CLOUD COMPUTING CONCEPTS

with Indranil Gupta (Indy)

KEY-VALUE STORES NoSQL

Lecture C

THE MYSTERY OF X
THE CAP THEOREM

CAP Theorem

- Proposed by Eric Brewer (Berkeley)
- Subsequently proved by Gilbert and Lynch (NUS and MIT)
- In a distributed system you can satisfy at most 2 out of the 3 guarantees:

1. **Consistency**: all nodes see same data at any time, or reads return latest written value by any client
2. **Availability**: the system allows operations all the time, and operations return quickly
3. **Partition-tolerance**: the system continues to work in spite of network partitions

Why is Availability Important?

- Availability = Reads/writes complete reliably and quickly.
- Measurements have shown that a 500 ms increase in latency for operations at Amazon.com or at Google.com can cause a 20% drop in revenue.
- At Amazon, each added millisecond of latency implies a $6M yearly loss.
- SLAs (Service Level Agreements) written by providers predominantly deal with latencies faced by clients.

Why is Consistency Important?

- Consistency = all nodes see same data at any time, or reads return latest written value by any client.
- When you access your bank or investment account via multiple clients (laptop, workstation, phone, tablet), you want the updates done from one client to be visible to other clients.
- When thousands of customers are looking to book a flight, all updates from any client (e.g., book a flight) should be accessible by other clients.

Why is Partition-Tolerance Important?

- Partitions can happen across datacenters when the Internet gets disconnected
  - Internet router outages
  - Under-sea cables cut
  - DNS not working
- Partitions can also occur within a datacenter, e.g., a rack switch outage
- Still desire system to continue functioning normally under this scenario

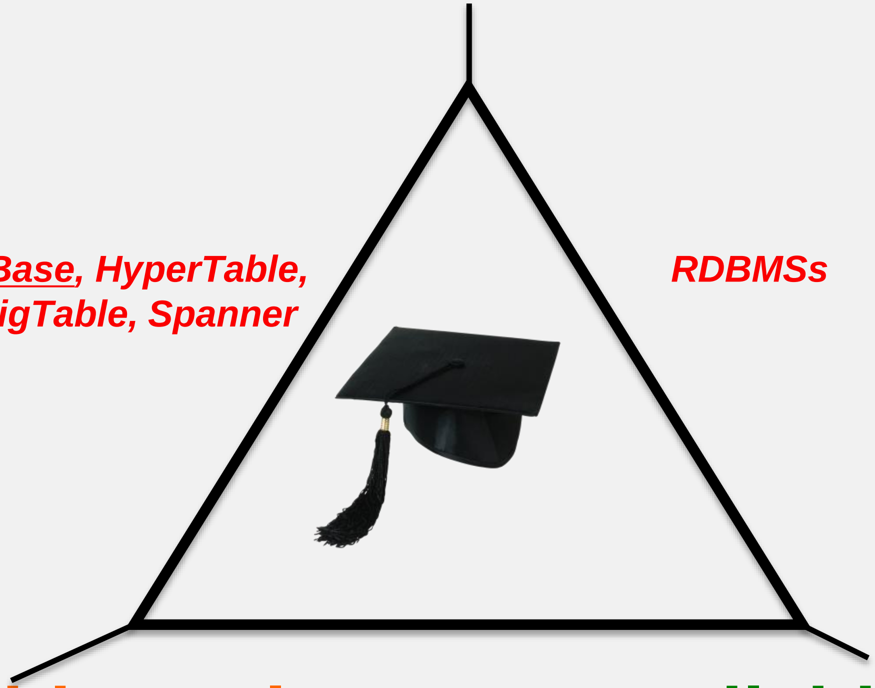
CAP Theorem Fallout

- Since partition-tolerance is essential in today's cloud computing systems, CAP theorem implies that a system has to choose between consistency and availability

- Cassandra
  - Eventual (weak) consistency, availability, partition-tolerance
- Traditional RDBMSs
  - Strong consistency over availability under a partition

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



Eventual Consistency

- If all writes stop (to a key), then all its values (replicas) will converge eventually.
- If writes continue, then system always tries to keep converging.
  - Moving "wave" of updated values lagging behind the latest values sent by clients, but always trying to catch up.
- 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.

