ORESTES: Scalability and Low Latency for Polyglot Database Services

Felix Gessert, Norbert Ritter
felix.gessert@gmail.com
About me

- Since 2 years: PhD student (advisor: Norbert Ritter)
- Since 1 year: CEO of Baqend

Research Project for PhD

Orestes

Cloud Database Startup
Outline

Motivation

- Cloud Databases
- The Latency Problem of the web

Cache Sketch Approach

Polyglot Persistence Mediator

Wrap-up and future work
Introduction: Which classes of cloud databases are there?
Cloud Databases

- **Infrastructure-as-a-Service**
  - Amazon RDS
  - SQL Azure
  - Heroku Pos.
  - Parse
  - Cloudant
  - Compose
  - DynamoDB
  - Google F1
  - BigQuery
  - EMR
  - Orestes

- **Platform-as-a-Service**
  - Heroku

- **Managed RDBMS**
Cloud Databases

- Infrastructure-as-a-Service
  - Managed RDBMS
  - Cloud-Deployment of DBMSs

- Platform-as-a-Service
  - Database-as-a-Service
    - Orestes
    - Parse
    - SQL Azure
    - Amazon RDS
    - Heroku Pos.
    - Compose
    - Cloudant
    - Google F1
    - Parse
    - Orestes
    - DynamoDB
    - BigQuery
    - EMR
    - Google F1
    - BigQuery
    - EMR
    - EMR

- Infrastructure-as-a-Service
  - Amazon RDS
  - SQL Azure
Cloud Databases

Managed NoSQL Databases

Cloud-only DBaaS-Systems

Orestes
Parse
Cloudant
Compose
DynamoDB
Google F1
BigQuery
EMR

Database-as-a-Service

Platform-as-a-Service

Infrastructure-as-a-Service

Cloud-Deployment of DBMSs

Managed RDBMS

Cloud Deployment of DBMSs
Cloud Databases

Managed NoSQL Databases
- Orestes
- Parse
- Cloudant
- SQL Azure
- Amazon RDS
- Heroku Pos.

Cloud-only DBaaS-Systems
- Compose
- DynamoDB
- Google F1
- BigQuery
- Cloudant
- Google F1
- EMR

Database-as-a-Service
- Orestes
- Parse
- Cloudant
- SQL Azure
- Amazon RDS
- Heroku Pos.

Platform-as-a-Service
- Managed RDBMS

Infrastructure-as-a-Service
- Cloud-Deployment of DBMSs

Analytics-as-a-Service
- Cloud-Deployment of DBMSs
Cloud Databases

- Backend-as-a-Service
  - Orestes
  - Compose
  - Cloudant
  - SQL Azure
  - Amazon RDS
  - Heroku Pos.

- Managed NoSQL Databases
  - DynamoDB
  - Google F1

- Cloud-only DBaaS-Systems
  - BigQuery
  - EMR

- Database-as-a-Service
  - Managed RDBMS

- Analytics-as-a-Service
  - Cloud Deployment of DBMSs

- Platform-as-a-Service

- Infrastructure-as-a-Service
Motivation
Motivation
Motivation

3-Tier Architecture

Database

Web Server

Web Server

Application

Internet

Client

Dynamic Web App
Motivation

3-Tier Architecture

Single Page Application

Database

Web Server

Web Server

Application

Internet

Data (e.g. JSON)

Client

Dynamic Web App
Motivation

3-Tier Architecture

Single Page Application

Database

Application

Web Server

Web Server

Data (e.g. JSON)

Internet

Client

Redundance

Applications reimplement backend functionality
**Motivation**

3-Tier Architecture

Single Page Application

Database

Application

Client

Data (e.g. JSON)

**Redundance**
Applications reimplement backend functionality

**Database Dependence**
Strong coupling between application and database system
Motivation

3-Tier Architecture

Database

Web Server

Web Server

Application

Internet

Client

Data (e.g. JSON)

