Distributed Systems

CS425/ECE428

Instructor: Radhika Mittal

Acknowledgements for the materials: Indy Gupta
Logistics

• HW5 due today.

• MP3 due on April 29.

• Second chance for MP1 functionality and MP2 with 30% penalty (due May 8th)
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
How to design a distributed key-value datastore?
CAP Theorem

- **Consistency**: reads return latest written value by any client (all nodes see same data at any time).
- **Availability**: every request received by a non-failing node in the system must result in a response (quickly).
- **Partition-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 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

Consistency

- HBase, HyperTable, BigTable, Spanner

Partition-tolerance

- Cassandra, RIAK, Dynamo, Voldemort

Availability

Conventional non-replicated RDBMSs (somewhat)
Case Study: Cassandra
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
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
- Eventual
- 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, but cannot tolerate a failure
  - 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. \( W > \frac{N}{2} \)
     - 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.
  - \((W=N/2+1, R=N/2+1)\):
    - 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, and ensures durability without failures
  - 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: supports hierarchical replies
  - 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 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 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.

• ....
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.
Next week

- Monday:
  - Guest lecture by my PhD student, Sachin Ashok
  - *Microservice based cloud applications*

- Wednesday:
  - Q/A session in class (optional attendance)