Towards Scalable Cloud Data Management

Felix Gessert, Norbert Ritter
gessert/ritter@informatik.uni-hamburg.de
3rd Workshop on Scalable Cloud Data Management

Co-located with the IEEE BigData Conference.
Santa Clara, CA, October 29th 2015.
Starting at 8am in Ballroom C.

Workshop Schedule

October 29, 2015

SCDM 2015

SCDM 2015 Workshop in Santa Clara, CA. The preliminary schedule is online. SCDM starts on Oct 29, 8am (Ballroom C) with a keynote by Russel Sears on "Purity and the future of scalable storage".
Outline

Motivation

ORESTES: a Cloud-Database Middleware

Solving Latency and Polyglot Storage

Wrap-up

• Cloud Data Management
• Cloud Database Models
• Research Challenges
Introduction: What are the challenges in Cloud Data Management?
Architecture

Typical Data Architecture:

- Analytics
- Reporting
- Data Mining

The era of one-size-fits-all database systems is over → Specialized cloud databases
Database Sweetspots

**IBM DB2**
- RDBMS
- General-purpose
- ACID transactions

**Greenplum**
- Parallel DWH
- Aggregations/OLAP for massive data amounts

**VoltDB**
- NewSQL
- High throughput relational OLTP

**HBase**
- Wide-Column Store
- Long scans over structured data

**mongoDB**
- Document Store
- Deeply nested data models

**riak**
- Key-Value Store
- Large-scale session storage

**Neo4j the graph database**
- Graph Database
- Graph algorithms & queries

**redis**
- In-Memory KV-Store
- Counting & statistics

**cassandra**
- Wide-Column Store
- Massive user-generated content
Cloud-Database Sweetspots

**Firebase**
*Realtime BaaS*
Communication and collaboration

**Azure Tables**
*Wide-Column Store*
Very large tables

**bonsai**
*Managed NoSQL*
Full-Text Search

**Amazon RDS**
*Managed RDBMS*
General-purpose ACID transactions

**Amazon DynamoDB**
*Wide-Column Store*
Massive user-generated content

**Google Cloud Storage**
*Object Store*
Massive File Storage

**Parse**
*Backend-as-a-Service*
Small Websites and Apps

**Amazon ElastiCache**
*Managed Cache*
Caching and transient storage

**Amazon Elastic MapReduce**
*Hadoop-as-a-Service*
Big Data Analytics
Cloud-Database Models

Data Model

- **structured**
  - RDBMS machine image
  - Managed RDBMS/DWH
  - RDBMS/DWH Service

- **unstructured**
  - NoSQL machine image
  - Managed NoSQL
  - NoSQL Service

Deployment Model

- **unmanaged**
  - cloud-deployed (IaaS/PaaS)
  - Proprietary DB & Cloud

- **managed**
  - Managed (cloud-hosted)
Cloud Data Management

- Research field tackling the *design, implementation, evaluation* and *application implications* of *database systems* in cloud environments:

  - Protocols, APIs, Caching
  - Load distribution, Auto-Scaling, SLAs
  - Workload Management, Metering
  - Replication, Partitioning, Transactions, Indexing
  - Application architecture, Data Models
  - Multi-Tenancy, Consistency, Availability, Query Processing, Security
Open Research Questions
Performance & Latency

- How can database systems support **novel application architectures** (e.g., single-page or real-time apps)?
- Can the **functionality-performance trade-off** popularized by the NoSQL movement be turned into a tunable runtime configuration?
- How can a DBaaS deliver **low latency** in face of distributed storage and application tiers?
Open Research Questions
Consistency & Transactionality

- Which consistency and transaction guarantees can be provided across (geo-)replicated, partitioned, possibly heterogeneous/polyglot database systems?
- How can the consistency-latency-availability trade-off be best exposed to applications and developers?
- Can the existing methods (quorum-based, consensus-based, master-slave, etc.) be reconciliated into a single approach and tied to application requirements?
- How can we replace CAP by a more fine-grained and nuanced consistency classification scheme?
Open Research Questions

Service-Level Agreements

- How can database SLAs be guaranteed in a virtualized, multi-tenant **cloud environment**?
- Can we derive Service-Level-Objectives that are easy enough to understand and maintain to be **practical**?