Redundance
Applications reimplement backend functionality

Performance & Scalability
Error-prone application-specific scaling policies

Database Dependence
Strong coupling between application and database system
Motivation

3-Tier Architecture

Database

Web Server

Web Server

Application

Data (e.g. JSON)

Client

Internet

Single Page Application

Redundance
Applications reimplement backend functionality

Performance & Scalability
Error-prone application-specific scaling policies

Database Dependence
Strong coupling between application and database system

Difficult Development
Both client and server have to be implemented
Motivation

3-Tier Architecture

Single Page Application

Database

Application

Web Server

Web Server

Internet

Data (e.g. JSON)

Client

With every 100ms of additional page load time, revenue decreases by 1%.

Study by Amazon
When increasing load time of search results by **500ms**, traffic decreases by 20%.

Study by Google
Motivation

3-Tier Architecture

Single Page Application

---

**Page Load Time as bandwidth increases**

- 1 Mbps: 3500 ms
- 2 Mbps: 2000 ms
- 3 Mbps: 1500 ms
- 4 Mbps: 1200 ms
- 5 Mbps: 1000 ms
- 6 Mbps: 800 ms
- 7 Mbps: 600 ms
- 8 Mbps: 400 ms
- 9 Mbps: 300 ms
- 10 Mbps: 200 ms

**Page Load Time as latency decreases**

- 200 ms: 3500 ms
- 180 ms: 3000 ms
- 160 ms: 2500 ms
- 140 ms: 2000 ms
- 120 ms: 1500 ms
- 100 ms: 1000 ms
- 80 ms: 500 ms
- 60 ms: 0 ms
- 40 ms: 0 ms
- 20 ms: 0 ms

---

With every 100 ms of additional page load time, revenue decreases by 1%

*Study by Amazon*

When increasing load time of search results by 500 ms, traffic decreases by 20%.

*Study by Google*
Vision: ORESTES

Cloud

DB-Cluster

REST Server

REST Server

Orestes

Internet

Low Latency

Rich Client

Dynamic Web App
Vision: ORESTES

Automated Choice of Database System (Polyglot Persistence)
Automated Choice of Database System (Polyglot Persistence)
Vision: ORESTES

- Automated Choice of Database System (Polyglot Persistence)
- Unified REST API
- Low Latency
Vision: ORESTES

Automated Choice of Database System (Polyglot Persistence)

Unified REST API

Low Latency
Vision: ORESTES

- Automated Choice of Database System (Polyglot Persistence)
- Extensible High-Level Database and Backend Abstractions:
  - User-Management
  - Access Control
  - Schema
  - Transactions

Unified REST API

Low Latency

DB-Cluster

Orestes

Rich Client

REST Server

REST Server

Internet

Dynamic Web App
Vision: **ORESTES**

- **DB-Cluster**
- **Orestes**
- **Rich Client**

**Unified REST API**

**Low Latency**

**Transparency Caching**

**Automated Choice of Database System (Polyglot Persistence)**

**Extensible High-Level Database and Backend Abstractions:**
- User-Management
- Access Control
- Schema
- Transactions
Outline

- Motivation
- Cache Sketch
- Polyglot Persistence Mediator
- Wrap-up and future work
- REST/HTTP API
- HTTP Caching
- The Cache Sketch
  - Principle
  - Construction
  - Use
Unified REST API

- Platform-specific interfaces map to unified REST API
Unified REST API

- Platform-specific interfaces map to unified REST API

Diagram:
- User
- Browser
- SDK
- Orestes

Register
Platform-specific interfaces map to unified REST API

```
var usr = new User(name, pw);
usr.register();
```
Unified REST API

