Scalable Push-Based Real-Time Queries on Top of Pull-Based Databases

Wolfram Wingerath

May 8, 2019, Disputation, Hamburg
Outline

- Problem Statement
  - Intro & Research Question

- Related Work
  - State of the Art & Open Issues

- A Scalable RTDB Design
  - InvaliDB: Concept & Prototype

- Discussion
  - Applications & Outlook

- Pull vs. Push
  - Traditional DB Queries
  - Why Real-Time Queries?
  - How to Provide Them?

- The Primary Challenges
  - C₁ Scalability
  - C₂ Expressiveness
  - C₃ Legacy Support
  - C₄ Abstract API

- Research Question
Traditional Databases
The Problem: No Request – No Data!

Periodic Polling for query result maintenance:
→ inefficient
→ slow
Real-time Databases
Always Up-to-Date With Database State

Real-Time Queries for query result maintenance:
→ efficient
→ fast
Real-Time Query Maintenance
Matching Every Query Against Every Update

→ Potential *bottlenecks*:
  - *Number of queries*
  - *Write throughput*
  - *Query complexity*

Similar processing for:
  - Triggers
  - ECA rules
  - Materialized views
Challenges
Real-Time Databases: Major challenges

C<sub>1</sub>: Scalability:
- Handle additional queries
- Handle increasing throughput

C<sub>2</sub>: Expressiveness:
- Content search? Composite filters?
- Ordering? Limit? Offset?

Research Question: „How can expressive push-based real-time queries be implemented on top of an existing pull-based database in a scalable and generic manner?“

C<sub>3</sub>: Legacy Support:
- Real-time queries for existing databases
- Decouple OLTP from real-time workloads

C<sub>4</sub>: Abstract API
- Data independence
- Self-maintaining queries
# Outline

<table>
<thead>
<tr>
<th>Problem Statement</th>
<th>Data Management Classes</th>
</tr>
</thead>
<tbody>
<tr>
<td>Intro &amp; Research Question</td>
<td>Historical Overview</td>
</tr>
<tr>
<td>Related Work</td>
<td>4-Part Categorization</td>
</tr>
<tr>
<td>State of the Art &amp; Open Issues</td>
<td>Real-Time Databases</td>
</tr>
<tr>
<td>A Scalable RTDB Design</td>
<td>Poll-and-Diff</td>
</tr>
<tr>
<td>InvaliDB: Concept &amp; Prototype</td>
<td>Oplog Tailing</td>
</tr>
<tr>
<td>Discussion</td>
<td>System Comparison</td>
</tr>
<tr>
<td>Applications &amp; Outlook</td>
<td>Meteor</td>
</tr>
<tr>
<td></td>
<td>RethinkDB</td>
</tr>
<tr>
<td></td>
<td>Parse</td>
</tr>
<tr>
<td></td>
<td>Firebase</td>
</tr>
<tr>
<td></td>
<td>InvaliDB</td>
</tr>
</tbody>
</table>
A Short History of Data Management
Hot Topics Through The Ages

Relational Databases
- Entity-Relationship Model
- Triggers
- Ingres
- System R
- SQL Standard
- Starburst
- HiPAC
- PostgreSQL

CEP & Streams
- MapReduce
- STREAM
- Rapide
- Telegraph
- Bigtable
- Spark
- Samza
- Meteor

Stream Processing
- Flink
- Dynamo
- Storm
- RethinkDB
- Firebase
- InvaliDB

Big Data & NoSQL
- Aurora & Borealis
- Bigtable
- GFS

Real-Time Databases

Active Databases
- Relational Model

[WRG19, WGW+18]
Data Management Systems
A High-Level Categorization

Database Management
- static collections
- pull-based

Real-Time Databases
- evolving collections

Data Stream Management
- structured streams

Stream Processing
- unstructured streams
- push-based

[WRG19, WGW+18]
Typical Maintenance Mechanisms (1/2)

Poll-and-Diff

- **Local change monitoring**: app servers detect local changes → incomplete in multi-server deployment
- **Poll-and-diff**: global changes are discovered through polling → staleness window → read scalability?

[GW17, Win17]
Typical Maintenance Mechanisms (2/2)

Change Log Tailing

- Every application server receives all DB writes through oplog → write scalability?
# Real-Time Database Comparison

<table>
<thead>
<tr>
<th></th>
<th>METEOR</th>
<th>RethinkDB</th>
<th>Parse</th>
<th>Firebase</th>
<th>Invati DB</th>
</tr>
</thead>
<tbody>
<tr>
<td>Poll-and-Diff</td>
<td>✓</td>
<td>✗</td>
<td>✗</td>
<td>✗</td>
<td>✗</td>
</tr>
<tr>
<td>Change Log Tailing</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>?</td>
</tr>
<tr>
<td>Unknown</td>
<td>✗</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>2-D Partitioning</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
</tbody>
</table>

| **Write Scalability**    | ✓      | ✗         | ✗     | ✗        | ✗         |
| **Read Scalability**     | ✗      | ✓         | ✓     | ✓        | ✓         |
| (100k connections)       | (AND In Firestore) | | | | |

| Composite Filters (AND/OR) | ✓      | ✓         | ✓     | ✓        | ✓         |
| Sorted Queries             | ✓      | ✓         | ✓     | ✗        | ✓         |
| (single attribute)         | (AND In Firestore) | | | | |
| Limit                      | ✓      | ✓         | ✓     | ✗        | ✓         |
| Offset                     | ✓      | ✓         | ✗     | ✗        | ✓         |
| (value-based)              | | | | | |
| Self-Maintaining Queries   | ✓      | ✓         | ✗     | ✗        | ✓         |
| Event Stream Queries       | ✓      | ✓         | ✓     | ✓        | ✓         |