<table>
<thead>
<tr>
<th>Model</th>
<th>CAP</th>
<th>SLAs</th>
</tr>
</thead>
<tbody>
<tr>
<td>SimpleDB</td>
<td>Table-Store</td>
<td>CP</td>
</tr>
<tr>
<td>DynamoDB</td>
<td>Table-Store</td>
<td>CP</td>
</tr>
<tr>
<td>Azure Tables</td>
<td>Table-Store</td>
<td>CP</td>
</tr>
<tr>
<td>AE/Cloud DataStore</td>
<td>Entity-Group Store</td>
<td>CP</td>
</tr>
<tr>
<td>S3, Az. Blob, GCS</td>
<td>Object-Store</td>
<td>AP</td>
</tr>
</tbody>
</table>
How can SLAs be incorporated in autoscaling to optimize costs and minimize SLA violations?

**Open Research Questions**

**Service-Level Agreements**

- Overprovisioning:
  - SLAs met
  - Excess Capacities

- Underprovisioning:
  - SLAs violated
  - Usage maximized

Provisioned Resources:
- Overprovisioning
- Underprovisioning

---

Can the data system functions of storage, search, streaming and analytics be unified and integrated?

Is it possible to automate, optimize and learn the best choice of given database systems?

How can queries and data be routed to databases, so that SLAs & performance characteristics are met?
Outline

Motivation

ORESTES: a Cloud-Database Middleware

Solving Latency and Polyglot Storage

Wrap-up

- Two problems:
  - Latency
  - Polyglot Storage
- Vision: Orestes Middleware
Latency & Polyglot Storage

Two central problems

- Goal of ORESTES: Solve both problems through a scalable cloud-database middleware

If the application is *geographically distributed*, how can we guarantee *fast database access*?

If one size *doesn’t fit all* – how can *polyglot persistence* be leveraged on a declarative, automated basis?
Problem I: Latency

Average: 9.3s

-1% Revenue

1000 ms Loading...

-20% Traffic

-7% Conversions

-4% Visitors

-1% Revenue
If perceived speed is such an important factor

...what causes slow page load times?
State of the art
Two bottlenecks: latency and processing
Network Latency

The underlying problem of high page load times

---

The low-latency vision

Data is served by ubiquitous web-caches
The web’s caching model

Staleness as a consequence of scalability

Expiration-based
Every object has a defined Time-To-Live (TTL)

Revalidations
Allow clients and caches to check freshness at the server

Research Question:
Can database services leverage the web caching infrastructure for low latency with rich consistency guarantees?
Problem II: Polyglot Persistence
Current best practice

Research Question:
Can we automate the mapping problem?
Vision
Schemas can be annotated with requirements

- Write Throughput > 10,000 RPS
- Read Availability > 99.9999%
- Scans = true
- Full-Text-Search = true
- Monotonic Read = true
Vision
The Polyglot Persistence Mediator chooses the database

Application

Polyglot Persistence Mediator

Database Metrics

Data and Operations

Latency < 30ms

Annotated Schema
The Big Picture
Implementation in ORESTES

Polyglot Storage and Low Latency are the central goals of ORESTES.
Unified REST API
Standard HTTP Caching

Database-as-a-Service Middleware:
Caching, Transactions, Schemas,
Authorization, Polyglot Storage

Content-Delivery-Network

Desktop
Mobile
Tablet

Information Storage and Low Latency are the central goals of ORESTES.
Unified REST API
Standard HTTP Caching

Outline

Motivation

ORESTES: a Cloud-Database Middleware

Solving Latency and Polyglot Storage

Wrap-up

• Cache Sketch Approach
  • Caching Objects
  • Caching Query Results
  • Continuous Queries

• Polyglot Persistence Mediator
  • Resolution
  • Mediation
  • Polyglot Materialized Views

RESTES: a Cloud-Database Middleware

Motivation

Solving Latency and Polyglot Storage

Wrap-up
The Cache Sketch approach
Letting the client handle cache coherence

Client

Expiration-based Caches

Request Path

Cache Hits

Server/DB

Invalidation-based Caches

Invalidations, Records

Browser Caches, Forward Proxies, ISP Caches

Content Delivery Networks, Reverse Proxies

Needs Revalidation?

Bloom filter

Periodic every Δ seconds at transaction begin

10101010

Non-expired Record Keys

10101010

10201040

Counting Bloom Filter

Report Expirations and Writes

Needs Invalidation?

