Distributed Systems

CS425/ECE428

Instructor: Radhika Mittal

Acknowledgements for the materials: Indy Gupta

Logistics

- HW5 due on April 24.
- MP3 due on April 29

Final Exam: May 2 – May 6

- Comprehensive:
  - Midterm 1 and Midterm 2 syllabus
  - Post-midterm2 content
- Relative weightage:
  - Midterm 1 and Midterm 2 contents: *roughly* 60%
  - Post-midterm2 contents: *roughly* 40%
- No practice questions this time!
  - Refer to practice questions for Midterm 1 and Midterm 2
  - Refer to Midterm 1 and Midterm 2 exams
  - Refer to homework assignments
- New post-midterm 2 content:
  - Homeworks provide some representative questions.
  - Your exam will also have short MCQ-type questions on the new materials.

Today's focus

- Brief overview of key-value stores
- Distributed Hash Tables
  - Peer-to-peer protocol for efficient insertion and retrieval of key-value pairs.
- Key-value stores in the cloud
  - How to run large-scale distributed computations over key-value stores?
    - Map-Reduce Programming Abstraction
    - Cloud Scheduling
  - How to design a large-scale distributed key-value store?
    - Case-study: Facebook's Cassandra

Distributed datastores

- Distributed datastores
  - Service for managing distributed storage.
- Distributed NoSQL key-value stores
  - BigTable by Google
  - HBase open-sourced by Yahoo and used by Hadoop.
  - DynamoDB by Amazon
  - Cassandra by Facebook
  - Voldemort by LinkedIn
  - MongoDB,
  - …
- *Spanner is not a NoSQL datastore. It's more like a distributed relational database.*

How to design a distributed key-value datastore?

Design Requirements

- High performance, low cost, and scalability.
  - Speed (high throughput and low latency for read/write)
  - Low TCO (total cost of operation)
  - Fewer system administrators
  - Incremental scalability
    - Scale out: add more machines.
    - Scale up: upgrade to powerful machines.
    - *Cheaper to scale out than to scale up.*

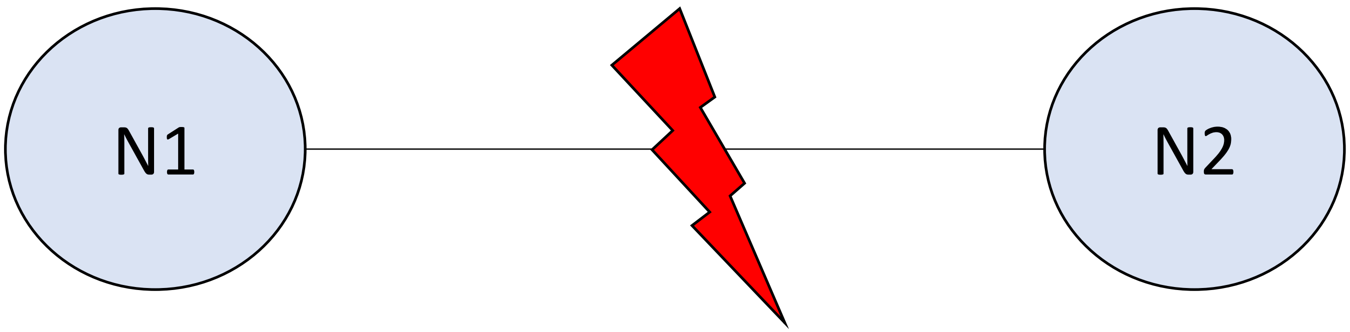
Design Requirements

- High performance, low cost, and scalability.
- Avoid single-point of failure
  - Replication across multiple nodes.
- Consistency: reads return latest written value by any client (all nodes see same data at any time).
  - *Different from the C of ACID properties for transaction semantics!*
- Availability: every request received by a non-failing node in the system must result in a response (quickly).
  - Follows from requirement for high performance.
- Partition-tolerance: the system continues to work in spite of network partitions.

