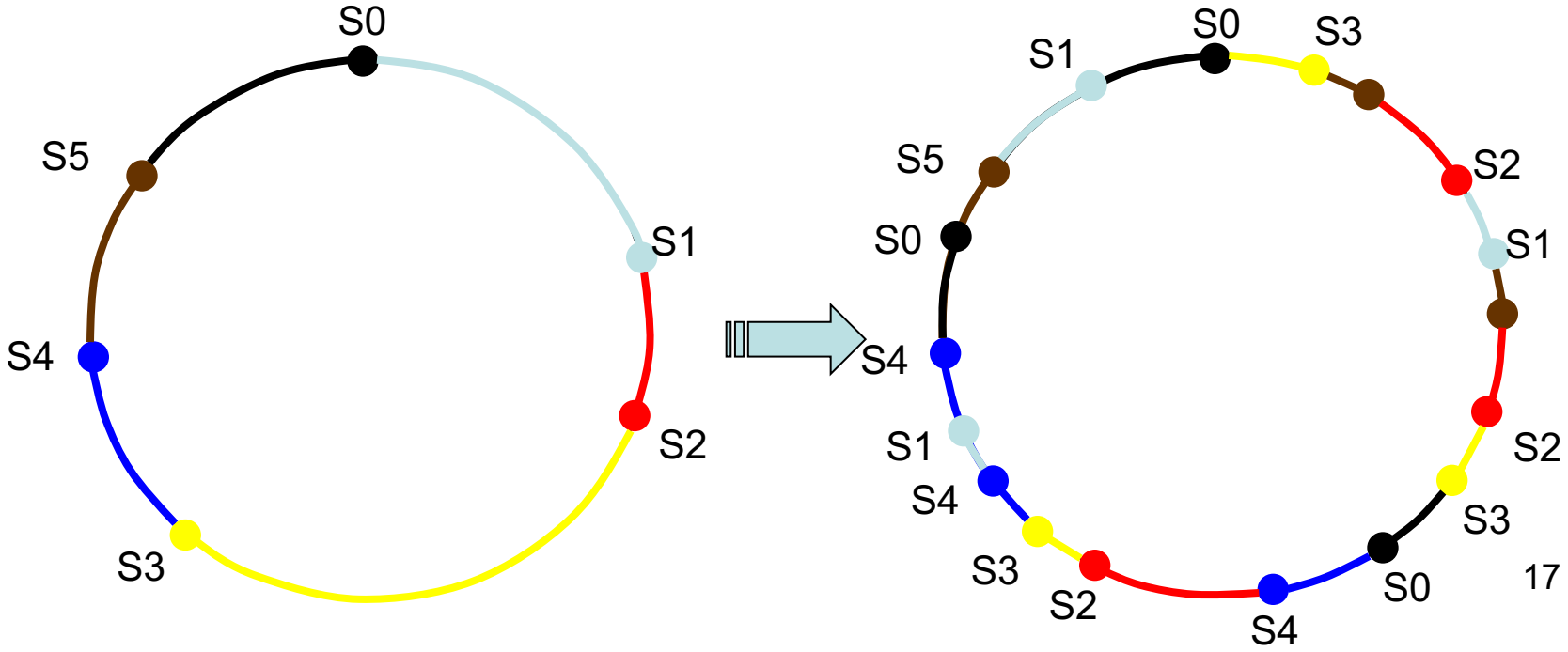
Consistent Hashing Issues

- Random placement on the ring
- Load
  - Storage bits
  - Popularity of the item
  - Processing required to serve the item
- Heterogeneous hardware is not accounted for
- Consistent hashing may lead to imbalance



Virtual Nodes

- Each physical node picks multiple random ring identifiers
  - Each identifier represents a virtual server
  - Each node runs multiple virtual servers
- Each node responsible for noncontiguous regions
- Better load balance



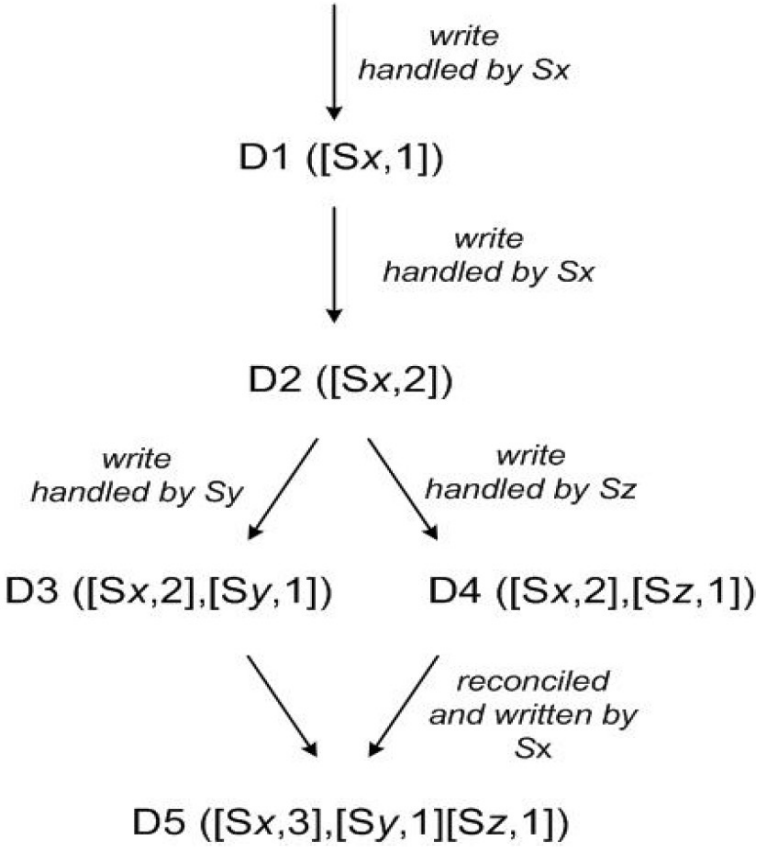
How do we speed up writes? Data Versioning

