

CS 425 / ECE 428

Distributed Systems

Fall 2026

Aishwarya Ganesan

w/ Ram Kesavan

Lecture 9-10: Key-Value/NoSQL Stores

Ack: Indranil Gupta

All slides © IG

Announcements

- HW2 Released, due on Sunday 10/4 in less than 2 weeks (start early, no extensions!)
- MP2 is already out, due soon 9/27 (no extensions!)
- MP1 grades released (see Piazza post)
- HW1 being graded
- In class midterm October 8th
 - Make sure you're in class that day!
 - There will be two rooms, different buildings (will update on Piazza and course webpage soon)



The Key-value Abstraction

- (Business) Key → Value
- (twitter.com) tweet id → information about tweet
- (amazon.com) item number → information about it
- (kayak.com) Flight number → information about flight, e.g., availability
- (yourbank.com) Account number → information about it

The Key-value Abstraction (2)

- It's a dictionary datastructure.
 - Insert, lookup, and delete by key
 - E.g., hash table, binary tree
- But distributed.
- Sound familiar? Remember Distributed Hash tables (DHT) in P2P systems?
- It's not surprising that key-value stores reuse many techniques from DHTs.

Isn't that just a database?

- Yes, sort of
- Relational Database Management Systems (RDBMSs) have been around for ages
 - E.g., MySQL
- Schema-based, i.e., structured tables
- Each row (data item) in a table has a primary key that is unique within that table
- Queried using SQL (Structured Query Language)
- Supports joins

Relational Database Example

users table

user_id	name	zipcode	blog_url	blog_id
101	Alice	12345	alice.net	1
422	Charlie	45783	charlie.com	3
555	Bob	99910	bob.blogspot.com	2

↑
Primary keys

↑
Foreign keys

blog table

id	url	last_updated	num_posts
1	alice.net	5/2/14	332
2	bob.blogspot.com	4/2/13	10003
3	charlie.com	6/15/14	7

Example SQL queries

1. `SELECT zipcode
FROM users
WHERE name = "Bob"`
2. `SELECT url
FROM blog
WHERE id = 3`
3. `SELECT users.zipcode, blog.num_posts
FROM users JOIN blog
ON users.blog_url = blog.url`

users.zipcode	blog.num_posts
12345	332
45783	7
99910	10003

Mismatch with today's workloads

- Data: Large and unstructured
- Lots of random reads and writes
- Sometimes write-heavy
- Foreign keys rarely needed
- Joins infrequent

Needs of Today's Workloads

- Speed
- Avoid Single point of Failure (SPoF)
- Low TCO (Total cost of operation/ownership)
- Fewer system administrators
- Incremental Scalability
- Scale out, not up
 - What?

Scale out, not Scale up

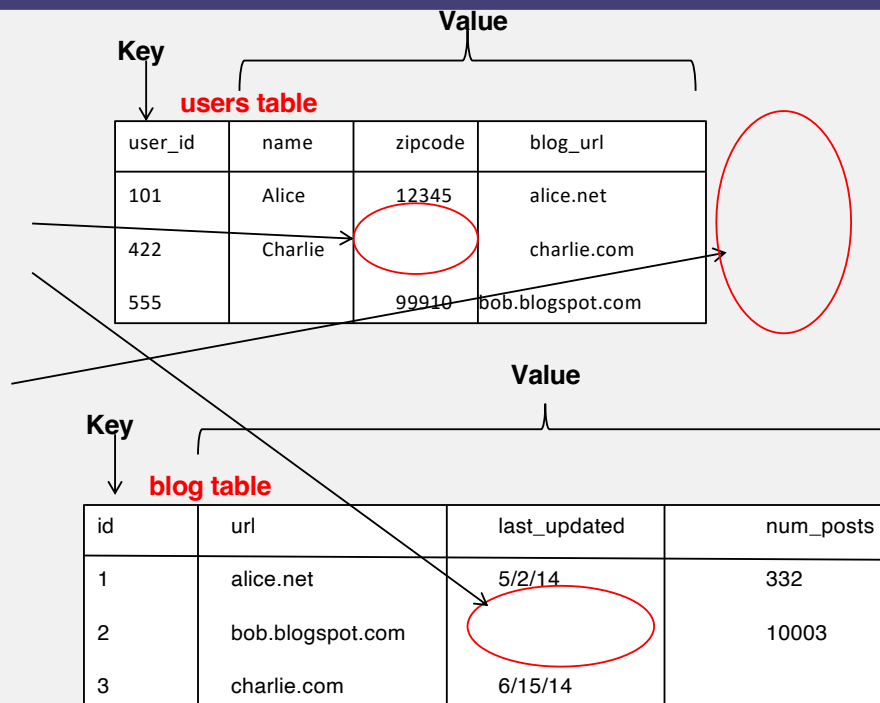
- Scale up = grow your cluster capacity by replacing with more powerful machines
 - Traditional approach
 - Not cost-effective, as you're buying above the sweet spot on the price curve
 - And you need to replace machines often
- Scale out = incrementally grow your cluster capacity by adding more COTS machines (Components Off the Shelf)
 - Cheaper
 - Over a long duration, phase in a few newer (faster) machines as you phase out a few older machines
 - Used by most companies who run datacenters and clouds today

