Polyglot Data Management: State of the Art & Open Challenges

Felix Kiehn  felix.kiehn@uni-hamburg.de
Mareike Schmidt  mareike.schmidt-3@uni-hamburg.de
Daniel Glake  daniel.glake@uni-hamburg.de

Fabian Panse  fabian.panse@uni-hamburg.de
Wolfram Wingerath  wolle@uni-oldenburg.de
Benjamin Wollmer  benjamin.wollmer@uni-hamburg.de

Martin Poppinga  martin.poppinga@uni-hamburg.de
Norbert Ritter  norbert.ritter@uni-hamburg.de
Who We Are

Felix Kiehn

Martin Poppinga

Mareike Schmidt

Fabian Panse

Norbert Ritter

Daniel Glake

Wolfram Wingerath

Benjamin Wollmer
Outlook

PART I: Motivation & Database Landscape

PART II: Terminology & Taxonomies

PART III: Basic Techniques & Concepts

PART IV: Current Systems

PART V: Open Challenges
Part I

Motivation & Database Landscape
Use Case 1: E-Commerce

Products
- descriptions
- ratings
- prices
- recommendations

Payment
- transactions

ERP
- logistics
- business numbers
- forecasts

High Availability & Flexible Consistency
High Read Throuput
OLTP (High Consistency)
OLAP
Machine Learning

https://www.baqend.com/publications
Use Case 2: Medical Application Data (e.g. MIMIC)

- **Patient Information**: patient data, medication, doctor's notes, monitoring
- **Examination Results**: laboratory results, statistics, medical images
- **Payment**: transactions

Structured Data, High Availability, Full Text Search, Realtime, Complex Analytics, Machine Learning (High Consistency)

[https://physionet.org/content/mimiciii](https://physionet.org/content/mimiciii)
Use Case 3: Digital Twin (e.g. MARS)

Static Data (Environmental)
- buildings
- streets
- POI's
- satellite images

Dynamic Data
- moving objects
- social network
- decision making

Updates
- sensors
- detectors

High Read Throughput
- Vector
- Polygon
- Linestring
- Point
- Raster 1,3 Band

Realtime
- Graph
- High Write Throughput
- Machine Learning

https://mars.haw-hamburg.de
A Short History of Data Management

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

CEP & Streams
- MapReduce
- Spark
- Starburst
- TELEGRAPH
- HiPAC
- Rapide
- Bigtable

Stream Processing
- Samza
- Baqend
- Aurora & Borealis
- GFS
- Dynamo
- Flink
- Firebase
- RethinkDB
- Storm
- Meteor

Active Databases
- Relational Model

Big Data & NoSQL
- Bigtable
- Aurora & Borealis
- Dynamo
- Flink
- RethinkDB

Real-Time Databases
- Samza
- Baqend
- Aurora & Borealis
- GFS
- Dynamo
- Flink
- RethinkDB
- Storm
- Meteor

Not included:
- Timeseries DBs
- (Geo-)spatial DBs
- Object-oriented DBs
- Probabilistic DBs
- Graph Stores
- ...

Polyglot Data Management: State of the Art & Open Challenges
Typical Classification Schemes

Not included:
- Functional/non-functional properties
- Cloud vs. on-premise
- ACID vs. BASE
- ...

Data Model
- Relational
- Key-Value
- Wide-Column
- Document
- Graph

CAP Classes
- AP: Available & Partition Tolerant
- CP: Consistent & Partition Tolerant
- CA: Not Partition Tolerant

Pull vs. Push
- Databases: static collections
- Real-Time DBs: dynamic collections
- Stream Processing: dynamic streams
How to Choose The „Right“ Database System?

Billing Data

Friend network

Cached data & metrics

Nested Application Data

Search Index

Recommendation Engine

Session data

Files

Google Cloud Storage

Amazon Elastic MapReduce

IBM DB2

Neo4j the graph database

redis

mongoDB

elasticsearch.

