12. Real-World Considerations

Obervations

- CAP theorem puts a natural limit on classical distributed transactions
- Maintaining consistency very expensive due to large number of messages (high latency)
- Web-facing datamanagement poses new challenges:
  - Large scalability (towards billions of users)
  - High availability (24x7 operations, global use)
  - Low response times
  - Write-intensive operations (shopping baskets, social media)

Approaches

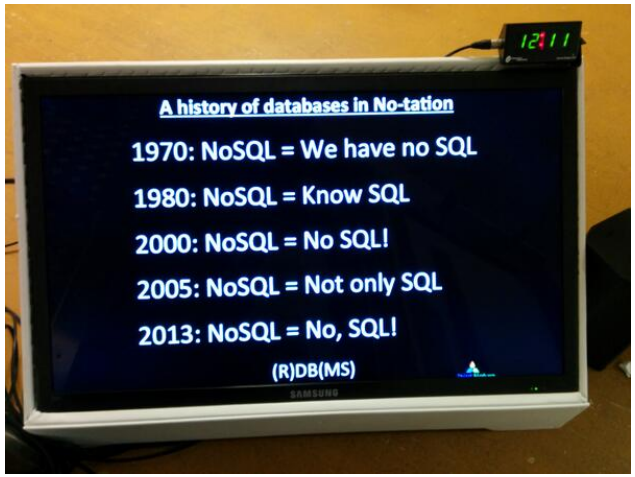
- NoSQL systems (2005 - now): simple data model, limited operations, reduced consistency
- NewSQL systems (2010 - now): SQL+ACID+scalability

$\Longrightarrow$ very active area in research and product development

12.1: NoSQL

Overview

- Catch-all phrase for basically all non-relational and/or non-ACID systems
- Very promiment subclass: Distributed Key/Value-Stores
- automatic partitioning and replication
- relaxed and tuneable consistency: trading off availability and consistency
- Examples: Apache Cassandra, MongoDB, ...

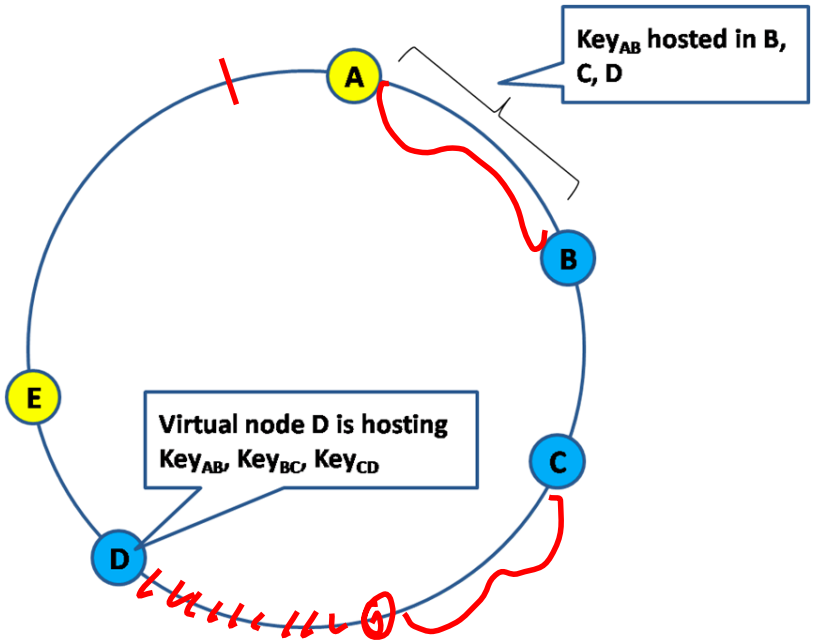


Cassandra: Background and Concepts

- Based on Amazon Dynamo/Google BigTable ideas
- Open-source software
- Key-value store distributed across nodes by key
  - Not a relational table with many column, many access possibilities
  - Instead a key→value mapping like in a hash table
- A value can have a complex structure as it is inside the node - in Cassandra it is columns and super columns

Partitioning and Replication

- Consistent Hashing: Hash Keys and node IDs mapped to (same) circled space
- Each node covers a segment of the rings
- Easy additions/removal and balancing
- content replicated on N subsequent nodes of the ring



Consistency Levels

- CAP theorem allows 2 out of 3
  - (C)onsistency
  - (A)vailability
  - (P)artition tolerance
- Options
  - CA: corruption possible
  - CP: not available if any nodes are down/blocked
  - AP: always available but clients may not always read most recent updates
- Most systems provide CP or AP (why not CA?)
- Cassandra prefers AP but makes "C versus A" configurable by allowing the user to specify a consistency level for each operation
- Consistency levels are handled by setting the quorum for read and write operations