- Platform-specific interfaces map to unified REST API

User → Browser

Register

`var usr=new User(name, pw);`  
`usr.register()`  

SDK

POST `db/_native.User/`  
`JSON Object`

Orestes
Unified REST API

- Platform-specific interfaces map to unified REST API

User → Browser

Register

```javascript
var usr = new User(name, pw);
usr.register();
```

SDK

POST `db/_native.User/JSON Object`

Orestes
Unified REST API

Platform-specific interfaces map to unified REST API

User → Register

Browser

var usr = new User(name, pw);
usr.register();
Unified REST API

- Platform-specific interfaces map to unified REST API

```
Register
```

```
var usr=new User(name, pw);
usr.register();
```

- REST API leverages existing HTTP infrastructure
  - Load-Balancer (*stateless communication REST constraint*)
  - Web-Caches (*caching REST constraint*)
Expiration-based Caching

HTTP Caching Model:
- **Expiration-based** with defined *TTL*
- **Revalidations** check freshness at the server
Expiration-based Caching

HTTP Caching Model:
- **Expiration-based** with defined TTL
- **Revalidations** check freshness at the server

Research Question:
Can database services leverage the web caching infrastructure for low latency with rich consistency guarantees?
The Cache Sketch approach

Client

Expiration-based Caches

Request Path

Cache Hits

Invalidation-based Caches

Browser Caches, Forward Proxies, ISP Caches

Content Delivery Networks, Reverse Proxies

Server/DB
The Cache Sketch approach

Client

Expiration-based Caches

Needs Revalidation?

Browser Caches, Forward Proxies, ISP Caches

Invalidation-based Caches

Content Delivery Networks, Reverse Proxies

Server/DB

Request Path

Cache Hits
The Cache Sketch approach
The Cache Sketch approach

Client

Expiration-based Caches

Request Path
Cache Hits

Invalidation-based Caches

Invalidations, Records

Server/DB

Needs Revalidation?

10101010 Bloom filter

Client Cache Sketch

Browser Caches, Forward Proxies, ISP Caches

Content Delivery Networks, Reverse Proxies

Needs Revalidation?

Needs Invalidation?
The Cache Sketch approach

Client Cache Sketch

Expiration-based Caches

Invalidation-based Caches

Server/DB

Needs Invalidation?

Needs Revalidation?

Request Path

Cache Hits

Invalidations, Records

Browser Caches, Forward Proxies, ISP Caches

Content Delivery Networks, Reverse Proxies

10101010 Bloom filter
The Cache Sketch approach

Client

Expiration-based Caches

Invalidation-based Caches

Request Path

Cache Hits

Invalidations, Records

Server/DB

Needs Revalidation?

Browser Caches, Forward Proxies, ISP Caches

Needs Invalidation?

4

Content Delivery Networks, Reverse Proxies

Report Expirations and Writes

Needs Invalidation?

Non-expired Record Keys

Counting Bloom Filter

10101010

Bloom filter

Client Cache Sketch

Server Cache Sketch

10101010

10201040
The Cache Sketch approach

- **Client**: Needs Revalidation?
- **Expiration-based Caches**
- **Invalidation-based Caches**
- **Server/DB**: Report Expirations and Writes

**Request Path**

- **Cache Hits**
- **Invalidations, Records**

**Browser Caches, Forward Proxies, ISP Caches**

**Content Delivery Networks, Reverse Proxies**

**Client Cache Sketch**

- **Bloom filter**
- **Periodic every \( \Delta \) seconds**
- **at connect**

**Server Cache Sketch**

- **Non-expired Record Keys**
- **Counting Bloom Filter**

**Needs Invalidation?**
The Cache Sketch approach

Client

Expiration-based Caches

Request Path

Cache Hits

Invalidation-based Caches

Invalidations, Records

Server/DB

Needs Revalidation?

Browser Caches, Forward Proxies, ISP Caches

Content Delivery Networks, Reverse Proxies

10101010 Bloom filter

Client Cache Sketch

Periodic every \( \Delta \) seconds

at connect

Report Expirations and Writes

Non-expired Record Keys

10101010

Counting Bloom Filter

Server Cache Sketch

Needs Invalidation?