cassandra
# NoSQL Toolbox: Requirements vs. Techniques

<table>
<thead>
<tr>
<th>Functional</th>
<th>Techniques</th>
<th>Non-Functional</th>
</tr>
</thead>
<tbody>
<tr>
<td>Scan Queries</td>
<td>Sharding</td>
<td>Data Scalability</td>
</tr>
<tr>
<td>ACID Transactions</td>
<td>Range-Sharding</td>
<td>Write Scalability</td>
</tr>
<tr>
<td>Conditional/In-Memory Ventures</td>
<td>Hash-Sharding</td>
<td>Read Scalability</td>
</tr>
<tr>
<td>Joins</td>
<td>Local Group Sharding</td>
<td>Elasticity</td>
</tr>
<tr>
<td>Sorting</td>
<td>Consistent Hashing</td>
<td>Consistency</td>
</tr>
<tr>
<td>Aggregation</td>
<td>Update</td>
<td>Read Availability</td>
</tr>
<tr>
<td>Analytics</td>
<td>Update in Place</td>
<td>Write Availability</td>
</tr>
<tr>
<td>Storage Management</td>
<td>Caching</td>
<td>Durability</td>
</tr>
<tr>
<td>Logging</td>
<td>In-Memory Storage</td>
<td></td>
</tr>
<tr>
<td>Update-in-Place</td>
<td>Append-Only Storage</td>
<td></td>
</tr>
<tr>
<td>Query Planning</td>
<td>Materialized Views</td>
<td></td>
</tr>
<tr>
<td>Pivotal NoSQL techniques</td>
<td>Global Secondary Indexing</td>
<td></td>
</tr>
<tr>
<td>application requirements</td>
<td>Local Secondary Indexing</td>
<td></td>
</tr>
<tr>
<td>enable</td>
<td>Query Planning</td>
<td></td>
</tr>
</tbody>
</table>

NoSQL Decision Tree

Access
- Fast Lookups
  - Volume
    - RAM
      - Redis
      - Cassandra
      - Memcache
    - Unbounded
      - Riak
      - Voldemort
      - Aerospike
  - Complex Queries
    - HDD-Size
      - CAP
        - AP
          - RDBMS
            - HBase
              - Neo4j
            - CouchBase
            - DynamoDB
        - CP
          - MongoDB
            - CouchDB
            - MongoDB
            - RethinkDB
            - HBase
            - Accumulo
            - ElaticSearch
            - Solr
          - Hadoop
            - Spark
            - Parallel DWH

Complex Queries
- Unbounded
  - ACID
    - Availability
      - Ad-hoc
        - CouchDB
        - MongoDB
        - RethinkDB
        - HBase, Accumulo
        - ElasticSearch, Solr
      - Analytics
        - Hadoop, Spark
        - Parallel DWH
        - Cassandra, HBase
        - Riak, MongoDB

Open Challenges:
- **Complex Trade-Offs** that may contradict one another
- **Complex Architectures** with many different data management systems
- **In-flux Requirements**: You may have to revisit your decision over time

Example Applications
- Cache
- Shopping-basket
- Order History
- OLTP
- Website
- Social Network
- Big Data
More on the Topic

For videos & books, visit dbis.hamburg!
Actual Question: How to Build a System That Does All This?
How to Choose The „Right“ Database System?

Research Question:
Can we automate the mapping problem?
Part II

Terminology & Taxonomies
Terminology & Taxonomies

Polyglot Persistence

Multistores

Modern Federated Database Systems

Hybrid Stores

Multidatabases

Federating (Specialized) Data Stores

Polystores

Generalized Data Federation
Query & Data Store based Taxonomy

- Query Interfaces:
  - Single
  - Multiple

- Data Stores:
  - Homogenous
  - Heterogenous
Query & Data Store based Taxonomy

- Query Interfaces:
  - Single
  - Multiple

- Data Stores:
  - Homogenous
  - Heterogenous

- Federated DB System
  Single interface, homogenous stores
Query & Data Store based Taxonomy

- **Query Interfaces:**
- **Data Stores:**

### Federated DB System
- Single interface, homogenous stores

### Polylingual / Polyglot DB System
- Multiple interfaces, homogenous stores
Query & Data Store based Taxonomy

- **Query Interfaces:**
- **Data Stores:**

 ![Diagram]

- **Single**
  - Federated DB System
    - Single interface, homogenous stores

- **Multiple**
  - Polylingual / Polyglot DB System
    - Multiple interfaces, homogenous stores

- **Heterogenous**
  - **MultiStore**
    - Single interface, heterogenous stores
Query & Data Store based Taxonomy

- **Query Interfaces:**

- **Data Stores:**

  - **Federated DB System**
    Single interface, homogenous stores
  
  - **Polylingual / Polyglot DB System**
    Multiple interfaces, homogenous stores
  
