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

Dr. Wolfram Wingerath
December 3, 2019, DBDC, Munich
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

Big Data Analytics
   What I Actually Do in My Job
Traditional Databases
The Problem: No Request – No Data!

What’s the current state?

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

C₁: Scalability:
- Handle additional queries
- Handle increasing throughput

C₂: 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?“

C₃: Legacy Support:
- Real-time queries for existing databases
- Decouple OLTP from real-time workloads

C₄: Abstract API
- Data independence
- Self-maintaining queries
Outline

- Problem Statement
  - Intro & Research Question

- Related Work
  - State of the Art & Open Issues

- A Scalable RTDB Design
  - InvaliDB: Concept & Prototype

- Discussion
  - Applications & Outlook

- Big Data Analytics
  - What I Actually Do in My Job

- Data Management Classes
  - Historical Overview
  - 4-Part Categorization

- Real-Time Databases
  - Poll-and-Diff
  - Oplog Tailing

- System Comparison
  - Meteor
  - RethinkDB
  - Parse
  - Firebase
  - InvaliDB

Discussion

Applications & Outlook

Big Data Analytics

What I Actually Do in My Job
A Short History of Data Management

Hot Topics Through The Ages

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

CEP & Streams
- MapReduce
- STREAM
- Bigtable
- Spark
- Aurora & Borealis
- GFS
- Dynamo
- Flink
- Storm
- Samza
- Meteor

Active Databases
- RethinkDB
- Big Data & NoSQL
- HiPAC
- PostgreSQL
- System R
- Rapide

Real-Time Databases
- InvaliDB
- [WRG19, WGW+18]
Data Management Systems
A High-Level Categorization

pull-based
Database Management
  static collections

Real-Time Databases
  evolving collections

Data Stream Management
  structured streams

push-based
Stream Processing
  unstructured streams

[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?

![Diagram showing database cluster with shards and application servers connected through change log broadcast]

[GWR17, Win17]
## Real-Time Database Comparison

<table>
<thead>
<tr>
<th></th>
<th>METEOR</th>
<th>RethinkDB</th>
<th>Parse</th>
<th>Firebase</th>
<th>InvatiDB</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Poll-and-Diff</strong></td>
<td>✔</td>
<td>✗</td>
<td>✗</td>
<td>✗</td>
<td>✗</td>
</tr>
<tr>
<td><strong>Change Log Tailing</strong></td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
<td>☐</td>
</tr>
<tr>
<td><strong>Unknown</strong></td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
<td>?</td>
</tr>
<tr>
<td><strong>2-D Partitioning</strong></td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
</tr>
</tbody>
</table>

### Write Scalability

- **Composite Filters (AND/OR)**: ✔ ✔ ✔ ✔ ☐
- **Sorted Queries**: ✔ ✔ ✔ ✗ ☐
- **Limit**: ✔ ✔ ✔ ✗ ✔
- **Offset**: ✔ ✔ ✗ ✗ ✔
- **Self-Maintaining Queries**: ✔ ✔ ✗ ✗ ✔
- **Event Stream Queries**: ✔ ✔ ✔ ✔ ✔

---

[GW17, Win17]
Outline

- Problem Statement
  - Intro & Research Question
- Related Work
  - State of the Art & Open Issues
- A Scalable RTDB Design
  - InvaliDB: Concept & Prototype
- Discussion
  - Applications & Outlook
- Big Data Analytics
  - What I Actually Do in My Job

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

1. Query Subscription
2. Write Ingestion
3. Change Propagation

[WGR20, WGF^17, GSW^17]
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

[WGR20, WGF+17, GSW+17]
InvaliDB: A Scalable Real-Time Database Design
Two-Dimensional Workload Partitioning

[WGR20, 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

[WGR20, 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

[WGR20, 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

[STORM, InvaliDB Cluster, Event Layer, Application Server, mongoDB, Pull-Based Database]

[WGR20, 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
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);
```

**Pull-Based Query**
```
query.resultList(result => ...);
```

**Real-Time Query**
```
query.resultStream(result => ...);
```
Bagend Real-Time Query Performance
Low Overhead, High Efficiency

Read-Heavy Workload

Write-Heavy Workload
How to detect changes to query results:
„Give me the most popular products that are in stock.“
Query Caching
Improving Pull-Based Query Performance

[Graph showing latency and throughput with different caching strategies.]

[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


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

[FWR17] Friedrich, Steffen; Wingerath, Wolfram; Ritter, Norbert: Coordinated Omission in NoSQL Database Benchmarking, SCDM 2017


[GWR17] Gessert, Felix; Wingerath, Wolfram; Ritter, Norbert: Scalable Data Management: An In-Depth Tutorial on NoSQL Data Stores, BTW 2017
Gessert, Felix; Schaarschmidt, Michael; Wingerath, Wolfram; Witt, Erik; Yoneki, Eiko; Ritter, Norbert: Quaestor: Query Web Caching for Database-as-a-Service Providers, *VLDB 2017*

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


Wingerath, Wolfram; Gessert, Felix; Ritter, Norbert: *Twoogle: Searching Twitter With MongoDB Queries*, *BTW 2019*

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

Wingerath, Wolfram; Gessert, Felix; Ritter, Norbert: *InvaliDB: Scalable Push-Based Real-Time Queries on Top of Pull-Based Databases*, *ICDE 2020*
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
Big Data Analytics With AWS Athena

Wolle’s Day-to-Day Business
3 Things Make Your Website Slow

1. Backend Processing
2. Network Delays
3. Client
Accelerating Personalized HTML

Browser

Merge

Fast & Anonymous

Speed Kit Cloud

Origin Server

Slow & Personalized

Number 3
Before Speed Kit

After Speed Kit

2.5x faster
My Domain: Performance & Business Insights

- Time-to-First-Byte
- First (Contentful) Paint
- DOM Timer
- First Input Delay

- Session Length
- Time on Site
- First User Interaction
- Bounce Rate

- Cart Size
- Transactions
- Conversion Rate
- Revenue

- Page Views & Sessions
- Browser Distribution
- JavaScript Errors
- Caching Insights

Performance
User Engagement
Business KPIs
QA Metadata

Browser
Cloud Backend

Timing API
Service Worker
Unhandled Errors
Split Testing for Web Performance

- Speed Kit Users
  - Speed Kit enabled

- vs.

- Normal Users
  - Speed Kit disabled (no acceleration)

- **Measurable uplift:**
  - Performance
  - User engagement
  - Business success
Join Our E-Commerce Performance Study

“Mobile Site Speed and the Impact on E-Commerce”

- **Topic:** largest systematic study on e-commerce performance
- **When:** start in **summer 2019** by Google, Baqend, and the University of Hamburg
- **Participants:** leading e-commerce players in Europe
- **Method:** A/B test in production with the support of Baqend and Google
Thanks! Any Questions?

Join the study!
Details & newsletter on
speedstudy.info

Wolfram Wingerath, Data Engineering ww@baqend.com