10201040

Non-expired Record Keys

Report Expirations and Writes

Invalidations, Records
The Caching Hierarchy

Client - (Browser-) Cache
Proxy Caches
ISP Caches
CDN Caches
Reverse-Proxy Caches
Orestes

Hit

Miss
Miss
Miss
Miss
Miss
The Caching Hierarchy

find(key) JavaScript

Dynamic Web App

Client-(Browser-)Cache Proxy Caches ISP Caches CDN Caches Reverse-Proxy Caches Orestes

Hit Miss Miss Miss Miss Miss

JavaScript

Client-(Browser-) Cache
Proxy Caches
ISP Caches
CDN Caches
Reverse-Proxy Caches
Orestes
The Caching Hierarchy

GET /db/posts/{id} HTTP

Dynamic Web App

Client- (Browser-) Cache

Proxy Caches

ISP Caches

CDN Caches

Reverse-Proxy Caches

Orestes

Hit

Miss

Miss

Miss

Miss

Miss

Miss
The Caching Hierarchy

HTTP

GET /db/posts/{id}

Client-
(Browser-) 
Cache

Proxy Caches

ISP Caches

CDN Caches

Reverse-
Proxy Caches

Orestes

Dynamic Web App

Hit

Miss

Miss

Miss

Miss

Miss
The Caching Hierarchy

Cache-Hit: Return Object
Cache-Miss or Revalidation: Forward Request
The Caching Hierarchy

Return record from DB with caching TTL
The Caching Hierarchy

Scalability and Cache-Hits
The Caching Hierarchy

Scalability and Cache-Hits

Latency Benefit
The Caching Hierarchy

Scalability and Cache-Hits

Low Latency
Less Load on Database Service
Protection Against Flash Crowds
Better Availability
Clients Profit from Each Other
Let $c_t$ be the client Cache Sketch generated at time $t$, containing the key $key_x$ of every record $x$ that was written before it expired in all caches, i.e. every $x$ for which holds:

$$\exists r(x, t_r, TTL), w(x, t_w) : t_r + TTL > t > t_w > t_r$$
Let $c_t$ be the client Cache Sketch generated at time $t$, containing the key $key_x$ of every record $x$ that was written before it expired in all caches, i.e. every $x$ for which holds:

$$\exists r(x, t_r, TTL), w(x, t_w) : t_r + TTL > t > t_w > t_r$$
Problem 1: Slow initial page loads

Solution: **Cached Initialization**
- Clients load the Cache Sketch at connection
- Every non-stale cached record can be reused without degraded consistency
Problem I: Slow initial page loads

Solution: **Cached Initialization**

- Clients load the Cache Sketch at connection
- Every non-stale cached record can be reused without degraded consistency
Problem 1: Slow initial page loads

Solution: **Cached Initialization**
- Clients load the Cache Sketch at connection
- Every non-stale cached record can be reused without degraded consistency
Problem 1: Slow initial page loads

Solution: **Cached Initialization**
- Clients load the Cache Sketch at connection
- Every non-stale cached record can be reused without degraded consistency
Problem 1: Slow initial page loads

Solution: **Cached Initialization**
- Clients load the Cache Sketch at connection
- Every non-stale cached record can be reused without degraded consistency
Problem I: Slow initial page loads

Solution: **Cached Initialization**

- Clients load the Cache Sketch at connection
- Every non-stale cached record can be reused without degraded consistency
**Problem 1: Slow initial page loads**

- **Solution:** **Cached Initialization**
  - Clients load the Cache Sketch at connection
  - Every non-stale cached record can be reused without degraded consistency
Problem 1: Slow initial page loads

Solution: **Cached Initialization**
- Clients load the Cache Sketch at connection
- Every non-stale cached record can be reused without degraded consistency

![Diagram showing database, cache, and load operations with hash functions and data entries.](image)
Problem 1: Slow initial page loads