CAP Theorem

- **C**onsistency**:** reads return latest written value by any client (all nodes see same data at any time).
- **A**vailability**:** every request received by a non-failing node in the system must result in a response (quickly).
- **P**artition-tolerance**:** the system continues to work in spite of network partitions.
- **In a distributed system you can only guarantee at most 2 out of the above 3 properties.**
  - Proposed by Eric Brewer (UC Berkeley)
  - Subsequently proved by Gilbert and Lynch (NUS and MIT)

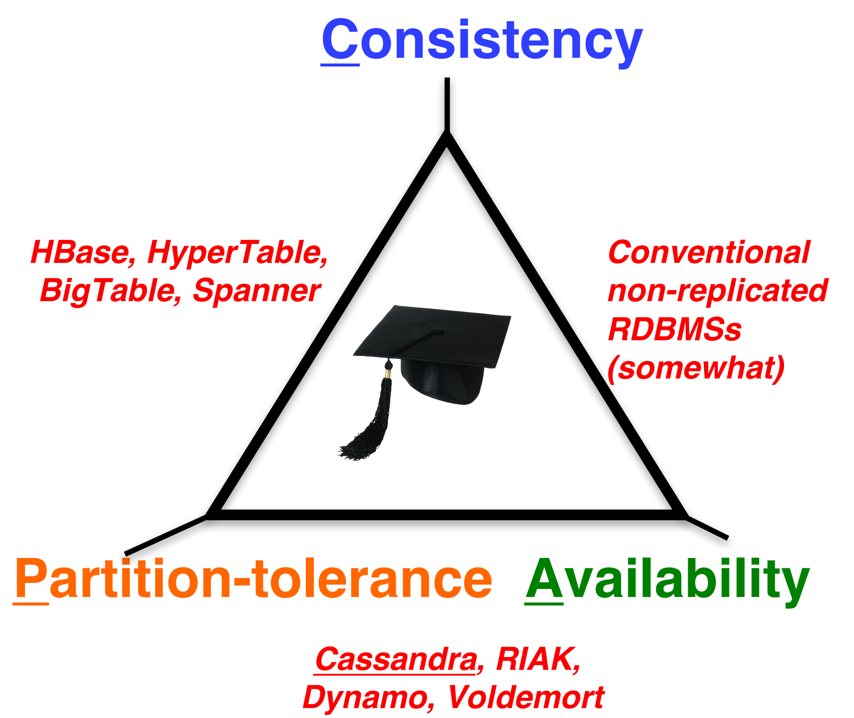
CAP Theorem



- Data replicated across both N1 and N2.
- If network is partitioned, N1 can no longer talk to N2.
- Consistency + availability
  - N1 and N2 must talk (no partition-tolerance).
- Partition-tolerance + consistency:
  - only respond to requests received at N1 (no availability).
- Partition-tolerance + availability:
  - write at N1 will not be captured by a read at N2 (no consistency).

CAP Tradeoff

- Starting point for NoSQL Revolution
- A distributed storage system can achieve at most two of C, A, and P.
- When partition-tolerance is important, you have to choose between consistency and availability



Modern key-value stores vs. RDBMS

- While RDBMS provide ACID
  - Atomicity
  - Consistency
  - Isolation
  - Durability

- Many modern key-value stores provide BASE
  - Basically Available Soft-state Eventual Consistency
  - Prefers Availability over Consistency