  - **MultiStore**
    Single interface, heterogenous stores
  
  - **PolyStore**
    Multiple interfaces, heterogenous stores
Query & Data Store based Taxonomy

- **Query Interfaces:**
  - Single
  - Multiple

- **Data Stores:**
  - Homogenous
  - Heterogenous

- **Federated DB System**
  - Single interface, homogenous stores

- **Polylingual / Polyglot DB System**
  - Multiple interfaces, homogenous stores

- **MultiStore**
  - Single interface, heterogenous stores

- **PolyStore**
  - Multiple interfaces, heterogenous stores
### Loosely vs. Tightly Coupled

**Loosely** (like a data integration scenario)

<table>
<thead>
<tr>
<th>Data:</th>
<th>External</th>
</tr>
</thead>
<tbody>
<tr>
<td>Store Autonomy:</td>
<td>High</td>
</tr>
<tr>
<td>Access:</td>
<td>Read-Only</td>
</tr>
<tr>
<td>Local Config.:</td>
<td>No</td>
</tr>
<tr>
<td>Data Quality:</td>
<td>Little control</td>
</tr>
<tr>
<td>Semantic Het.:</td>
<td>Possible</td>
</tr>
</tbody>
</table>

---

**Diagram**

- **Query Interface**
  - User
  - Admin
  - Query Engine
  - Data Stores:
    - S1
    - S2
    - S3

- Departments:
  - Department A
  - Department B
  - Department C

- S1: 10111 01001
- S2: 11011 01011
- S3: 00111 11011

---

Polyglot Data Management: State of the Art & Open Challenges
Loosely vs. Tightly Coupled

Query Interface

Query Engine

Loosely
(like a data integration scenario)

- Data: External
- Store Autonomy: High
- Access: Read-Only
- Local Config.: No
- Data Quality: Little control
- Semantic Het.: Possible

department A
department B
department C
Loosely vs. Tightly Coupled

Tightly (like a distributed system)

- Data: Internal
- Store Autonomy: None
- Access: Write & Read
- Local Config.: Yes
- Data Quality: High control
- Semantic Het.: None

able to migrate data between stores
able to reconfigure individual stores

user
admin / owner

Query Interface
Query Engine

S1  10111 01001

S2

S3  10111 01001

able to migrate data between stores
able to reconfigure individual stores
Evaluation Framework

- **Heterogeneity**: Data Stores, Processing Engines & Query Interfaces
- **Autonomy**: Association, Execution & Evolution
- **Transparency**: Location & Transformation
- **Flexibility**: Schema, Interface & Architectural
- **Optimality**: Federated Plan & Data Placement

Reference:
Part III

Basic Techniques & Concepts
Basic Techniques & Concepts: Overview

Mediator-Wrapper Architecture
Popular architecture of integration systems

Schema Mapping Languages
How to model relationships between schemas?

Joins
How to combine data from different stores?

Cross-Platform Query Planning
How to optimize queries across stores?
Mediator-Wrapper Architecture

Tasks:
• build query plan
• decompose query
• determine join order
• push subqueries
• orchestrate query execution
• combine subquery results

Tasks:
• translate query in local language / schema
• transform result records to global schema
Query Interface

Global-as-View (GaV)

Flexibility: Low
Query Processing: Simple
Number Views: Low
Complexity Views: High
Modeling Power: Medium

G1 ⊇ S1 ⋈ (S2 ⋈ S3)

G1 ⊈ G2

G1 global schema

local schemas

S1, S2, S3

S1 ⊇ S2 ⊇ S3

wrappers
Schema Mapping Languages

Global-as-View (GaV)

- Flexibility: Low
- Query Processing: Simple
- Number Views: Low
- Complexity Views: High
- Modeling Power: Medium

G2 \supseteq \pi_A(\sigma_c(S1)) \cup S3

\sigma_c

\pi_A

G1

G2

G1 - as View (GaV)

G2 \supseteq \pi_A(\sigma_c(S1)) \cup S3

S1

S2

S3

local schemas

Query Interface
Schema Mapping Languages

Local-as-View (LaV)

- Flexibility: High
- Query Processing: Complex
- Number Views: High
- Complexity Views: Low
- Modeling Power: Medium

S1 \subseteq G1 \bowtie \sigma_c(G2)
Schema Mapping Languages

Global-Local-as-View (GLaV)