Solution: **Cached Initialization**
- Clients load the Cache Sketch at connection
- Every non-stale cached record can be reused without degraded consistency
**Problem 1: Slow initial page loads**

- **Solution:** **Cached Initialization**
  - Clients load the Cache Sketch at connection
  - Every non-stale cached record can be reused without degraded consistency
Problem I: Slow initial page loads

Solution: **Cached Initialization**
- Clients load the Cache Sketch at connection
- Every non-stale cached record can be reused without degraded consistency

\[ f \approx \left(1 - e^{-\frac{kn}{m}}\right)^k \]

Hash-Functions:
\[ k = \left\lfloor \ln(2) \cdot \left(\frac{n}{m}\right) \right\rfloor \]

False-Positive Rate:

With 20,000 distinct updates and 5% error rate: **14 KByte**
**Problem 1: Slow initial page loads**

Solution: **Cached Initialization**
- Clients load the Cache Sketch at connection
- Every non-stale cached record can be reused without degraded consistency
Problem II: Slow CRUD performance

- Solution: \(\Delta\)-Bounded Staleness
  - Clients refresh the Cache Sketch so its age never exceeds \(\Delta\)
  \(\rightarrow\) Consistency guarantee: \(\Delta\)-atomicity
**Problem II: Slow CRUD performance**

- **Solution:** **Δ-Bounded Staleness**
  - Clients refresh the Cache Sketch so its age never exceeds \( \Delta \)
  - \( \rightarrow \) **Consistency guarantee:** \( \Delta \)-atomicity
Solution: Δ-Bounded Staleness
- Clients refresh the Cache Sketch so its age never exceeds Δ

→ Consistency guarantee: Δ-atomicity
Problem II: Slow CRUD performance

- Solution: Δ-Bounded Staleness
  - Clients refresh the Cache Sketch so its age never exceeds Δ
  - → Consistency guarantee: Δ-atomicity
**Problem II: Slow CRUD performance**

- **Solution:** Δ-Bounded Staleness
  - Clients refresh the Cache Sketch so its age never exceeds Δ
  
    → *Consistency guarantee*: Δ-atomicity
Problem II: Slow CRUD performance

- Solution: $\Delta$-Bounded Staleness
  - Clients refresh the Cache Sketch so its age never exceeds $\Delta$
  - $\rightarrow$ Consistency guarantee: $\Delta$-atomicity
Problem III: High Abort Rates in OCC

Solution: Conflict-Avoidant Optimistic Transactions

- Cache Sketch fetched with transaction begin
- Cached reads → Shorter transaction duration → less aborts
**Problem III: High Abort Rates in OCC**

- **Solution:** *Conflict-Avoidant Optimistic Transactions*
  - Cache Sketch fetched with transaction begin
  - *Cached reads → Shorter transaction duration → less aborts*
Problem III: High Abort Rates in OCC

Solution: **Conflict-Avoidant Optimistic Transactions**
- Cache Sketch fetched with transaction begin
- **Cached reads → Shorter transaction duration → less aborts**
**Problem III: High Abort Rates in OCC**

- **Solution:** *Conflict-Avoidant Optimistic Transactions*
  - Cache Sketch fetched with transaction begin
  - *Cached reads → Shorter transaction duration → less aborts*

![Diagram showing transaction process with Bloom Filter, Cache, REST-Server, and DB]

- **Begin Transaction**
- **Bloom Filter**
- **Reads**
  - *Cache*
- **Writes**
  - *Cache*
- **Commit:** read- & write-set versions
  - Committed OR aborted + stale objects

- **Steps:**
  1. **REST-Server**
  2. **REST-Server**
  3. **REST-Server**

- **Writes (Hidden)**
- **DB**
- **Coordinator**
**Problem III: High Abort Rates in OCC**

- **Solution:** *Conflict-Avoidant Optimistic Transactions*
  - Cache Sketch fetched with transaction begin
  - **Cached reads → Shorter transaction duration → less aborts**