Key-value/NoSQL Data Model

- NoSQL = “Not Only SQL”
- Necessary API operations: `get(key)` and `put(key, value)`
 - And some extended operations, e.g., “CQL” in Cassandra key-value store
- Tables
 - “Column families” in Cassandra, “Table” in HBase, “Collection” in MongoDB
 - but ...
 - May be unstructured: May not have schemas
 - Some columns may be missing from some rows
 - Don’t always support joins or have foreign keys
 - Can have index tables, just like RDBMSs

Key-value/NoSQL Data Model

- Unstructured
- Columns Missing from some Rows
- No schema imposed
- No foreign keys, joins may not be supported



CS 425 / ECE 428

Distributed Systems

Fall 2026

Aishwarya Ganesan

w/ Ram Kesavan

Lecture 9-10: Key-Value/NoSQL Stores

Ack: Indranil Gupta

All slides © IG

Some history...

Earlier: RDBMS

- SQL, ACID transactions, enterprise workloads
- Systems: Oracle, DB2, SQL Server, MySQL

~late '00s and '10s: Rise of NoSQL

- Large web companies build distributed stores to handle massive scale
- Focus on partitioning, replication, high availability, and simpler data models
- Systems: Google Bigtable, Amazon Dynamo, **Cassandra, HBase, MongoDB**

Today: Converging

- SQL databases increasingly support horizontal scale-out (cloud databases)
- NoSQL databases increasingly add transactions and richer query capabilities
- Systems: Spanner, CockroachDB, distributed PostgreSQL, DynamoDB

Agenda

Design of a real key-value store, Cassandra.

- Replication

- Consistency vs. Availability tradeoff

- Quorums

- Local storage

HBase, another KV store and Consistency models (subsequent lectures)

Cassandra

- A distributed key-value store
- Intended to run in a **datacenter** (and also across DCs)
- Originally designed at Facebook (influenced by Amazon Dynamo [1])
- Open-sourced later, today an Apache project [2]
- Widely used in production
 - IBM, Adobe, HP, eBay, Ericsson, Symantec, Twitter, Spotify, PBS Kids, Netflix
 - Netflix: uses Cassandra to keep track of your current position in the video you're watching
- (Version from 2015)

[1] <https://www.allthingsdistributed.com/files/amazon-dynamo-sosp2007.pdf>

[2] <https://github.com/apache/cassandra>

Let's go Inside Cassandra:

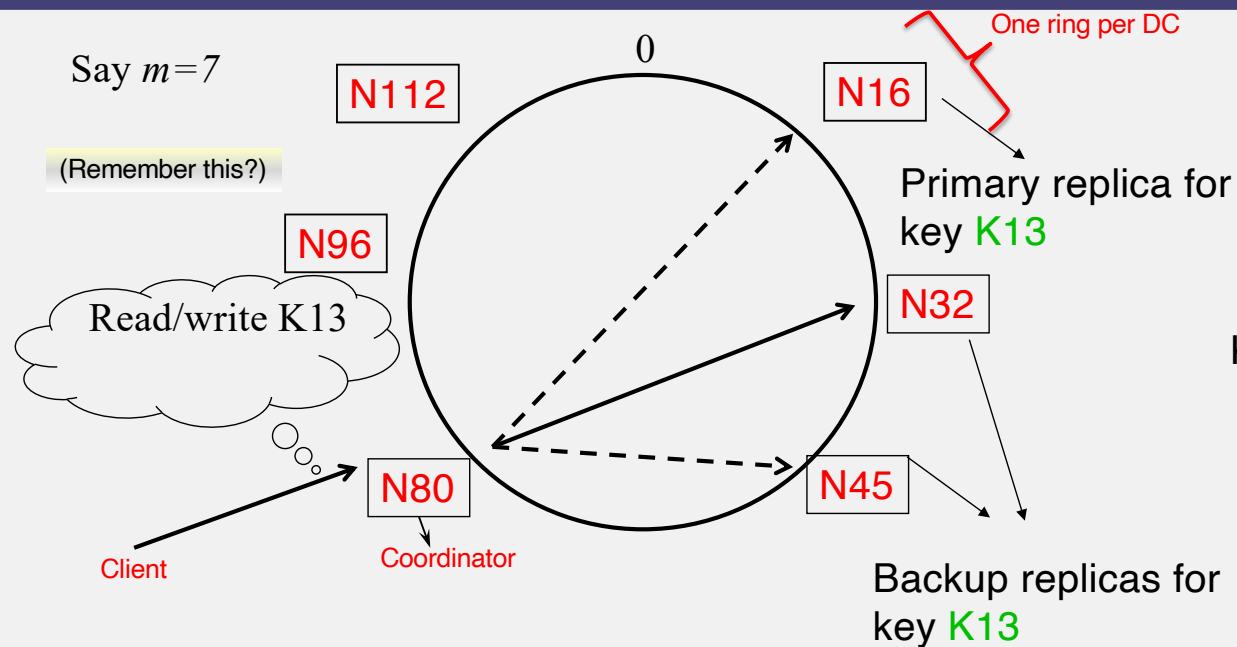
Key -> Server Mapping

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

? Why okay to store full list at all nodes in Cassandra but not Chord ?

? Routing hops?

Key replicated on its r successors
 r -replication factor



Cassandra uses a Ring-based DHT but without finger tables
Each node has the full membership list

Data Placement Strategies

Replication Strategy: two options:

1. SimpleStrategy: uses the Partitioner, of which there are two kinds
 1. *RandomPartitioner*: Chord-like hash partitioning
 2. *ByteOrderedPartitioner*: Assigns ranges of keys to servers.
 - Easier for range queries (e.g., Get me all twitter users starting with [a-b])
2. NetworkTopologyStrategy:
 - E.g., Two replicas per DC
 - Per DC
 - First replica placed according to Partitioner
 - Then go clockwise around ring until you hit a different rack
 - Cassandra uses a component called **snitches** to map IPs to racks and DCs

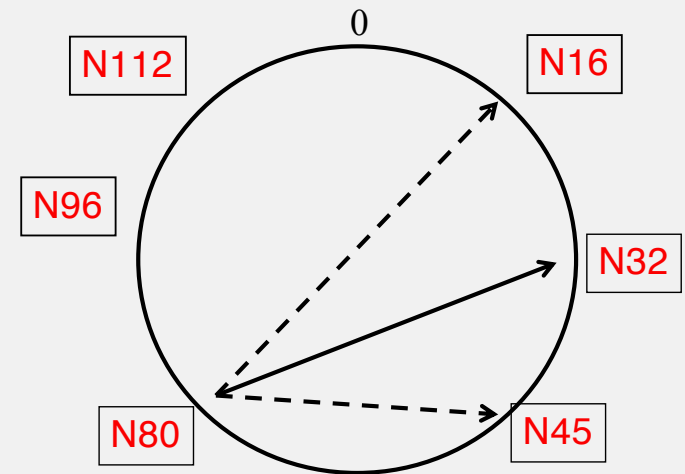
Writes

- Client sends write to one coordinator node in Cassandra cluster
 - Coordinator may be per-key, or per-client, or per-query
 - Typically, the successor of key in the ring
- Coordinator assigns its current timestamp for the write (or client supplies it)
 - Per-key Coordinator ensures writes for the key are ordered
- Coordinator sends write to all replica nodes responsible for key (let's call it **N** from now)
- A replica records the write to its *local storage* – We'll see later.
- When first **X** replicas respond, coordinator acknowledges client
 - *X*? We'll see later.
- Note that multiple writes to same key **K** can be in progress at the same time.