- Updates generate a new timestamp
- Multiple versions of the same object might co-exist
  - Eventual consistency
- If a conflict arise, fix during Read
  - Syntactic reconciliation
    - The application is able to resolve the conflict automatically
  - Semantic reconciliation
    - Merge results from different conflicts, make the user revise the new values e.g., for Amazon's shopping cart
      - Preserve "Add to cart" items
      - Deleted items can resurface

Data Versioning: Vector Clocks

- Vector clocks are used to detect **update** (write) conflicts in distributed systems
- A vector clock is a set of (Node, Counter) pairs
- One vector clock is associated with every version of every object.
- How it works:
  - E.g. Given an initial state <(A, 3), (B, 1), (C, 2)>
  - If A modifies the associated data, then <(A, 4), (B, 1), (C, 2)>
  - *If all first object's counters are strictly less than or equal to the second clock, the first object happened before the second..*
  - ***Otherwise, conflicts are detected!***

Vector Clocks: Version evolution of an object over time



Example

For a given key *K,* replicated on servers SX, SY, SZ

- A client writes "D1"

  SX: D1 ([SX,1])

- Another client reads D1, writes back D2

  SX: D2 ([SX,2])

➢ Partitioning happens: SZ cannot sync with the other servers!

- A client reads D2, writes back D3

  SY: D3 ([SX,2], [SY,1])

- A client reads D2, writes back D4

  SZ: D4 ([SX,2], [SZ,1])

➢ The system heals

- Dynamo detects and resolves conflicts at read-time by sending all available versions to the client to reconcile.

Example of Client Reconciliation

- An example of semantic reconciliation is the shopping cart feature provided by Amazon.
- This mechanism guarantees that an ‘Add to cart’ operation is never lost, but it is possible that deleted items will resurface.

Example: Conflict or not?

Object A	Object A	Conflict ?
([SX,3],[SY,6])	([SX,3],[SZ,2])	Y
([SX,3])	([SX,5])	N
([SX,3])	([SX,3],[SY,6],[SZ,2])	N
([SX,3],[SY,10], [SZ,1])	([SX,3],[SY,6],[SZ,2])	Y
([SX,3],[SY,10])	([SX,3],[SY,20],[SZ,2])	N

Vector Clocks (Pros and Cons)

- ✓ The sequencing cannot be wrong
- – Push the complexity (and the logic to the client)

Get() and Put() operations

- Set of nodes for a key is its *preference_list*
  - *i.e., replica nodes*
- Requests are routed to the *Coordinator*, One-Hop routing:
  - Each node knows preference list of each key
- Coordinator node is among the top N in the preference list
- Coordinator runs a R/W quorum system
    - R number of nodes that must reply in a successful read operation.
    - W number of nodes that must reply in a successful write operation.
  - Such that: R+W > N

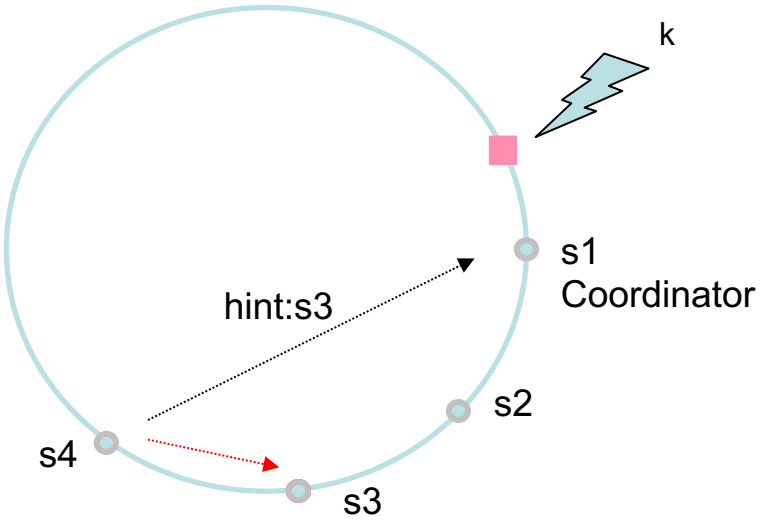
Get() and Put() operations

- Write operations
  - Coordinator generates vector clock & stores locally
  - Coordinator forwards new version to all N replicas
  - If at least W replica nodes respond then success!
- Read operations
  - Initial request sent to coordinator
  - Coordinator requests data from all N replicas
  - Once gets R responses, returns data

Handling (temporary) Failures: Hinted Handoff

- Do not enforce strict quorum membership.
- Sloppy quorum: Involve first N healthy nodes from the preference list
- Data returned from healthy but not first N nodes, contain a hint that this is temporary
- Responsibility is sent back when failed node recovers

Example: put(k)
k's preference list (s1-4)
N = 3
R = 2
W= 3



Summary

Problem	Technique	Advantage
Partitioning (Sharding)	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.
Recovering from permanent failures	Anti-entropy using Merkle trees	Synchronizes divergent replicas in the background.
Membership and failure detection	Gossip-based membership protocol and failure detection.	Preserves symmetry and avoids having a centralized registry for storing membership and node live information.

Reading

- Dynamo: Amazon's Highly Available Key-value Store.
  - By Giuseppe DeCandia et. al. SOSP 2007