---

1. **Begin Transaction**
   - **Bloom Filter**
   - **Cache**

2. **Reads**
   - **Cache**
   - **REST-Server**

3. **Commit:** read- & write-set versions
   - **REST-Server**
   - Committed OR aborted + stale objects

4. **Write**
   - **Hidden**
   - **Validation**
   - **Coordinator**

5. **Commit:**
   - read- & write-set versions
   - Committed OR aborted + stale objects

6. **Validation**
   - **Prevent conflicting validations**

---

**Diagram:**
- **Client**
- **REST-Server**
- **Cache**
- **DB**
- **Coordinator**
- **Writes** (Hidden)
- **Commit**
Solution: **Conflict-Avoidant Optimistic Transactions**

- Cache Sketch fetched with transaction begin
- **Cached reads → Shorter transaction duration → less aborts**

![Diagram showing the process of conflict-avoidant transactions in OCC with steps for Begin Transaction, Bloom Filter, Cache, REST-Server, DB, Coordinator, and validation.]
The Server Cache Sketch

- Goal: Efficient *Generation of Cache Sketch* and *Invalidation Minimization*
- **Counting Bloom Filter** and *key → expiration* mapping

Add $key_x$ if $x$ unexpired

- **redis** Cache Sketch for Table **A**
- **redis** ... **B**
- **redis** ... **C**

Get Cache Sketch:

Union

(Bitwise AND)

https://github.com/Baqend/Orestes-Bloomfilter
Goal: Efficient Generation of Cache Sketch and Invalidation Minimization

- Counting Bloom Filter and key → expiration mapping

Add $key_x$ if $x$ unexpired

- Cache Sketch for Table A
- ... B
- ... C

Get Cache Sketch: Union (Bitwise AND)

https://github.com/Baqend/Orestes-Bloomfilter
**Problem:** if TTL $\ll$ time to next write, then it is contained in Cache Sketch unnecessarily long

**TTL Estimator:** finds „best“ TTL

TTL Estimator

Objective:
- maximize Cache Hits
- minimize Purges
- minimize Stale Reads
- bound Cache Sketch false positive rate
Problem: if $TTL \ll$ time to next write, then it is contained in Cache Sketch unnecessarily long

TTL Estimator: finds „best“ TTL
**Problem:** if $TTL \ll \text{time to next write}$, then it is contained in Cache Sketch unnecessarily long

**TTL Estimator:** finds „best“ TTL

- **Client**
  -Writes $\sim \text{Poisson}$
- **Caches**
- **Server**
  -$\lambda_m$: Miss Rate
  -$\lambda_w$: Write Rate
  -Collect $\lambda_m \lambda_w$ per record

**TTL Estimator**

Objective:
- maximize Cache Hits
- minimize Purges
- minimize Stale Reads
- bound Cache Sketch false positive rate

![Write CDF](image)

- $E[T_m]=19000$, $E[T_w]=30000$
YCSB Monte Carlo Caching Simulator (YMCA)

- **Goal:** Analysis of arbitrary caching architectures using the standard YCSB benchmark
  - **Metrics:** Latency, TP, Cache Hits, Stale Reads, Invalidations
  - **Training of TTL Estimator:** Hill Climber finds optimal params
Results: Simulation & real-world

Setup:

Page load times with **cached initialization** (YMCA):

![Graph showing page load times with cached initialization](image)

Average Latency for YCSB Workloads A and B (real):

![Graph showing average latency for YCSB Workloads A and B](image)
Outline

- Motivation
- Cache Sketch Approach
- Polyglot Persistence Mediator
- Wrap-up and future work

- Idea
- Process
- Evaluation
The Polyglot Persistence Mediator

- Fully transparent choice of DB, based on requirements (SLAs)
- 3-step-process:
  1. Requirements
  2. Resolution
  3. Mediation
The Polyglot Persistence Mediator