Writes – Hinted Handoff

Always writable: Hinted Handoff mechanism

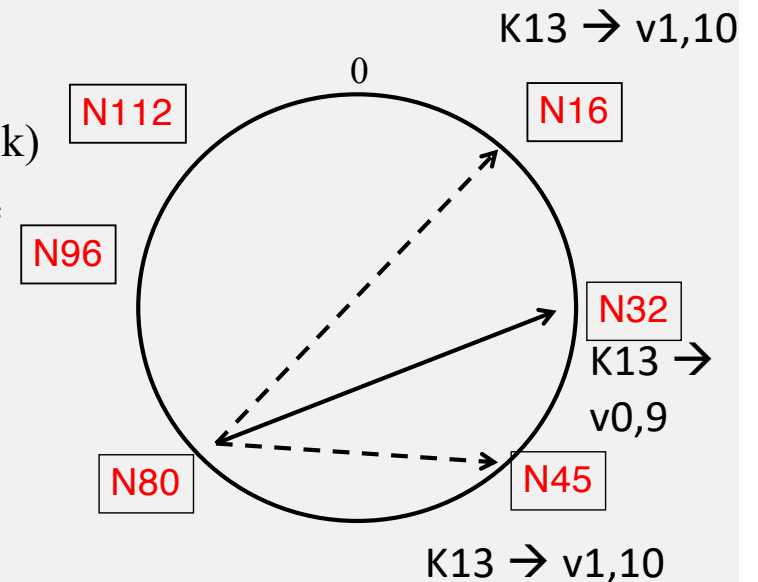
- If any replica is down, the coordinator writes to all other replicas
 - And keeps the write locally until down replica comes back up.
 - Catch up replica after it is up
-
- When all replicas are down, the Coordinator buffers writes (for up to a few hours).



Reads

Read: Similar to writes

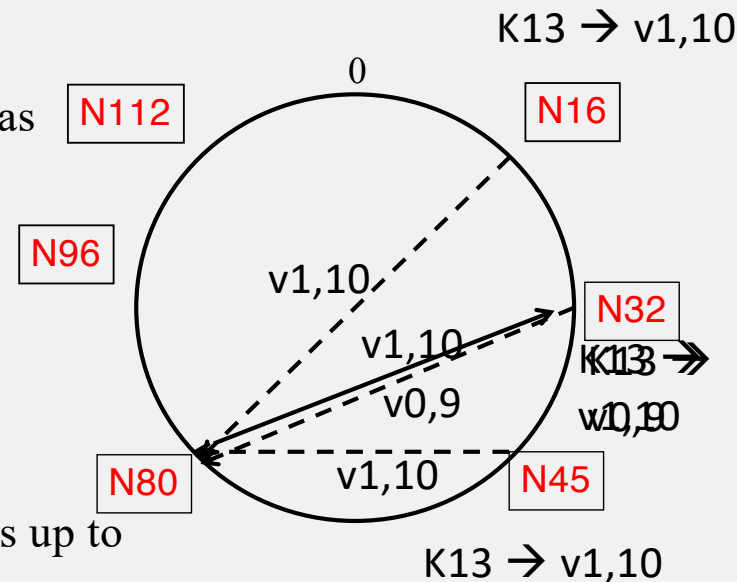
- Coordinator can contact X replicas (e.g., in same rack)
 - Coordinator sends read to replicas that have responded quickest in past
 - Or, send to all N and wait for first X
- When X replicas respond, coordinator returns the **latest-timestamped** value from among those X
 - (X? We'll see later.)



