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
  • How to design a large-scale distributed key-value store?
    • Case-study: Facebook’s Cassandra
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.
Case Study: Cassandra
Recap

• Partitioner: identifies primary replica for a key
  • hash-based or range based.

• Replication in multi-DC environments
  • replicate across datacenters.
  • replicate across different racks within a datacenter.

• Writes:
  • Client send writes to the coordinator.
  • Coordinator sends query to all replicas.
  • Waits for X replicas to respond before returning acknowledgement to client (X determines consistency level. To be discussed.)
  • Hinted handoffs to ensure writes are eventually written to all replicas.
  • At a replica: first log to disk, then write to memtable (in memory).
    • When memtable full or old, flush to SSTable (in permanent storage).
    • Periodic compaction of SSTables.
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 $\Rightarrow$ reads need to touch multiple SSTables $\Rightarrow$ 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

Cassandra uses gossip-based cluster membership

- Nodes periodically gossip their membership list
- On receipt, the local membership list is updated, as shown
- If any heartbeat older than $T_{fail}$, node is marked as failed

Current time: 70 at node 2 (asynchronous clocks)
Consistency Spectrum

Faster reads and writes

More consistency

Eventual → Strong
Eventual Consistency

- Cassandra offers **Eventual Consistency**
  - If writes to a key stop, all replicas of key will converge.
  - Originally from Amazon's Dynamo and LinkedIn's Voldemort systems

---

Faster reads and writes

More consistency

Strong (e.g., Sequential)
Cassandra write and read recap

• Writes
  • Client sends write request to a *coordinator*.
  • Coordinator writes to all replicas.
  • Waits for $X$ replicas to respond before returning acknowledgement to the client.
  • Hinted handoff: if a replica is down, it receives the write request once it comes back up.

• Reads
  • Client sends read request to a *coordinator*.
  • Coordinator contacts $X$ replicas, and returns the latest returned value.
  • Read repair: After returning a response, coordinator continues with fetching values from other replicas, and initiates repairs to outdated values.
Consistency levels: value of $X$

- Cassandra has **consistency levels**.
- Client is allowed to choose a consistency level for each operation (read/write)
  - ANY: any server (may not be replica)
    - Fastest: coordinator caches write and replies quickly to client
  - ALL: all replicas
    - Ensures strong consistency, but slowest
  - ONE: at least one replica
    - Faster than ALL
  - QUORUM: quorum across all replicas in all datacenters (DCs)
Quorums?

In a nutshell:

• Quorum = (typically) majority
• Any two quorums intersect
  • Client 1 does a write in red quorum
  • Then client 2 does read in blue quorum
• At least one server in blue quorum returns latest write
• Quorums faster than ALL, but still ensure strong consistency
• Several key-value/NoSQL stores (e.g., Riak and Cassandra) use quorums.
Read Quorums

• Reads
  • Client specifies value of \( R \) (\( \leq N = \) total number of replicas of that key).
  • \( R = \) read consistency level.
  • Coordinator waits for \( R \) replicas to respond before sending result to client.
  • In background, coordinator checks for consistency of remaining \( (N-R) \) replicas, and initiates read repair if needed.
Write Quorums

- Client specifies $W$ ($\leq N$)
- $W = $ write consistency level.
- Client writes new value to $W$ replicas and returns when it hears back from all.
  - Default strategy.
Quorums in Detail (Contd.)

• R = read replica count, W = write replica count

• Necessary conditions for consistency:
  1. W+R > N
     • Write and read intersect at a replica. Read returns latest write.
  2. W > N/2
     • Two conflicting writes on a data item don’t occur at the same time.

• Select values based on application
  • (W=N, R=1):
    • great for read-heavy workloads
  • (W=1, R=N):
    • great for write-heavy workloads with no conflicting writes.
  • (W=N/2+1, R=N/2):
    • great for write-heavy workloads with potential for write conflicts.
  • (W=1, R=1):
    • very few writes and reads / high availability requirement.
Cassandra Consistency Levels

• Client is allowed to choose a consistency level for each operation (read/write)
  • ANY: any server (may not be replica)
    • Fastest: coordinator may cache write and reply quickly to client
  • ALL: all replicas
    • Slowest, but ensures strong consistency
  • ONE: at least one replica
    • Faster than ALL,
  • QUORUM: quorum across all replicas in all datacenters (DCs)
    • Global consistency, but still fast
  • EACH QUORUM: quorum in every DC
    • Lets each DC do its own quorum (not supported for reads)
  • LOCAL QUORUM: quorum in coordinator’s DC
    • Faster: only waits for quorum in first DC client contacts
Eventual Consistency

• Sources of inconsistency:
  • Quorum condition not satisfied $R + W < N$.
    • $R$ and $W$ are chosen as such.
    • When write returns before $W$ replicas respond.
      • Sloppy quorum: when value stored elsewhere if intended replica is down, and later moved to the intended replica when it is up again.
  • When local quorum is chosen instead of global quorum.

• Hinted-handoff and read repair help in achieving eventual consistency.
  • If all writes (to a key) stop, then all its values (replicas) will converge eventually.
  • May still return stale values to clients (e.g., if many back-to-back writes).
  • But works well when there are a few periods of low writes – system converges quickly.
Cassandra vs. RDBMS

• MySQL is one of the most popular RDBMS (and has been for a while)
• On > 50 GB data
• MySQL
  • Writes 300 ms avg
  • Reads 350 ms avg
• Cassandra
  • Writes 0.12 ms avg
  • Reads 15 ms avg
• Orders of magnitude faster.
Other similar NoSQL stores

• Amazon’s DynamoDB
  • Cassandra’s data partitioning, replication, and eventual consistency strategies inspired from Dynamo.
  • Uses sloppy quorum as the default mechanism for eventual consistency with availability.
  • Uses vector clocks to capture causality between different versions of an object.
  • Dynamo: Amazon’s Highly Available Key-value Store, SOSP’2007.

• LinkedIn’s Voldemort
  • Inspired from DynamoDB.

• ......
Is it a good idea to trade-off consistency for availability?

A recent tweet by a distributed systems researcher:

Due to a shopping cart weak consistency error, my mom has found herself with an extra 4 dozen eggs and 4 pounds of beets she didn't mean to order.

Isn't this what I've been warning everyone about for years?
Summary

• CAP theorem: cannot only achieve 2 out of 3 among consistency, availability, and partition-tolerance.

• Partition-tolerance is required in distributed datastores.
  • Choose between consistency and availability.

• Many modern distributed NoSQL key-value stores (e.g. Cassandra) choose availability, providing only eventual consistency.