[GWR17, Win17]
Outline

- System Model & Overview
  - Query Subscription
  - Write Ingestion
  - Change Propagation
- Real-Time Query Processing
  - Two-Dimensional Workload Partitioning
  - Processing Stages
- Performance Evaluation
  - Read Scalability
  - Write Scalability
  - Multi-Tenancy

- Problem Statement
  Intro & Research Question

- Related Work
  State of the Art & Open Issues

- A Scalable RTDB Design
  InvaliDB: Concept & Prototype

- Discussion
  Applications & Outlook

- Applications & Outlook
InvaliDB: A Scalable Real-Time Database Design
System Model & Overview

Realtime-as-a-Service For Heterogeneous Tenants:
- resource pooling: high matching performance & overall efficiency
- multi-tenancy: low provisioning overhead per application server

Real-Time & OLTP Workloads Decoupled:
- isolated failure domains
- separated resource requirements & independent scaling

InvaliDB Cluster
Event Layer
Application Server
Pull-Based Database

[WGF+17, GSW+17]
InvaliDB: A Scalable Real-Time Database Design

Two-Dimensional Workload Partitioning

Read & Write Scalability:
- many concurrent users
- high write throughput
- no single-server bottleneck

Pluggable Query Engine:
- legacy-compatibility
- multi-tenancy across databases

[WGF+17, GSW+17]
InvaliDB: A Scalable Real-Time Database Design
Staged Real-Time Query Processing

Change notifications go through different 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

[WGF+17, GSW+17]
Evaluation: Performance & Scalability
Prototype Implementation

Query Processing
- low latency
- customizability
- tried & tested

Event Layer
- low latency
- high per-node throughput
- ease of deployment

Database
- typical RTDB expressiveness
- typical NoSQL datastore
- wildly popular

[WGF+17, GSW+17, Win16, WGFR16, GWFR16]
Linear Read Scalability
Sustainable Queries at 1k Writes per Second

1.5 mio. matching ops/s per node
Linear Write Scalability
Sustainable Throughput With 1k Active Queries

1 mio. matching ops/s per node
Outline

Problem Statement
Intro & Research Question

Related Work
State of the Art & Open Issues

A Scalable RTDB Design
InvaliDB: Concept & Prototype

Discussion
Applications & Outlook

• Application Scenarios
  • Real-Time Queries
  • Query Caching

• Future Work

• Publications
  • Articles & Papers
  • Tutorials
  • Book

• Contributions
  • Data Management Categorization
  • InvaliDB: Design & Impl.
  • Proof of Practicality
Use Case 1: Real-Time Queries
An Easy-to-Use JavaScript API

```javascript
var query = DB.Tweet.find()
  .matches('text', /my filter/)
  .descending('createdAt')
  .limit(10)
  .offset(20);

query.resultList(result => ...);

query.resultStream(result => ...);
```

[Use Case 1: Real-Time Queries](WGR19a)
Baqend Real-Time Query Performance
Low Overhead, High Efficiency

Read-Heavy Workload

Write-Heavy Workload
Use Case 2: Consistent Query Caching
InvaliDB For Invalidating DB Queries

How to detect changes to query results:
„Give me the most popular products that are in stock.“

[WGFF17, GSW17]
Query Caching
Improving Pull-Based Query Performance

Latency

Throughput

[GSW+17]
Future Research
Open Challenges & Follow-Up Work

Extending Semantics
- Additional Languages, Joins & Aggregations
- Transactions
- Stream-Based Queries & CEP

Trade-Offs & Optimizations
- Failure Transparency
- Deployment & Adaptive Scaling
- Client Performance

Exploring New Use Cases
- Reactive & Collaborative (Mobile) Apps
- Enhancing UI in Existing Applications
- Augmenting Cache Coherence Schemes
Publications
DMC 2014, Datenbank-Spektrum, BTW 2015

[GFW+ 14] Gessert, Felix; Friedrich, Steffen; Wingerath, Wolfram; Schaarschmidt, Michael; Ritter, Norbert: Towards a Scalable and Unified REST API for Cloud Data Stores, Informatik 2014 (DMC 2014)


Publications
..., highscalability.com, it – Information Technology


Publications
..., SummerSOC 2016, SCDM 2017, BTW 2017


[WGF+17] Wingerath, Wolfram; Gessert, Felix; Friedrich, Steffen; Witt, Erik; Ritter, Norbert: *The Case For Change Notifications in Pull-Based Databases*, SCDM 2017


[GWR17] Gessert, Felix; Wingerath, Wolfram; Ritter, Norbert: *Scalable Data Management: An In-Depth Tutorial on NoSQL Data Stores*, BTW 2017
Publications

..., VLDB 2017, EDBT 2018, Springer Book, BTW 2019

[GSW+17] Gessert, Felix; Schaarschmidt, Michael; Wingerath, Wolfram; Witt, Erik; Yoneki, Eiko; Ritter, Norbert: Quaestor: Query Web Caching for Database-as-a-Service Providers, **VLDB 2017**

[WGW+18] Wingerath, Wolfram; Gessert, Felix; Witt, Erik; Friedrich, Steffen; Ritter, Norbert: *Real-Time Data Management for Big Data*, **EDBT 2018**


[WGR19b] Wingerath, Wolfram; Gessert, Felix; Ritter, Norbert: *NoSQL & Real-Time Data Management in Research & Practice*, **BTW 2019**
Summary & Contributions

1.) System Categorization

Traditional Databases:
pull-based queries
  - inefficient
  - slow

2.) RTDB System Design for Opt-in Real-Time Queries

With InvaliDB:
push-based queries
  - scalable & fast
  - expressive
  - legacy-compatible

3.) A MongoDB-Based Implementation

4.) Proof of Practicality Through Integration With Orestes