RDBMS vs. Key-value stores

- While RDBMS provide ACID
  - Atomicity
  - Consistency
  - Isolation
  - Durability
- Key-value stores like Cassandra provide BASE
  - Basically Available Soft-state Eventual consistency
  - Prefers availability over consistency

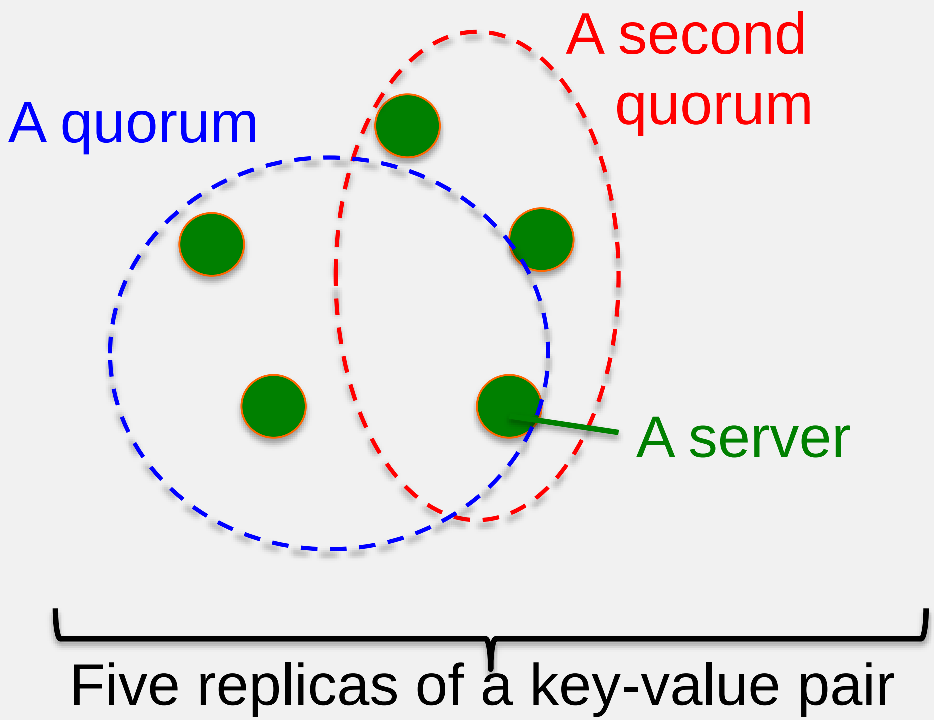
Back to Cassandra: Mystery of X

- 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)
    - What?

Quorums?

In a nutshell:

- Quorum = majority
  - > 50%
- 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



Quorums in Detail

- Several key-value/NoSQL stores (e.g., Riak and Cassandra) use 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.

Quorums in Detail (Contd.)

- Writes come in two flavors
  - Client specifies W ($\leq$ N)
  - W = write consistency level.
  - Client writes new value to W replicas and returns. Two flavors:
    - Coordinator blocks until quorum is reached.
    - Asynchronous: Just write and return.

Quorums in Detail (Contd.)

- R = read replica count, W = write replica count
- Two necessary conditions:
  1. W+R > N
  2. W > N/2
- Select values based on application
  - (W=1, R=1): very few writes and reads
  - (W=N, R=1): great for read-heavy workloads
  - (W=N/2+1, R=N/2+1): great for write-heavy workloads
  - (W=1, R=N): great for write-heavy workloads with mostly one client writing per key

Cassandra Consistency Levels (Contd.)

- 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
  - LOCAL_QUORUM: quorum in coordinator's DC
    - Faster: only waits for quorum in first DC client contacts
  - EACH_QUORUM: quorum in every DC
    - Lets each DC do its own quorum: supports hierarchical replies

Types of Consistency

- Cassandra offers eventual consistency
- Are there other types of weak consistency models?