Case Study: Cassandra

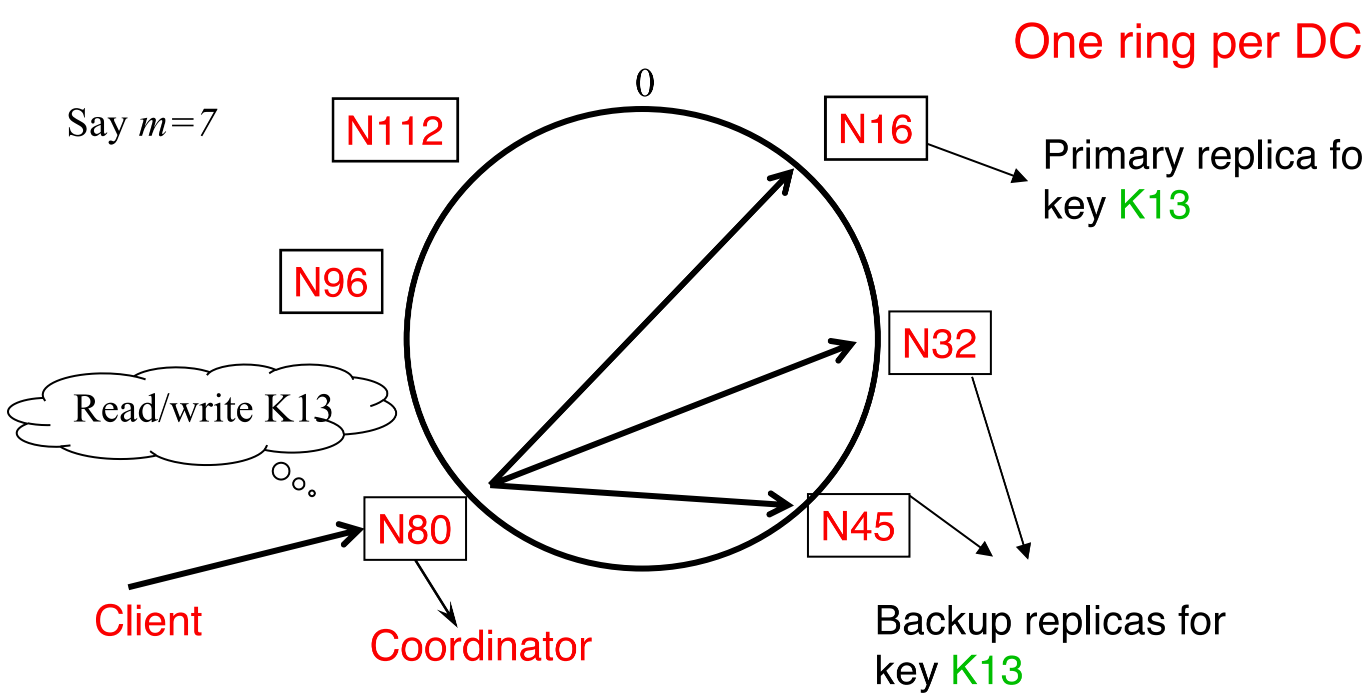
Cassandra

- A distributed key-value store.
- Intended to run in a datacenter (and also across DCs).
- Originally designed at Facebook.
- Open-sourced later, today an Apache project.
- Some of the companies that use Cassandra in their production clusters.
    - IBM, Adobe, HP, eBay, Ericsson, Symantec
    - Twitter, Spotify
    - PBS Kids
    - Netflix

Data Partitioning: Key to Server Mapping

- How do you decide which server(s) a key-value resides on?

Cassandra uses a ring-based DHT but without finger or routing tables.



Partitioner

- Component responsible for key to server mapping (hash function).

- Two types:
  - *Chord-like hash partitioning*
    - *Murmer3Partitioner* (default): uses *murmer3* hash function.
    - *RandomPartitioner*: uses MD5 hash function.
  - *ByteOrderedPartitioner*: Assigns ranges of keys to servers.
    - Easier for *range queries* (e.g., get me all twitter users starting with [a-b])

- Determines the primary replica for a key.

Replication Policies

Two options for replication strategy:

1. SimpleStrategy:
   - First replica placed based on the partitioner.
   - Remaining replicas clockwise in relation to the primary replica.
2. NetworkTopologyStrategy: for multi-DC deployments
   - Two or three replicas per DC.
   - Per DC
     - First replica placed according to Partitioner.
     - Then go clockwise around ring until you hit a different rack.

Writes

- Need to be lock-free and fast (no reads or disk seeks).
- Client sends write to one coordinator node in Cassandra cluster.
  - Coordinator may be per-key, or per-client, or per-query.
- Coordinator uses Partitioner to send query to all replica nodes responsible for key.
- When X replicas respond, coordinator returns an acknowledgement to the client
  - X = any one, majority, all….(consistency spectrum)
  - More details later!

Writes: Hinted Handoff

- Always writable: Hinted Handoff mechanism
  - If any replica is down, the coordinator writes to all other replicas, and keeps the write locally until down replica comes back up.
  - When all replicas are down, the Coordinator (front end) buffers writes (for up to a few hours).

