DESIGNING FOR CONSUMPTION

Seth Dobbs
VP of Engineering – HS2 Solutions
@sethdtech
https://www.linkedin.com/in/seth-dobbs-hs2/

O’REILLY SOFTWARE ARCHITECTURE 4/4/2017
RELATIONAL WORLD
RELATIONAL DATA

- RDBMS has longevity
- ACID (Atomicity, Consistency, Isolation, Durability)
- Normalization (3NF) is good
- Designs often assume data exists in it’s own right
- Standard data design patterns
EARLY ARCHITECTURE
RDBMS TRADEOFFS

- Schema rigidity makes it difficult/costly to evolve over time
- Impedance mismatch: difference between relational models and in-memory models
- RDBMS scaling model is to scale up (buying bigger servers) not out (buying more servers)
- Geographic distribution – backup DB servers require “nearness” to minimize latency
- Database as integration point creates complexity – particularly as different teams need to pursue different destinies
Increased demands from multiple applications are leading to a major single point of failure.

Clustering provides some help, but clusters are more about failover than capacity.
MODERN CHALLENGES

Global distribution and latency

Rapid product iteration
READ/WRITE RATIOS

New world – often 100:1 read:write. Availability and Partition tolerance become key.

RDBMS origins – within corporate firewall, near read/write parity.
DATA: RELATIONAL VS. NON-RELATIONAL (NOSQL)
SCHEMALESS

• Schemaless → no rigid structure
• Don’t have to know exactly what you need in advance
• Can easily add fields or stop storing something you don’t need
• Helpful for non-uniform data
• Implicit schema still exists (need to know if field is qty or quantity)
• Fields still have meaning if you’re doing more than simply displaying / saving
• Rigid schema can have value which is why RDBMS, XML, etc. have them
• Schemaless is a shifting of schema knowledge to application code
AGGREGATE ORIENTATION

• Recognition that we often operate on data in units that have a more complex structure
• Denormalize to minimize the number of aggregates we access during a query
• Design requires thinking about aggregate boundaries based on usage
• RDBMS are aggregate-ignorant; not necessarily bad; boundaries hard to define
• Aggregation helps with running on a cluster – gives clear direction as to what data should live on what node
• Aggregation can be a barrier to some kinds of transactions
RELATIONAL VS. AGGREGATION
NORMALIZED WORKLOAD
DE-NORMALIZED WORKLOAD

Browser

Browser

Browser

API

Aggregation
“Can only optimize at most two of the three”
Single RDBMS server: C
Most NoSQL focus on AP
Availability essentially the limit of latency we can accept on C
EVENTUAL CONSISTENCY

- Nodes communicate their writes to each other
- No single point of write bottleneck or failure
SERVICE DEFINITION
RISK OF LOW GRANULARITY

- Services equating to relational table level CRUD – too granular
- Brittle: no encapsulation of implementation
- Chatty: typically requires multiple service calls to accomplish business purpose
- Expensive: increases latency due both to number of calls and hits against database
AGGREGATE ORIENTATION (SERVICES)

- Similar to NoSQL, consumption-focused services should be aggregate-oriented.
- Applications typically operate on data in units that have a more complex structure than a single relational table.
- Design should contemplate aggregate boundaries based on usage.
EXAMPLE: PRODUCTS

Approach 1

Browser

getPart(s) -> Select part(s)

part results -> results

getXxxAttrs -> Select Xxx Attrs

attr results -> results

update screen

Approach 2

Browser

getPart(s) -> Select part(s) and attr

part results -> results

update screen
DEEPER DIVE SCENARIO

• Website provides access to physical auctions around the world
• Users need to register with auction houses in order to bid on their auctions
• Users want to:
  • View list of auctions starting soon
  • Know at which of those auction house they’ve registered
  • Know in which of the auctions they have bid
APPROACH 1: LET THE USER SORT IT OUT
APPROACH 2: FRONT END CODE SORTS IT OUT
APPROACH 3: SERVICE COMPOSITION
APPROACH 4: COMPLEX SERVICE
APPROACH 5: SIMPLIFIED SERVICE
ARCHITECTURAL APPROACHES
DATA CONSIDERATIONS

Acceptable Latency = $1 / (\text{Criticality} \times \text{Volatility})$

Strength of write transaction directly related to Risk of Conflict
SERVICE CONSIDERATIONS

Intent of consumption

Viewing vs. adding vs. manipulating

Aggregate boundaries
SCENARIO: USER PROFILE

Data Considerations
• Low criticality
• Low volatility
• Minimum risk of conflict

Service Considerations
• Need CRUD
• Can work with aggregate profile
• Exception: CC data, password, etc.

Architecture Choices
• Document-based DB
• Service aligned with document
**SCENARIO: HISTORY VS. ACTIVE**

**Data Considerations**
- Ex. Orders or Events
- Active entities are volatile, higher demand
- Historical entities are static, low demand, ever growing

**Service Considerations**
- Active entities need CRUD
- Historical entities are R

**Architecture Choices**
- Separate Active from Historical
- Every day active entities become a smaller and smaller percentage of the overall pool
- Most users concerned with Active
SCENARIO: SOCIAL MEDIA POSTS

Data
• R/W ratio is HIGH
• Data volatile but not critical
• Minimal conflict
• Tolerant update latency

Service
• Personal stream CRD
• Majority of services are R
SCENARIO: HOTEL RESERVATIONS

Attributes
- International users
- R:W ratio is high
- Low-mid volatility
- Low-mid conflict risk
- Maintain high browse/search performance for int’l

Analysis
- High tolerance for latency at read time
- Zero tolerance for transaction conflict at write
- Cannot partition data by locale
- Partitioned architecture

DATA CENTER 1
DATA CENTER N
## SCENARIO: MARKET MAKER

<table>
<thead>
<tr>
<th>Attributes</th>
<th>Analysis</th>
<th>Thoughts</th>
</tr>
</thead>
<tbody>
<tr>
<td>• Producers/sellers need to manipulate store info, product info, etc. (low R:W)</td>
<td>• Potentially partitioned architecture</td>
<td>• Universal service definition across different forms of use is problematic</td>
</tr>
<tr>
<td>• P/S need to run reports / analysis</td>
<td>• Potentially different service definitions; Sellers need lower level CRUD</td>
<td>• CQRS Pattern</td>
</tr>
<tr>
<td>• Consumers want to browse and buy (high R:W)</td>
<td>• Does a consumer app need to know who signed on a seller?</td>
<td>• Single source of truth != single location of data</td>
</tr>
</tbody>
</table>
MULTI-PATTERN ARCHITECTURE

- **User Interfaces**
- **Geographic Distribution**
- **Horizontal Scaling**

- **DATA CENTER 1**
  - WEB
  - DATA
  - PUBLISH

- **DATA CENTER X**
  - WEB
  - DATA

- **High Concurrency Data**

- **B2B Clients**
- **Analytics**
- **Alerting**
- **Back Office**
THANK YOU

Seth Dobbs
VP of Engineering, HS2 Solutions
@sethdtech
https://www.linkedin.com/in/seth-dobbs-hs2/