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

Wolfram Wingerath, Felix Gessert, Norbert Ritter
ICDE 2020, Dallas/USA
InvaliDB: Scalable Push-Based Real-Time Queries on Top of Pull-Based Databases

Wolfram Wingerath, Felix Gessert, Norbert Ritter

ICDE 2020, Dallas/USA
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?
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
Outline

Problem Statement
- Intro & Research Question

Related Work
- State of the Art & Open Issues

A Scalable RTDB Design
- InvaliDB: Concept & Prototype

Discussion
- Applications & Outlook

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

• System Comparison
  - Meteor
  - RethinkDB
  - Parse
  - Firebase
  - InvaliDB
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*?

repeat query every 10 seconds

monitor incoming writes

forward CRUD

CRUD

app server

app server

{ }
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>InvaliDB</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>
<tr>
<td><strong>Write Scalability</strong></td>
<td>✓</td>
<td>✗</td>
<td>✗</td>
<td>✗</td>
<td>✗</td>
</tr>
<tr>
<td><strong>Read Scalability</strong></td>
<td>✗</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td><strong>Composite Filters (AND/OR)</strong></td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td><strong>Sorted Queries</strong></td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✗</td>
<td>✓</td>
</tr>
<tr>
<td><strong>Limit</strong></td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✗</td>
<td>✓</td>
</tr>
<tr>
<td><strong>Offset</strong></td>
<td>✓</td>
<td>✓</td>
<td>✗</td>
<td>✗</td>
<td>✓</td>
</tr>
<tr>
<td><strong>Self-Maintaining Queries</strong></td>
<td>✓</td>
<td>✓</td>
<td>✗</td>
<td>✗</td>
<td>✓</td>
</tr>
<tr>
<td><strong>Event Stream Queries</strong></td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
</tbody>
</table>

- ✓: Available
- ✗: Not Available
- ?: Unknown
Outline

- Problem Statement
  - Intro & Research Question

- Related Work
  - State of the Art & Open Issues

- A Scalable RTDB Design
  - InvaliDB: Concept & Prototype

- Discussion
  - Applications & Outlook

- System Model & Overview
  - Query Subscription
  - Write Ingestion
  - Change Propagation

- Real-Time Query Processing
  - Two-Dimensional Workload Partitioning

Problem Statement
Intro & Research Question

Related Work
State of the Art & Open Issues

A Scalable RTDB Design
InvaliDB: Concept & Prototype

Discussion
Applications & Outlook

- System Model & Overview
  - Query Subscription
  - Write Ingestion
  - Change Propagation

- Real-Time Query Processing
  - Two-Dimensional Workload Partitioning
InvaliDB: A Scalable Real-Time Database Design
System Model & Overview

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

Event Layer
Application Server
Pull-Based Database
InvaliDB Cluster
InvaliDB: A Scalable Real-Time Database Design
Two-Dimensional Workload Partitioning
InvaliDB: A Scalable Real-Time Database Design
Two-Dimensional Workload Partitioning
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
Production System

**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

![Diagram](image-url)
Outline

- **Problem Statement**
  - Intro & Research Question

- **Related Work**
  - State of the Art & Open Issues

- **A Scalable RTDB Design**
  - InvaliDB: Concept & Prototype

- **Application Scenarios**
  1. Real-Time Queries
  2. Query Caching

- **Future Work**
  - Extending Semantics
  - Trade-Offs & Tuning
  - New Use Cases

- **Summary & Contributions**
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 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.“
Use Case 2: Consistent Query Caching
Improving Pull-Based Query Performance
InvaliDB
A Scalable Real-Time Database Design

Traditional Databases:
- pull-based queries

InvaliDB:
- push-based queries

- Scalable & Fast
- Developer-friendly
- Legacy-Compatible

MongoDB Expressiveness

Continuous

wolle@baqend.com