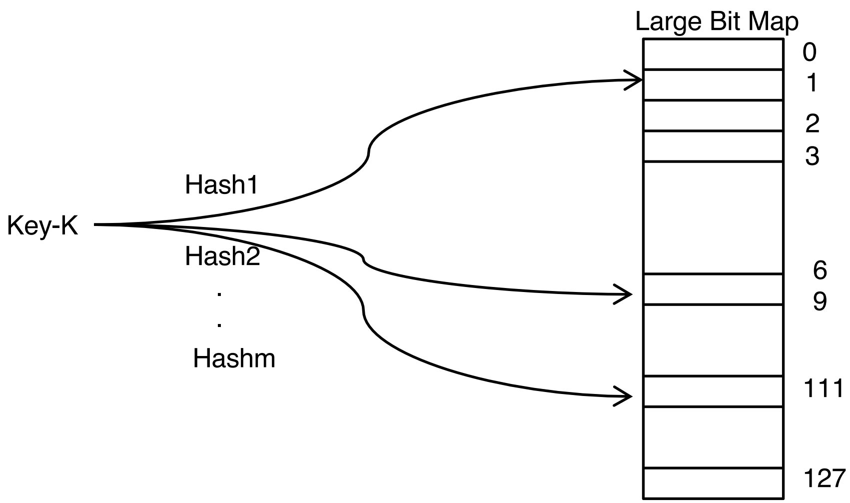
Writes at a replica node

On receiving a write

1. Log it in disk commit log (for failure recovery)
2. Make changes to appropriate memtables
   - **Memtable** = In-memory representation of multiple key-value pairs
   - Cache that can be searched by key
   - Write-back cache as opposed to write-through
3. Later, when memtable is full or old, flush to disk
   - Data File: An **SSTable** (Sorted String Table) – list of key-value pairs, sorted by key
   - Index file: An SSTable of (key, position in data sstable) pairs
   - And a Bloom filter (for efficient search) – next slide.

Bloom Filter

- Compact way of representing a set of items.
- Checking for existence in set is cheap.
- Some probability of false positives: an item not in set may check true as being in set.
- No false negatives.



On insert, set all hashed bits.

On check-if-present,
return true if all hashed bits set.

- False positives

False positive rate low

- m=4 hash functions
- 100 items
- 3200 bits
- FP rate = 0.02%

Writes at a replica node

On receiving a write

1. Log it in disk commit log (for failure recovery)
2. Make changes to appropriate memtables
    - **Memtable** = In-memory representation of multiple key-value pairs
    - Cache that can be searched by key
    - Write-back cache as opposed to write-through
3. Later, when memtable is full or old, flush to disk
    - Data File: An **SSTable** (Sorted String Table) – list of key-value pairs, sorted by key
    - Index file: An SSTable of (key, position in data sstable) pairs
    - And a Bloom filter (for efficient search) – next slide.

Compaction

- Data updates accumulate over time and over multiple SSTables.
- Need to be compacted.
- The process of compaction merges SSTables, i.e., by merging updates for a key.
- Run periodically and locally at each server.

Deletes

Delete: don't delete item right away

- Write a **tombstone** for the key.
- Eventually, when compaction encounters tombstone it will delete item

Reads

- Coordinator contacts X replicas (e.g., in same rack)
  - Coordinator sends read to replicas that have responded quickest in past.
  - When X replicas respond, coordinator returns the latest-timestamped value from among those X.
  - X = based on consistency spectrum (more later).
- Coordinator also fetches value from other replicas
  - Checks consistency in the background, initiating a **read repair** if any two values are different.
  - This mechanism seeks to eventually bring all replicas up to date.
- At a replica
  - Read looks at Memtables first, and then SSTables.
  - A row may be split across multiple SSTables => reads need to touch multiple SSTables => reads slower than writes (but still fast).

Cross-DC coordination

- Replicas may span multiple datacenters.
- Per-DC coordinator elected to coordinate with other DCs.
- Election done via Zookeeper which runs a Bully algorithm variant.

Membership

- Any server in cluster could be the leader.
- So every server needs to maintain a list of all the other servers that are currently in the cluster.
- List needs to be updated automatically as servers join, leave, and fail.

Cluster Membership

Next class!