The Case for Change Notifications in Pull-Based Databases

Wolfram Wingerath, Felix Gessert, Steffen Friedrich, Erik Witt and Norbert Ritter

Wolfram Wingerath
wingerath@informatik.uni-hamburg.de
March 6th, 2017, Stuttgart
Traditional Databases

No Request? No Data!

What’s the current state?

circular shapes

Query maintenance: periodic polling

→ Inefficient

→ Slow
Find people in Room B:

```javascript
db.User.find()
  .equal('room', 'B')
  .ascending('name')
  .limit(3)
  .streamResult()
```
Real-Time Databases
Overview:

- Real-time state synchronization across devices
- Simplistic data model: nested hierarchy of lists and objects
- Simplistic queries: mostly navigation/filtering
- Fully managed, proprietary
- App SDK for App development, mobile-first
- Google services integration: analytics, hosting, authorization, ...

History:

- 2011: chat service startup Envolve is founded
  → was often used for cross-device state synchronization
  → state synchronization is separated (Firebase)
- 2012: Firebase is founded
- 2013: Firebase is acquired by Google
Firebase
Real-Time State Synchronization

• **Tree data model**: application state ~JSON object
• **Subtree synching**: push notifications for specific keys only
  → Flat structure for fine granularity

→ *Limited expressiveness!*
Firebase
Query Processing in the Client

• Push notifications for **specific keys** only  
  • Order by a **single attribute**  
  • Apply a **single filter** on that attribute

• Non-trivial query processing in client
  → **does not scale!**

Illustration taken from: Frank van Puffelen, *Have you met the Realtime Database?* (2016)

Jacob Wenger, on the Firebase Google Group (2015)
https://groups.google.com/forum/#!topic/firebase-talk/d-XjaBVL2Ko (2017-02-27)
Overview:

- **JavaScript Framework** for interactive apps and websites
  - **MongoDB** under the hood
  - **Real-time** result updates, full MongoDB expressiveness
- **Open-source**: MIT license
- **Managed service**: Galaxy (Platform-as-a-Service)

History:

- 2011: *Skybreak* is announced
- 2012: Skybreak is renamed to Meteor
- 2015: Managed hosting service Galaxy is announced
Live Queries
Poll-and-Diff

- **Change monitoring**: app servers detect relevant changes → *incomplete* in multi-server deployment
- **Poll-and-diff**: queries are re-executed periodically → *staleness window* → does not scale with queries
Oplog Tailing
Basics: MongoDB Replication

- **Oplog**: rolling record of data modifications
- **Master-slave replication**: Secondaries subscribe to oplog

![Diagram of MongoDB cluster](image)

- **mongoDB cluster (3 shards)**
- **write operation**
- **apply**
- **propagate change**

Secondary C1  Secondary C2  Secondary C3
Oplog Tailing
Tapping into the Oplog

- Every Meteor server receives all DB writes through oplogs → does not scale
Oplog Tailing
Oplog Info is Incomplete

What game does Bobby play?
→ if baccarat, he takes first place!
→ if something else, nothing changes!

Partial update from oplog:
{ name: "Bobby", score: 500 } // game: ???

Baccarat players sorted by high-score
1. { name: "Joy", game: "baccarat", score: 100 }
2. { name: "Tim", game: "baccarat", score: 90 }
3. { name: "Lee", game: "baccarat", score: 80 }
**Overview:**
- „MongoDB done right“: comparable queries and data model, but also:
  - **Push-based queries** (filters only)
  - **Joins** (non-streaming)
  - **Strong consistency**: linearizability
- **JavaScript SDK** (*Horizon*): open-source, as managed service
- **Open-source**: Apache 2.0 license

**History:**
- 2009: RethinkDB is founded
- 2012: RethinkDB is open-sourced under AGPL
- 2016, May: first official release of Horizon (JavaScript SDK)
- 2016, October: RethinkDB announces shutdown
- 2017: RethinkDB is relicensed under Apache 2.0
RethinkDB
Changefeed Architecture

- Range-sharded data
- **RethinkDB proxy**: support node without data
  - Client communication
  - Request routing
  - Real-time query matching

- *Every* proxy receives all database writes
  → **does not scale**


Daniel Mewes, *Comment on GitHub issue #962: Consider adding more docs on RethinkDB Proxy* (2016)
https://github.com/rethinkdb/docs/issues/962 (2017-02-27)
Overview:

- **Backend-as-a-Service** for mobile apps
  - **MongoDB**: largest deployment world-wide
  - **Easy development**: great docs, push notifications, authentication, ...
  - **Real-time** updates for most MongoDB queries
- **Open-source**: BSD license
- **Managed service**: discontinued

History:

- 2011: Parse is founded
- 2013: Parse is acquired by Facebook
- 2015: more than 500,000 mobile apps reported on Parse
- 2016, January: Parse shutdown is announced
- 2016, March: **Live Queries** are announced
- 2017: Parse shutdown is finalized
Parse
LiveQuery Architecture

- **LiveQuery Server**: no data, real-time query matching
- *Every* LiveQuery Server receives *all* database writes
  - → **does not scale**

---

Illustration taken from:
## Comparison by Real-Time Query

### Why Complexity Matters

<table>
<thead>
<tr>
<th>matching conditions</th>
<th>ordering</th>
<th>Firebase</th>
<th>Meteor</th>
<th>RethinkDB</th>
<th>Parse</th>
</tr>
</thead>
<tbody>
<tr>
<td>Todos</td>
<td>created by „Bob“</td>
<td>ordered by deadline</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>Todos</td>
<td>created by „Bob“ AND with status equal to „active“</td>
<td></td>
<td>❌</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>Todos</td>
<td>with „work“ in the name</td>
<td></td>
<td>❌</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>Todos</td>
<td>with „work“ in the name AND status of „active“</td>
<td>ordered by deadline</td>
<td>❌</td>
<td>✓</td>
<td>✓</td>
</tr>
</tbody>
</table>

60
# Quick Comparison

## DBMS vs. RT DB vs. DSMS vs. Stream Processing

<table>
<thead>
<tr>
<th></th>
<th>Database Management</th>
<th>Real-Time Databases</th>
<th>Data Stream Management</th>
<th>Stream Processing</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Data</strong></td>
<td>persistent collections</td>
<td>persistent/ephemeral streams</td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Processing</strong></td>
<td>one-time</td>
<td>one-time + continuous</td>
<td>continuous</td>
<td></td>
</tr>
<tr>
<td><strong>Access</strong></td>
<td>random</td>
<td>random + sequential</td>
<td>sequential</td>
<td></td>
</tr>
<tr>
<td><strong>Streams</strong></td>
<td>structured</td>
<td></td>
<td>structured, unstructured</td>
<td></td>
</tr>
</tbody>
</table>

**Database Management Examples:**
- ORACLE
- PostgreSQL
- MySQL
- IBM DB2

**Real-Time Databases Examples:**
- Firebase
- METEOR
- RethinkDB
- Parse

**Data Stream Management Examples:**
- PIPELINE DB
- EsperTech
- sqlstream
- influx data

**Stream Processing Examples:**
- STORM
- samza
- Flink
- Spark Streaming
Discussion

Common Issues

Every database with real-time features suffers from several of these problems:

• **Expressiveness**:
  • Queries
  • Data model
  • Legacy support

• **Performance**:
  • Latency & throughput
  • **Scalability**

• **Robustness**:
  • Fault-tolerance, handling malicious behavior etc.
  • Separation of concerns:
    → **Availability**:
      will a crashing real-time subsystem take down primary data storage?
    → **Consistency**:
      can real-time be scaled out independently from primary storage?
Engineering Efforts:
Add-On Real-Time Queries
InvaliDB
External Query Maintenance
InvaliDB
Change Notifications

SELECT *
FROM posts
WHERE title LIKE "%NoSQL%"
ORDER BY year DESC

{ title: "SQL",
  year: 2016 }

add    changelIndex    change    remove
InvaliDB
Filter Queries: Distributed Query Matching

Two-dimensional partitioning:
• by Query
• by Object
→ scales with queries and writes

Implementation:
• Apache Storm
• Topology in Java
• MongoDB query language
• Pluggable query engine

SELECT * FROM posts WHERE tags CONTAINS 'NoSQL'

For Each Query:
- Is Match?
  - Yes
  - No
- Was Match?
  - Yes
  - No
- change
- add
- remove
- %

Write op!

Match!
InvaliDB
Staged Real-Time Query Processing

Change notifications go through up to 4 query processing stages:
1. **Filter queries**: track matching status → before- and after-images
2. **Sorted queries**: maintain result order
3. **Joins**: combine maintained results
4. **Aggregations**: maintain aggregations
InvaliDB
Low Latency + Linear Scalability

![Graph showing throughput vs matching nodes with latency markers.]
Research in Hamburg
Delivering Dynamic Content

Two Bottlenecks: Latency und Processing
Solution: Global Caching
Fresh Data from Ubiquitous Web Caches

Low Latency

Less Processing
Caching Dynamic Content
Now Feasible: Invalidating Updated Queries
Wrap-up

- **Push-based data access**
  - Natural for many applications
  - Hard to implement on top of traditional (pull-based) databases

- **Real-time databases**
  - Natively push-based
  - Not legacy-compatible
  - Barely scalable

- **InvaliDB**
  - Add-On push-based queries
  - Database-independent
  - Linear scalability
  - Filter, sorting, joins, aggregations
Questions?