Navigating the Shifting Database Landscape

Garrett Jacobson
gjacobso
Overview

- Motivation
- Early DB History
  - A Lifetime of Relational Data
- Dawn of Distributed Systems
  - CAP Theorem
  - BigTable and Dynamo
  - NoSQL & Motivations
- Sampling of New DB Systems
  - Redis
  - MongoDB
  - Cassandra
Motivation

- I ❤️ DBs

- understand what all the fuss is about!
Early DBs

"a collection of organized data" -wikipedia

- Created in the 1960's
- IDS at Honeywell
  - network model where data relationship is a graph
- IMS at IBM
  - hierarchical model where data relationship is a tree
- Navigational DBs has many pain points
  - Tied data storage to structure
  - Poor performance in many applications
  - Required experts and big HW
Relational DBs

- Defined in 1970 by Edgar Codd of IBM
- Introduced concept of tables and keys and data normalization
- Led to creation of SQL to allow easier and abstracted data access
- Saw creation of major relational DBs
  - Oracle
  - INGRES
- Oracle actually beat IBM to market
Data Normalization

"process of organizing the fields and tables of a relational database to minimize redundancy and dependency"
- wikipedia

- Results in easier to understand data models
- Prevents modification anomalies
- Allows more general querying patterns
## Data Normalization

<table>
<thead>
<tr>
<th>Employee Name</th>
<th>Course Number</th>
<th>Course Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>Jim Blomo</td>
<td>CS262</td>
<td>Intro to Spam</td>
</tr>
<tr>
<td>Jimmy Retz</td>
<td>CS210</td>
<td>Training Ads</td>
</tr>
<tr>
<td>Jimmy Retz</td>
<td>EE200</td>
<td>Circuits &amp; Kindles</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Employee ID</th>
<th>Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>21</td>
<td>Jim Blomo</td>
</tr>
<tr>
<td>30</td>
<td>Jimmy Retz</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Course Number</th>
<th>Course Name</th>
<th>Teacher ID</th>
</tr>
</thead>
<tbody>
<tr>
<td>CS262</td>
<td>Intro to Spam</td>
<td>21</td>
</tr>
<tr>
<td>CS210</td>
<td>Training Ads</td>
<td>30</td>
</tr>
<tr>
<td>EE200</td>
<td>Circuits &amp; Kindles</td>
<td>30</td>
</tr>
</tbody>
</table>
Data Denormalization

"process of attempting to optimise the read performance of a database by adding redundant data or by grouping data"

- wikipedia

- Does not refer to un-normalized data
- DB assisted denormalization
  - materialized views, etc
- Writes vs Reads
80's and 90's

- Mostly incremental improvements
  - changes in HW requirements
  - performance tuning
- Exploration of other data models
  - most disappeared
  - some became constructs within relational model
  - some remain niche models
- Transactions introduced and refined

However, some very famous properties were defined...
ACID

- DB transactional properties
- Defined by Jim Gray in 1983
  - coined as ACID by others a few years later
- Atomic
  - transactions are all or nothing
- Consistent
  - transactions will always transition between valid states
- Isolated
  - concurrent transactions result in same state as serial
- Durable
  - once transaction is committed, it is not lost*

*lost referred to power loss, crash, error, etc...not to catastrophe or hw failure...
2000's

- CAP Theorem
- BigTable & Dynamo papers
- BASE and eventual consistency
- Post-relational DBs start taking root
  - NoSQL
  - NewSQL
CAP Theorem

- Eric Brewer conjectured in 2000
  - proven in 2002 by others

- Consistency
  - all nodes report the most correct data

- Availability
  - node failures do not prevent continued operation

- Partition-tolerant
  - functions in spite of network disconnect or message loss
BigTable Paper

"sparse, distributed, persistent multi-dimensional sorted map"

- Published by Google in 2006
  - developed internally since 2004
- Targets PB of data on 1000's of machines
- Column family data model
- Focus on compression and storage techniques
- Focus on C and A (trades off P)
Dynamo Paper

"highly available key/value store"

- Published by Amazon in 2007
  - at the core of S3, SimpleDB, and DynamoDB
  - also applied to their core business
- Key / Value data model
- Node symmetry
- Consistent hashing
- Focus on A and P, trade-offs on C
  - conflict resolution strategies
  - NRW tunable consistency
Why Consistent Hashing?

- How do you distribute keys across N nodes?
  - hash(k) mod N

Benefits:
- Simple
- Load spread evenly across N machines

What if we want to add/remove a machine?
Consistent Hashing

- hash keys into some continuous range
- assign nodes to points within this range

Benefits:
- controllable load balancing
- allows incremental scaling
- simpler rebalancing

Virtual Nodes!
NRW (tunable consistency)

- **N**
  - copies of each data item within a cluster

- **R**
  - copies that will be accessed when reading a data item

- **W**
  - copies that must be written before write completes

<table>
<thead>
<tr>
<th>W = N</th>
<th>R = N, W = 1</th>
<th>N &gt; W &gt; 1</th>
</tr>
</thead>
<tbody>
<tr>
<td>like traditional relational DB</td>
<td>fast write performance, slow read performance</td>
<td>R = 1: least consistent, R &gt; 1: better consistency, W+R &gt; N: always latest</td>
</tr>
</tbody>
</table>
BASE

- Basically
- Available
- Soft-state
- Eventual consistency

So now we have a bunch of new systems that start making new trade-offs between CAP. But what have we lost with these new approach?
Relationships

We lost the ability to do JOINs...