- Fully transparent choice of DB, based on requirements (SLAs)
- 3-step-process:
  1. Requirements
  2. Resolution
  3. Mediation

1. Define schema
2. Choose Materialization Model
   - Sticky Partitioning
     schema-node → db mapping
   - Primary Database
     materializes data with staleness bound

1. Requirements
The Polyglot Persistence Mediator

- Fully transparent choice of DB, based on requirements (SLAs)
- 3-step-process:
  1. Requirements
  2. Resolution
  3. Mediation
The Polyglot Persistence Mediator

- Fully transparent choice of DB, based on requirements (SLAs)
- 3-step-process:
  1. Requirements
  2. Resolution
  3. Mediation

<table>
<thead>
<tr>
<th>Annotation</th>
<th>Type</th>
<th>Annotated at</th>
</tr>
</thead>
<tbody>
<tr>
<td>Read Availability</td>
<td>Continuous</td>
<td>*</td>
</tr>
<tr>
<td>Write Availability</td>
<td>Continuous</td>
<td>*</td>
</tr>
<tr>
<td>Read Latency</td>
<td>Continuous</td>
<td>*</td>
</tr>
<tr>
<td>Write Latency</td>
<td>Continuous</td>
<td>*</td>
</tr>
<tr>
<td>Write Throughput</td>
<td>Continuous</td>
<td>*</td>
</tr>
<tr>
<td>Data Vol. Scalability</td>
<td>Non-Functional</td>
<td>Field/Class/DB</td>
</tr>
<tr>
<td>Write Scalability</td>
<td>Non-Functional</td>
<td>Field/Class/DB</td>
</tr>
<tr>
<td>Read Scalability</td>
<td>Non-Functional</td>
<td>Field/Class/DB</td>
</tr>
<tr>
<td>Elasticity</td>
<td>Non-Functional</td>
<td>Field/Class/DB</td>
</tr>
<tr>
<td>Durability</td>
<td>Non-Functional</td>
<td>Field/Class/DB</td>
</tr>
<tr>
<td>Replicated</td>
<td>Non-Functional</td>
<td>Field/Class/DB</td>
</tr>
<tr>
<td>Linearizability</td>
<td>Non-Functional</td>
<td>Field/Class</td>
</tr>
<tr>
<td>Read-your-Writes</td>
<td>Non-Functional</td>
<td>Field/Class</td>
</tr>
<tr>
<td>Causal Consistency</td>
<td>Non-Functional</td>
<td>Field/Class</td>
</tr>
<tr>
<td>Writes follow reads</td>
<td>Non-Functional</td>
<td>Field/Class</td>
</tr>
<tr>
<td>Monotonic Read</td>
<td>Non-Functional</td>
<td>Field/Class</td>
</tr>
<tr>
<td>Monotonic Write Scans</td>
<td>Non-Functional</td>
<td>Field/Class</td>
</tr>
<tr>
<td>Sorting</td>
<td>Functional</td>
<td>Field</td>
</tr>
<tr>
<td>Range Queries</td>
<td>Functional</td>
<td>Field</td>
</tr>
<tr>
<td>Point Lookups</td>
<td>Functional</td>
<td>Field</td>
</tr>
<tr>
<td>ACID Transactions</td>
<td>Functional</td>
<td>Class/DB</td>
</tr>
<tr>
<td>Conditional Updates</td>
<td>Functional</td>
<td>Field</td>
</tr>
<tr>
<td>Joins</td>
<td>Functional</td>
<td>Class/DB</td>
</tr>
<tr>
<td>Analytics Integration</td>
<td>Functional</td>
<td>Field/Class/DB</td>
</tr>
<tr>
<td>Fulltext Search</td>
<td>Functional</td>
<td>Field</td>
</tr>
<tr>
<td>Atomic Updates</td>
<td>Functional</td>
<td>Field/Class</td>
</tr>
</tbody>
</table>
PPM Step 2: Resolution