Reads – read repair

After X responses, in the **background**

- Coordinator also fetches value from other replicas
 - Checks consistency
 - Initiates a **read repair** if any two values are different
 - To install the latest value
- This mechanism seeks to eventually bring all replicas up to date



Membership

- Any server in cluster could be the coordinator
- Every server maintains a full list of all other servers
 - updated automatically as servers join, leave, fail
- Cassandra uses gossip-based cluster membership
- One ring per datacenter
 - Per-DC leader elected to coordinate with other DCs
 - Election done via Zookeeper, which runs a Paxos (consensus) variant (Paxos: elsewhere in this course)
 - Leader tells new nodes what key ranges they are replicas for

Failure detection

- Use heartbeat-style Failure Detector (uses Φ Accrual detector)
 - Adaptively sets timeout based on underlying network behavior
 - Entry not deleted on failure (unlike fail-stop model)
 - Cassandra uses fail-recover model and so nodes just marked down/up



Respond at

pe.app/aishwaryaganesa

Poll Everywhere

N80 stops receiving fresh heartbeat information about N45, and its ϕ -accrual detector crosses the failure threshold. Meanwhile, N16 continues receiving heartbeats from N45. Which conclusion is justified?

Responses hidden

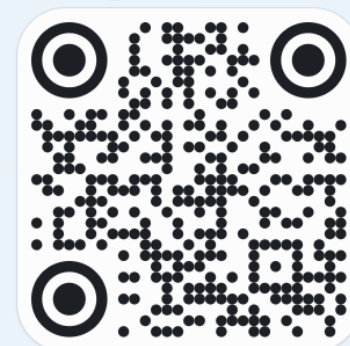


Anonymous 100

N80 can treat N45 as DOWN locally while N45 retains its ownership of keys in the range.

Crossing the threshold automatically removes N45 and reassigns its keys.

N45 must have crashed; N16's observations cannot be valid.



+ Add choice

[Manual flow](#)

☒ Choices

☐ Results

☐ Lock

☐ Correctness

Mystery of “X”: 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.
 - More recent measurements: Amazon: 100 ms increase $\rightarrow$ 1% drop in sales. Akamai: 100 ms $\rightarrow$ 7% drop.
- At Amazon, each added millisecond of latency implies a \$6M yearly loss.
 - More recent measurements: Amazon: 1 s $\rightarrow$ \$1.6 B.
- User cognitive drift: If more than a second elapses between clicking and material appearing, the user's mind is already somewhere else
- 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?

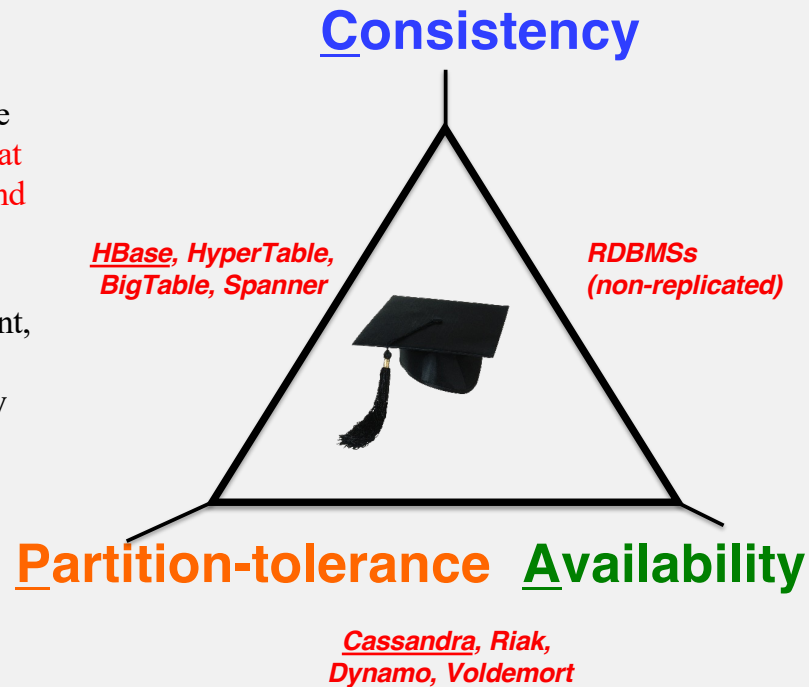
- Partitions can happen across datacenters when the Internet gets disconnected
 - Internet router outages
 - Under-sea cables cut
 - DNS not working
- Partitions can also occur within a datacenter, e.g., a rack switch outage
- Still desire system to continue functioning normally under this scenario

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



What mechanisms in Cassandra we saw earlier helps with eventual convergence?