Server Cache Sketch

10101010

10201040

Counting Bloom Filter

Client Cache Sketch

10101010

Bloom filter

Connect Periodic every Δ seconds at transaction begin

Client Cache Sketch

10101010
Visually Explained
Cache Sketch in Action

False-Positive Rate: \( f \approx (1 - e^{-\frac{kn}{m}})^k \)

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

With 20,000 distinct updates and 5% error rate: **11 KByte**

\( \text{hashA(oid)} \) \( \text{hashB(oid)} \)
Object Caching
Summary of Properties

- **Consistency guarantee:** \( \Delta \)-atomicity

- **Modes:**
  - *Cached initialization:* piggybacked Cache Sketch enables fast page loads
  - *Bounded Staleness:* application refreshes Cache Sketch in fixed intervals
  - *Conflict-Avoidant Optimistic Transactions:* guarantee ACID despite cached reads

- **TTL Estimator:** learns and (statistically) estimates appropriate expirations

---

Felix Gessert, Florian Bucklers, Norbert Ritter: *Orestes: A scalable Database-as-a-Service architecture for low latency.* CloudDB2014@ICDE.

From Object Caching to Query Caching
Generalizing the Cache Sketch to query results

- **Main challenge**: when to invalidate?
  - **Objects**: for every update and delete
  - **Queries**: when the query result changes

→ How to detect query result changes in real-time?

\[ \{\text{obj}_1, \text{obj}_2, \text{obj}_3\} \]

- **Add Event**: \( \text{obj} \notin \text{Q} \) Inserted or updated so that \( \text{P} \) matches
- **Remove Event**: \( \text{obj} \in \text{Q} \) updated or deleted so that \( \text{P} \) no longer matches
- **Change Event**: \( \text{obj} \in \text{Q} \) updated so that \( \text{P} \) still matches

Query result \( \text{Q} \) for predicate \( \text{P} \)
Query Caching

Example

Add, Change, Remove all entail an invalidation and addition to the cache sketch.
Architecture

ORESTES

Create
Update
Delete

Continuous Queries (Websockets)

Polyglot Views

Fresh Caches

Fresh Cache Sketch

Pub-Sub

Architecture
Generalizing the Cache Sketch to Query Results

- Query
- CUD Operations

DBaaS-Server

- Query & Id-lists of results
- After-Images of operations

Distribution Layer

Match-Events/Invalidations
Query State Updates

Redis

DBaaS-Server

DBaaS-Server

- Invalidations
- Cache Sketch

Shared: Cache Sketch
Local: Access Metrics

State: Active Queries, Id-lists of results
PubSub: updates, queries

Matching, partitioned by queries and objects

Streaming Layer

Storm
Streaming Layer

Query Matching

Design goals:
• Scalability
• Elasticity
• Low Latency

SELECT * FROM posts
WHERE tags CONTAINS 'example'
Optimal Query Representation

Id-Lists
\{id_1, id_2, id_3\}

Invalidate by: Add, Remove
\rightarrow less invalidations

Performance: at least two network round-trips

Object-Lists
\{\{id: 1, tag: 'a'\}, \{id: 2, tag: 'b'\}, \{id: 3, tag: 'c'\}\}

Invalidate by: Add, Remove, Change

Performance: one round-trip
\rightarrow lower latency

Cost-based decision model:

\[ w \frac{\text{changes}}{\text{removes} + \text{adds} + \text{changes}} > 1 - \frac{1}{1 + [\text{resultsize} / \text{connections}]} \]

Fraction of avoided invalidations

avoided round-trips
Query Lifecycle
Disitributed Capacity Management

- Matching capacity is limited
  - Always cache hinted queries
  - Allocate available capacity (best-effort queries)

If query is hinted → always add
If capacity available → greedily add and estimate TTL

If query was not recently reactivated discard its metrics (invalidation counters, TTLs, etc.)

Active queries are matched until they are invalidated or expired
Continuous Queries
Complementing Cached Queries

- Same streaming architecture can similarly notify applications (browsers) about query result changes

Application Pattern:
Matching Performance
Latency of detecting invalidations

- Latency mostly < 15ms, scales linearly w.r.t. number of servers and number of tables

![Graph showing matching performance](image)
Performance

Setup:

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

- With Facebook’s cache hit rate: \(>2.5x\) improvement

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

- 95% Read 5% Writes \(\rightarrow\) 5x latency improvement
If the application is geographically distributed, how can we guarantee fast database access?