Dealing with eventual consistency

- When W $<$ N (not all replicas are updated) the update is propagated in background
- Version resolution:
  - Each value in a database has a timestamp $\Rightarrow$ key, value, timestamp
  - The timestamp is the timestamp of the latest update of the value (the client must provide a timestamp with each update)
  - When an update is propagated, the latest timestamp wins
- There are two mechanisms to propagate updates:
  - Read repair: hot keys
  - Anti-Entropy: cold keys

Read Repair

- Perform read on multiple replicas.
- Perform reconciliation (e.g. pick the most recent)
- Update all read replicas to the chosen version

Anti-Entropy

- AE is used to repair cold keys - keys that have not been read, since they were last written
- AE works as follows:
  - It generates Merkle Trees for tables periodically
  - These trees are then exchanged with remote nodes using a gossip protocol
  - When ranges in the trees disagree, the corresponding data are transferred between replicas to repair those ranges

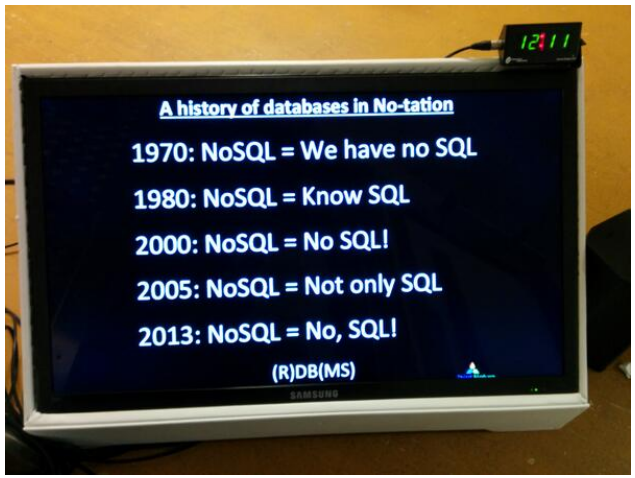
  N.B. Merkle Tree (or hash trees) are a compact representation of data for comparison:

  - A Merkle tree is a hash tree where leaves are hashes of individual values.
  - Parent nodes higher in the tree are hashes of their respective children.

12.2: NewSQL

Overview

- Absence of transactional guarantees cumbersome for developers
- Focus on getting as many SQL/ACID properities while providing (close to) NoSQL scalability
- Main directions:
  1. speeding up/scaling up closely-coupled systems
  2. "Tweaking" consistency models and coordination
  3. clever implementations



H-Store/Volt-DB: Concepts

- Aimed at transaction-heavy workloads, providing ACID and high scalability
- Breaks several assumptions how do DB design with updates
- Aimed at small-scale clusteres (no latency penalty), but providing very high speeds

Design Considerations

- Main-Memory Databases, since most transactional workloads need less than 1 TB
- Single-Thread per core execution model
  - No delays from disk I/O
  - No long-running transactions allowed: typical update queries take few milliseconds!
  - reduces synchronisation cost
- Availability via replication, not log shipping

Implementation

- Shared-nothing architecture over a cluster
- Further shared-nothing decomposition among CPU cores
  - dedicated data structures: tables, indexes
  - transactions run sequentially/serial on a core
- Transactions are known in advance
  - Expressed as stored procedures
  - Information on data access drives scheduling and replication
- Transactions are executed as much as possible on a single site

$\Longrightarrow$ very high transaction rates: several 100K transactions per second per node

Google Spanner/F1

- Massively distributed database, aimed at million servers over the whole world
- Provide synchronous replication
- ACID-style transactional semantics

Replication

- Replicas coordinated with Paxos
- Application specify
  - Datacenters
  - Distance from application (read latency)
  - Distances among replicas (write latency)
  - Number of replicas (durability, availability, read performance)¡

Synchronisation via Time

- Global time hard to achieve!
- Accept bounded uncertainty
  - Establish via GPS and atomic clocks
  - use interval time to define now, before, after
- Use Timestamp-based synchronisation and two-phase commit

Performance

- High commit latency incurred by 2PC over distributed replicas
- Latency hiding via
  - Hierarchical schema: provide partitoning/placement info at schema level (US data at US, European data in the EU)
  - Efficient transfer: protocol buffers
  - Batch reading and writing