- 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

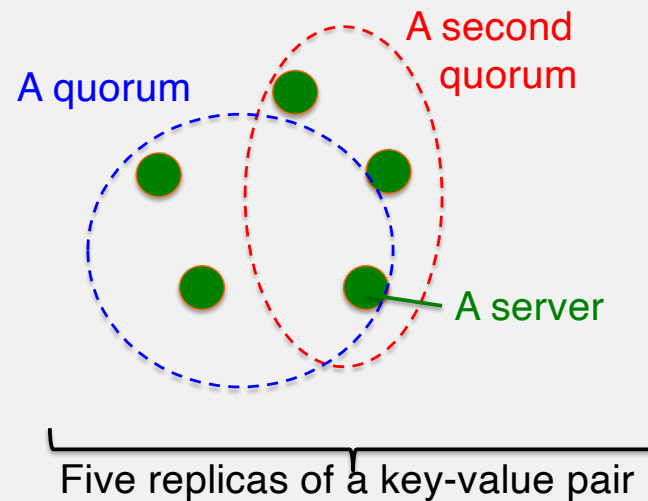
Back to Cassandra: Mystery of X

- 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
 - QUORUM: quorum across all replicas in all datacenters (DCs)
 - What?

Quorums?

In a nutshell:

- Quorum = majority
 - > 50%
- **Any two quorums intersect** no matter how you select the quorums
 - 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 (returns the latest-timestamped value)
 - In background, coordinator checks for consistency of remaining $(N-R)$ replicas, and initiates read repair if needed.

Quorums in Detail (Contd.)

- Writes
- Client specifies W ($\leq N$)
 - W = write consistency level.
 - Coordinator writes new value to W replicas and returns.
 - Coordinator blocks until W replicas perform the write

Quorums in Detail (Contd.)



Does the system choose consistency or availability upon partitions?

1. $W=N, R=1$
2. $W=R=QUORUM$
3. $W=R=ANY$

- R = read replica count, W = write replica count
- Two necessary conditions for consistency:
 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



Respond at
pe.app/[aishwaryaganesa](#)

Poll Everywhere

K13 has three replicas: N16, N32, and N45. A QUORUM write of v2 succeeds at N16 and N32 while N45 is down. N45 recovers before receiving its hint. A subsequent QUORUM read contacts N32 and N45, receiving v2 and v1, respectively. Assume no other writes and that v2 has the newer timestamp. What happens?

Responses hidden



Anonymous 3

The read fails because the replicas disagree.

The coordinator must contact N16 because a quorum requires two identical values.

The read must wait for hinted handoff to complete.

The coordinator can return v2, the latest timestamped value



Add choice

[Manual flow](#)

☒ Choices

☐ Results

☐ Lock

☐ Correctness



Respond at

pe.app/[aishwaryaganesa](https://pe.app/aishwaryaganesa)

Poll Everywhere

What can you say about the consistency and availability for the following configurations of R and W:

1. $W=N, R=1$
2. $W=R=QUORUM$
3. $W=R=1$

22%

Only configuration 1 preserves consistency; configuration 2 requires every replica to respond.

14%

All three preserve consistency because Cassandra eventually synchronizes replicas.

21%

Two and three are available because they don't write to all nodes

43%

Configurations 1 and 2 preserve consistency but not available; 3 is available but may return stale values.

Add choice

[Manual flow](#)

☒ Choices

☒ Results

☐ Lock

☒ Correctness



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
 - **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

Announcements

- (All) HW2 due 10/4 @ 11:59 pm (Sunday!) US Central (hard deadline)
- (All) Midterm Thursday (10/8): Written, in class
- (4cr) **MP2 Demo, Signup, and Submission instructions release on Piazza**
 - ✓ (Due by Fri) [MP Demo Signup Sheet]
 - ✓ (Due by Sun) [a. MP Submission Form & push MP code to GitHub]
 - ✓ (Due by Sun 13) [MP Report Submission on Gradescope]
 - ✓ (On Mon) MP Demos

Local storage at a replica – Writes



How slow is a disk seek compared to DRAM?

- a. 5x b. 50x
- c. 5000x d. **50000x**

Need to be lock-free and fast (no reads or disk seeks)

On receiving a write

1. Log it in disk commit log (for failure recovery)

Writes to a data structure called Log-Structured Merge-tree (LSM)

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

LSM animation – see additional slides