It's a big paradigm shift, but that doesn't mean it's better or worse! It's just a different way to model data...
A Whole New World

We didn't lose anything! We just gained new tools...

<table>
<thead>
<tr>
<th>OLD DESIGN PROCESS</th>
<th>NEW DESIGN PROCESS</th>
</tr>
</thead>
<tbody>
<tr>
<td>How do I <strong>model</strong> this data in a relational DB and <strong>what systems do I need to build around it</strong> to make it meet my performance and availability requirements?</td>
<td>What are the <strong>right tool(s)</strong> to store and expose my data to meet my performance and availability requirements?</td>
</tr>
</tbody>
</table>
No Really, Why Now?

Problem:
A small startup needs to store a bunch of user attributed data that will accumulate over time.

Solution:
Model it all properly and throw it into a SQL database.
its getting slow!

add RAM and an SSD

reads are slow!

memcache results

too much read load!

offload with read slaves

write load is too high!

time for vertical sharding...
Complexity

Essential complexity is caused by the characteristics of the problem and cannot be reduced.

Incidental complexity is the difficulties faced due to the chosen software engineering tools.

NoSQL databases step back and solve different data storage and retrieval problems by designing new solutions with different trade-offs.
NoSQL Varietals

- Varying data models
  - column family
    - Cassandra, HBase, Hypertable, Megastore, ...
  - key/value
    - SimpleDB, DynamoDB, Riak, Voldemort, Redis (?), Couchbase, ...
  - document
    - MongoDB, CouchDB, ElasticSearch, ...
  - graph
    - Neo4J, GraphDB, ...

- Varying attributes
  - CAP trade-offs, scaling strategies, performance characteristics, replication, etc
- In-memory key/value store (with typed values)
  - some call it "data structure server"
- Notable Features
  - blazingly fast?
  - value types: strings, hashes, lists, sets, sorted sets
  - data structure specific (atomic) operations
● Supports transactions

● Entries can have expiry times

● Supports statement logging and slave replication
  ○ data is backed up to disk periodically

● Server side lua scripting

● Publish/subscribe interface
is web scale!
• Document store

• Focuses on CP and makes trade-offs in A

• Notable Features:
  ○ schema-less data model
  ○ run arbitrary javascript on server
  ○ strong replication features + datacenter awareness
  ○ geospatial indexing
    ■ $near and $within
  ○ per-write durability settings
    ■ fnf, safe
- **Data Model**
  - mongo instance runs a database
  - database has 0 or more collections
  - collection has 0 or more documents
  - document has 0 or more fields

- **Data is JSON-style**
  ```javascript
  db.users.save({name: "darwin", email: "dstop@yelp.com"})
  db.employees.find({name: "darwin"})
  [ {"name": "darwin", "_id": { "$oid" : "..."}, "email": "dstop@yelp.com"} ]
  ```

- **Collections can be indexed**
  - improves lookup and sort
Had lots of durability issues early-on
  ○ Solved many with journaling and sharding features

Supports replication
  ○ replica set

No transactions
  ○ supports many atomic ops, or use 2PC

Built-in sharding functionality
  ○ spread data out across replica sets
Based on BigTable and Dynamo principles

Column family data model
  ○ keyspace has column families
  ○ column family has keyed rows
  ○ row has columns

Focuses on AP, trade-offs on C
[default@MyKeyspace] set User['dstop']['name']='Darwin'

[default@MyKeyspace] set User['dstop']['email']='darwin@aerospace.org'

[default@MyKeyspace] get User['dstop']
=> (column=666e616d65, value=Darwin, timestamp=1282510290343000)
=> (column=656d61696c, value=darwin@aerospace.org, timestamp=1282510313429000)
Returned 2 results.
- fully distributed cluster architecture
  - all nodes are equal
  - linear scaling effects
  - minimizes bottlenecks
  - datacenter aware (programmable strategies)
- request tunable consistency
- tunable replication
- Log structured storage engine
  - SSTables and append-only logs

- Writes have no seeks and only perform sequential i/o!
  - Designed for HDDs but actually helps with SSDs too

writes can be really fast
- **Bloom Filters**
  - helps speed up lookups and searches

- **Tombstones**
  - hope you didn't really need that data gone!
My Takeaways

● We're going through a period of database innovation and some really cool technologies
  ○ Relational DB's are here to stay

● We've got a lot more tools at our disposal to store data in distributed systems than ever before!
  ○ Even if many tools are still experiencing growing pains

Increased importance of system architecture!
Don't Forget To Be Pragmatic

"Tool vendors tout the miracles of their products platform. Everyone claims their programming language is the best, and every OS is the answer to all conceivable ills. Of course, none of this is true. There are no easy answers. There is no such thing as the best solution, be it a tool, a language, or an OS. There can only be systems that are more appropriate in a particular set of circumstances."

-The Pragmatic Programmer
Andrew Hunt & David Thomas
Questions?