Transparent **end-to-end caching** using the Cache Sketch.

If one size *doesn’t* fit all – how can **polyglot persistence** be leveraged on a declarative, automated basis?
Towards Automated Polyglot Persistence

Necessary steps

- **Goal:**
  - Extend classic workload management to *polyglot persistence*
  - Leverage heterogeneous (NoSQL) databases

1. **Requirements**
   - Tenant specifies requirements as Service-Level-Agreements

2. **Resolution**
   - Find or provision a suitable combination of databases

3. **Mediation**
   - Mediate data and database operations
Step I - Requirements
Expressing the application’s needs

<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/DB</td>
</tr>
<tr>
<td>Read-your-Writes</td>
<td>Non-Functional</td>
<td>Field/Class/DB</td>
</tr>
<tr>
<td>Causal Consistency</td>
<td>Non-Functional</td>
<td>Field/Class/DB</td>
</tr>
<tr>
<td>Writes follow reads</td>
<td>Non-Functional</td>
<td>Field/Class/DB</td>
</tr>
<tr>
<td>Monotonic Read</td>
<td>Non-Functional</td>
<td>Field/Class/DB</td>
</tr>
<tr>
<td>Monotonic Write</td>
<td>Non-Functional</td>
<td>Field/Class/DB</td>
</tr>
<tr>
<td>Scans</td>
<td>Functional</td>
<td>Field</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/DB</td>
</tr>
</tbody>
</table>

Annotations
- Continuous non-functional e.g. write latency < 15ms
- Binary functional e.g. Atomic updates
- Binary non-functional e.g. Read-your-writes

1. Define schema
2. Annotate

Tenant

Database

Table

Field Field Field Field

annotated

Inherits continuous annotations

1 Requirements
Step II - Resolution
Finding the best database

- The Provider resolves the requirements
- **RANK**: scores available database systems
- **Routing Model**: defines the optimal mapping from schema elements to databases
Step III - Mediation
Routing data and operations

- The PPM routes data
- **Operation Rewriting:** translates from abstract to database-specific operations
- **Runtime Metrics:** Latency, availability, etc. are reported to the resolver
- **Primary Database Option:** All data periodically gets materialized to designated database
Evaluation: News Article
Prototype of Polyglot Persistence Mediator in ORESTES

**Scenario:** news articles with impression counts
**Objectives:** low-latency top-k queries, high-throughput on counter, article-queries

[Image of Hacker News article]

*Announcing MongoDB 3.0 (mongodb.com)*

196 points by meghan 142 days ago | 144 comments | in pocket speicher

read by 1.344.222
Evaluation: News Article
Prototype built on ORESTES

**Scenario:** news articles with impression counts

**Objectives:** low-latency top-k queries, high-throughput on counter, article-queries

Counter updates kill performance
Evaluation: News Article
Prototype built on ORESTES

Scenario: news articles with impression counts
Objectives: low-latency top-k queries, high-throughput on counter, article-queries

No powerful queries
Evaluation: News Article
Prototype built on ORESTES

**Scenario:** news articles with impression counts
**Objectives:** low-latency top-k queries, high-throughput on counter, article-queries

![Diagram](chart.png)

*Found Resolution*
Polyglot Materialized Views
Arbitrary Queries over arbitrary databases

- **Approach:**
  - **Mediator** emits change data stream (after-images)
  - **Streaming layer** maintains registered materialized views using pluggable query engines
  - **Serving layer** stores materialized views and serves them to applications
Outline

• Motivation

ORESTES: a Cloud-Database Middleware

Solving Latency and Polyglot Storage

Wrap-up

• Current/Future Work
• Summary
• Putting ORESTES into practice
Summary

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

- **Query Caching**:
  - Invalidations and Cache Sketch updates in real-time
  - Cache-optimal representation of results

- **Continuous & Materialized Queries**
  - Real-time updates to query results

- **Polyglot Persistence Mediator**:
  - SLA-based routing of queries and data to appropriate database systems
What impact does the Cache Sketch have?
Thanks a lot!

gessert/ritter@informatik.uni-hamburg.de
aqend.com, orestes.info