CSE 550: Systems for all

Au 2022

Ratul Mahajan

The need for structured data

U*nstructured* data is arbitrary blob of bytes (for the storage system)

- Arbitrary blobs of bytes to be interpreted by programs
- Example systems: Chord, Memcached, GFS

Lots of data is *structured*

- Relationships between data items
- "Web scale" requires the ability to look up items and relationships quickly.
  - Billions of websites, TB of data
- Data should also be searchable, sortable, etc.
- Too hard to build applications without system support for such data/functions

Why not a commercial DBMS?

Existing systems simply could not scale

- Web data is orders of magnitude larger that “bank workloads”
- Some tried: FB used MySQL for years and had a team focused entirely on getting it to scale

Existing systems were hard to tune for modern workloads

- Web relationships are rich; bank transactions are narrow, non-overlapping
- Workloads evolve rapidly – roll out new features/products
- Different goals: Optimize cost -- don’t use high-end machines, tolerate failures
  - 2000: 2500+ for search, 15K+ by 2004, and 250K+ by 2007

Relaxing SQL

Can improve performance by “relaxing” the guarantees of traditional DBMS systems. But relax what?

A - Availability (No - Unavailable means we can’t serve websites)
C - Consistency (Yes - “eventually consistent” is good enough)
I - Isolation (Mostly no - We don’t want malformed data)
D - Durability (Mostly no - Missing data can be a problem)

Relaxing consistency

Idea: Sacrifice strong consistency for improved performance

Particularly appealing to Web use cases -

- Search: can show older (~few hours) web results
- Social: can show out-of order newsfeed elements
- Ads: only need to maintain eventual count

"Eventual" is one form of relaxed consistency

- Merge different results into final result
- Easier to implement (not necessarily easy)
  - Requires merging or versioning

NoSQL Databases

Developed in early Web 2.0 period

Designed to scale laterally
- If you need more storage, provision more machines

Really more about non-relational DBs (noSQL follows)

Tend to not support many traditional DBMS properties
- Specifically, largely lack the ability to do *joins*
  - Joins combine columns from different tables, require that relationships between data elements remain consistent

Key-Value Stores

Essentially a large distributed hash table
Things inside of the table are inherently opaque to the DB
Simplifies much of the DB operation
Expect the application to handle the logic
- DB handles stable storage and consistency

Examples: memcached, Amazon's Dynamo, BerkeleyDB, Redis

“Document-oriented” database

Specialized key-value store

- Specific focus on “semi-structured” data
- “Documents” fit a specific format, e.g., JSON, XML, etc
- Examples: MongoDB, couchDB, Microsoft’s CosmosDB, Amazon’s SimpleDB

Reduce space of queries to improve performance: range, regex, field

Core functions: CRUD

- Creation (or insertion)
- Retrieval (or query, search, read or find)
- Update (or edit)
- Deletion (or removal)

Consider these specialized DBs

- Time-series DBs: Data with timestamps (e.g., IoT sensors)
- Graph DBs: Graph relations (not tables); edge labels are first-class
- Object DBs: Like document DBs but store program-defined objects

Q1: How might you optimize data placement and query speed?

Q2: What guarantees should be given?

Key points:

- Lots of ways to optimize (SQL or not) for specific data types
- Guarantees depend more on applications not data types

Back to Spanner

Nature of data?

Consistency model?

Where is the trade-off?

Over to Tongyan and Tuochao