Towards Automated Polyglot Persistence

Michael Schaarschmidt, Felix Gessert, Norbert Ritter
gessert@informatik.uni-hamburg.de
Polyglot Persistence
Current best practice

Application Layer

- Billing Data
- Nested Application Data
- Session data
- Search Index
- Files
- Recommendation Engine

Friend network
Cached data & metrics
DB2
MongoDB
the graph database
Redis
Elasticsearch
Google Cloud Storage
Amazon Elastic MapReduce
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
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
Service Level Agreements
Expressing application requirements

**Functional** Service Level Objectives
- Guarantee a “feature”
- Determined by database system
- *Examples*: transactions, join

**Non-Functional** Service Level Objectives
- Guarantee a certain *quality of service* (QoS)
- Determined by database system and service provider
- *Examples*:
  - **Continuous**: response time (latency), throughput
  - **Binary**: Elasticity, Read-your-writes
Service Level Agreements
Refining the utility of each SLO

Utility expresses „value“ of a continuous non-functional requirement:

\[ f_{utility}(\text{metric}) \rightarrow [0,1] \]
SLA Example
For MongoDB

Functional Requirements
- Scan-Querys
- Transactions
- Conditional Updates
- Joins
- Query by Example
- Analytics

Non-Functional Requirements
- Scalability of Data Volume
- Write Scalability
- Read Scalability
- Elasticity
- Read-Availability
- Consistency
- Write-Availability
- Durability
- Read-Latency
- Write-Throughput
- Write-Latency

Durability
Write Scalability
Read Scalability
Elasticity
Read-Availability
Consistency
Write-Availability
Durability
Read-Latency
Write-Throughput
Write-Latency
Step 1 - Requirements
Expressing the application’s needs

- Tenant annotates schema with his requirements

1. Define schema
   - Tenant
   - Inherits continuous annotations
   - Annotates schema with his requirements

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

![Diagram](image.png)
Step 1 - 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</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</td>
<td>Non-Functional</td>
<td>Field/Class</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</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
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

![Diagram showing the process of finding the best database](image_url)

1. Find optimal
2a. If unsatisfiable: Refuse or Provision new DB
2b. Generates routing model

**RANK**(schema_root, DBs) through recursive descent using annotated schema and metrics

Routing Model
Route schema_element → db
- transform db-independent to db-specific operations
Step II - Resolution
Ranking algorithm by example

Annotations
- Linearizability
- Availability

Schema
- ECommerceDB database
  - Customers Table
    - ShoppingBasket List<String>
    - UserName String

Rank Algorithm
- No annotation → recursive descent to child

DBs = { MongoDB, Riak, Cassandra, CouchDB, Redis, MySQL, S3, Hbase }
Step II - Resolution
Ranking algorithm by example

Annotations
- Linearizability
- Availability

Schema
- ECommerceDB database
- Customers Table
- ShoppingBasket List<String>
- UserName String

RANK Algorithm
- DBs = { MongoDB, Riak, Cassandra, CouchDB, Redis, MySQL, S3, Hbase }

Binary requirement
1. Exclude DBs that do not support it
2. Recursive descent
Step II - Resolution
Ranking algorithm by example

Annotations
- Linearizability
- Availability

Schema
ECommerceDB
database

Customers
Table

ShoppingBasket
List<String>

String

RANK Algorithm

<table>
<thead>
<tr>
<th>Database</th>
<th>Availability</th>
</tr>
</thead>
<tbody>
<tr>
<td>MongoDB</td>
<td>99% → 0.8</td>
</tr>
<tr>
<td>Redis</td>
<td>95% → 0.05</td>
</tr>
<tr>
<td>MySQL</td>
<td>94% → 0.04</td>
</tr>
<tr>
<td>HBase</td>
<td>99.9% → 0.9</td>
</tr>
</tbody>
</table>

Continuous requirement → ∀ databases calculate

\[
db \rightarrow f_{utility}(db.\text{availability})
\]
Step II - Resolution
Ranking algorithm by example

Annotations
- Linearizability
- Availability

Schema
- ECommerceDB database
- Customers Table
  - ShoppingBasket: List<String>
  - UserName: String

RANK Algorithm

<table>
<thead>
<tr>
<th>Database</th>
<th>Availability</th>
<th>Latency</th>
</tr>
</thead>
<tbody>
<tr>
<td>MongoDB</td>
<td>99% → 0.8</td>
<td>10ms → 1</td>
</tr>
<tr>
<td>Redis</td>
<td>95% → 0.05</td>
<td>1ms → 1</td>
</tr>
<tr>
<td>MySQL</td>
<td>94% → 0.04</td>
<td>40ms → 0.2</td>
</tr>
<tr>
<td>HBase</td>
<td>99.9% → 0.9</td>
<td>50ms → 0.1</td>
</tr>
</tbody>
</table>

Continuous requirement → ∀ databases calculate

\[ db \rightarrow f_{utility}(db.\text{latency}) \]
Step II - Resolution

Ranking algorithm by example

Annotations

Linearizability
Availability

Schema

ECommerceDB database

Customers Table

ShoppingBasket List<String>
UserName String

DB Score
MongoDB 0.9
Redis 0.525
MySQL 0.12
HBase 0.5

Binary requirement →
1. Exclude DBs that do not support it
2. Recursive descent
3. Pick DB with best total score and add it to routing model
Step II - Resolution

Ranking algorithm by example

Annotations
- Linearizability
- Availability

Schema
- ECommerceDB database
- Customers Table
- ShoppingBasket List<String>
- UserName String

Routing Model:
- Customers ➔ MongoDB

DB | Score
---|---
MongoDB | 0.9
Redis | 0.525
MySQL | 0.12
HBase | 0.5

 expenditures

Binary requirement ➔
1. Exclude DBs that do not support it
2. Recursive descent
3. Pick DB with best total score and add it to routing model
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 built on ORESTES

Scenario: news articles with impression counts
Objectives: low-latency top-k queries, high-throughput counts, article-queries
Evaluation: News Article

Prototype built on ORESTES

Scenario: news articles with impression counts

Objectives: low-latency top-k queries, high-throughput counts, article-queries
Evaluation: News Article
Prototype built on ORESTES

Scenario: news articles with impression counts
Objectives: low-latency top-k queries, high-throughput counts, 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 counts, article-queries
Evaluation: News Article
Prototype built on ORESTES

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

![Diagram showing article structure and sorted set with MongoDB and Redis logos]

Found Resolution
Challenges & Future Work

**Workload Management**: during mediation actively schedule requests based on requirements

**Ranking**: Predict future metrics from historic ones (*time-series analysis*) or from performance models

**Database selection**: minimize $P(\text{SLA violation}) \times \text{penalty}$ (e.g. through reinforcement learning)
Challenges & Future Work

**Meta-DBaaS**: Mediate over DBaaS-systems and factor in their SLAs

**Live Migration**: Enable requirement changes

**Requirements**: collect library of common ones

**Utility**: Provide intuitive, visual „knobs“ for developers
Summary

- (Manual) Polyglot Persistence is a reality - but difficult and error-prone
- **Polyglot Persistence Mediator**: SLA-driven, fine-grained selection of database systems
  1. Let the tenant define his requirements
  2. Choose or provision a database based on that
  3. Route data and operations according to that mapping
Thank you.

gessert@informatik.uni-hamburg.de
Orestes.info
Baqend.com