- The Provider resolves the requirements
- RANK algorithm recursively analyzes schema annotations
- For each schema element:
  - Find each DB that satisfies all binary requirements
  - Pick the one that has the best score according to utility function of historic or predicted metrics

Resolution 2
Provider

Capabilities for available DBs

1. Find optimal RANK(schema_root, DBs) through recursive descent using annotated schema and metrics

Resolution 2
PPM Step 2: Resolution

- The Provider resolves the requirements
- RANK algorithm recursively analyzes schema annotations
- For each schema element:
  - Find each DB that satisfies all binary requirements
  - Pick the one that has the best score according to utility function of historic or predicted metrics

Resolution

Provider

Capabilities for available DBs

1. Find optimal

\[ \text{RANK}(\text{schema\_root}, \text{DBs}) \]
through recursive descent using annotated schema and metrics

Resolution
PPM Step 2: Resolution

- The Provider resolves the requirements
- RANK algorithm recursively analyzes schema annotations
- For each schema element:
  - Find each DB that satisfies all binary requirements
  - Pick the one that has the best score according to utility function of historic or predicted metrics

```
RANK(schema_root, DBs) through recursive descent using annotated schema and metrics
```
PPM Step 2: Resolution

- The Provider resolves the requirements
- RANK algorithm recursively analyzes schema annotations
- For each schema element:
  - Find each DB that satisfies all binary requirements
  - Pick the one that has the best score according to utility function of historic or predicted metrics
PPM Step 3: Mediation

- The PPM routes data and operations to the chosen DBs
- In the primary database model, it triggers periodic materializations
- Metrics (latency, availability, etc.) are reported to the resolver

Diagram:
- Application
  - 1. CRUD, queries, transactions, etc.
  - Polyglot Persistence Mediator
    - Uses Routing Model
    - Triggers periodic materialization
- Periodic materializations to primary copy
- Mediation
Evaluation

Scenario: news articles with impression counts
Objective: low-latency top-k queries, high-throughput counts

Found Resolution
Evaluation

Scenario: news articles with impression counts
Objective: low-latency top-k queries, high-throughput counts

Found Resolution
Summary

- **Cache Sketch**: dual approach to web caching for database services
  - Consistent (Δ-atomic) *expiration-based* caching
  - *Invalidation-based* caching with minimal purges

- **Polyglot Persistence Mediator**: SLA-driven, fine-grained selection of database systems
Future & Current Work

- **Cache Sketch:**
  - Online learning of best *TTL estimation*
  - Quantify *COT* properties
  - *Query* result caching
  - Extend *YMCA* to replication and sharding architectures

- **Polyglot Persistence:**
  - Common requirements/SLA „library“
  - Implementation of *Resolution*
  - *Live Migration*
  - Scalable *metrics* aggregation
Future Work: Query Caching

Write record $x$
Future Work: Query Caching
Future Work: Query Caching

Write record $x$

Stream Processing System

Treat every matching query $q$ as changed

Non-expired Record and Query Keys

Counting Bloom Filter

Server Cache Sketch

ORESTES
Future Work: Query Caching

Stream Processing System

Write record $x$

Treat every matching query $q$ as changed

Server Cache Sketch

Non-expired Record and Query Keys

Counting Bloom Filter

ORESTES

Dynamic Web App

$x$

10101010
10201040
Future Work: Query Caching

Write record $x$ to ORESTES

Notification to Subscribers (WebSockets, SSE, Mobile Push)

Stream Processing System

Treat every matching query $q$ as changed

Server Cache Sketch

Non-expired Record and Query Keys

Counting Bloom Filter

10101010
10201040

Server Cache Sketch

10201040
10101010

Counting
Bloom Filter

Non-expired
Record and
Query Keys
Orestes as a startup, funded since March 2014
Thank you

Contact:

gessert@informatik.uni-hamburg.de

http://orestes.info
http://baqend.com