- **Flexibility:** High
- **Query Processing:** Complex
- **Number Views:** Medium
- **Complexity Views:** Low - High
- **Modeling Power:** High

\[ S_2 \Join S_3 \subseteq G_1 \Join \sigma_c(G_2) \]
Join Operators, Frameworks & Algorithms

**Polyglot Data Management**

**Limited store capabilities:**
- Store is not able to perform joins
- Store is not able to provide all records
- Store returns records at irregular frequency

**Live Data Streams**
- Ajoin (2021)
- FastJoin (2019)
- ScaleJoin (2016)
- BiStream (2015)
- ...

**Network & Data Skews**
- FastJoin (2019)
- SharesSkew (2018)
- SquirrelJoin (2017)
- Flow-Join (2016)
- ...

**Distributed, Parallel, & GPUs**
- GPU-NL Join (2021)
- SquirrelJoin (2017)
- Flow-Join (2016)
- Track Join (2014)
- ...

**Spatio-temporal**
- k-SDJoin (2020)
- HyMJ (2019)
- TL-Join (2019)
- ...

**Data Streams**
- Data Streams

**Network & Data Skews**
- Network & Data Skews

**Distributed, Parallel, & GPUs**
- Distributed, Parallel, & GPUs

**Spatio-temporal**
- Spatio-temporal
Bind Join (Fetch-Match Strategy)

- **Scenario:**
  - Stores are not able to perform joins
  - Mediator cannot ship data between stores
  - \(|\text{Subquery1}| \ll |\text{Subquery2}|\)

- Without IN-subquery support: One query per join value \(x \in A\)

\[
\text{SELECT * FROM S2 WHERE Y = x;}
\]

- Fast access per index
- Used in CloudMdsQL and ESTOCADA
Double Pipelined Symmetric Hash Join

- Using two hash tables
  - insert new record in own hash
  - probe with other hash for potential join partners
- Symmetry reduces risks of blocking
- Pipelining (records are pushed immediately to next operator)
- **XJoin**: Spills partitions to disk
  - Enables larger data sets
  - Allows more parallelism
  - Reduces down times

---

SELECT * FROM D, H
WHERE D.A = H.Y AND D.B = 5 AND H.Z = 4;

- Join between
  - (distributed) RDBMS D
  - HDFS H

- Assumptions:
  - |H| ≫ |D|
  - Local predicates not selective
  - Hash $h$ for repartitioning

1. Local selection in D
2. Send bloom filter $BF_{D^*}$
3. Local & Join selection in H
4. Send bloom filter $BF_{H^*}$
5. Shuffle $H^*$ using $h$

ZigZag Join (JEN)

SELECT * FROM D, H
WHERE D.A = H.Y AND D.B = 5 AND H.Z = 4;

- Join between
  - (distributed) RDBMS D
  - HDFS H
- Assumptions:
  - $|H| \gg |D|$  
  - Local predicates not selective
  - Hash $h$ for repartitioning

6. Join selection in D
7. Distribute $D^{**}$ to HDFS partitions using $h$
8. Join $D^{**}$ and $H^*$
   (+ group by & aggregate)
9. Send join result

Cross-Platform Query Planning

**Planning:**
- Map operators to stores
- NP-hard for discrete sets
- Add migrations of data or operators

**Single Application Query** (e.g., Use-Case)

**Operator in Query**
- Filter
- Map
- Aggregation
- Grouping
- Join
- Path Matches
- ...

**Constraints:**
- Service Level Objectives
- Find efficient Migrations in platform network w.r.t. computing resources

**Placement Planning**

**IBM DB2**
- Billing Data
- Nested Application Data
- Search Index
- Session data
- Recommendation Engine

**Amazon Elastic MapReduce**
- Friend network
- Cached data & metrics

**Google Cloud Storage**

**Neuo4j the graph database**

**redis**

**mongoDB**

**Elasticsearch**

**Cassandra**

Polyglot Data Management: State of the Art & Open Challenges
Cross-Platform Query Planning

Tasks:
- Operator plan
- Select platforms with mappings for operations
- Push-down operations and link platforms with migrations

Query Interface
- global schema

Mediator

Mapping
- O₁, ..., Oₙ
- S₁, ..., Sₙ

Capabilities
- Resources
- Migration

Model Constrains
- Constrain

Parallelize
- Cardinality
- Latency

Join and Merge

Operator plan
- Select platforms with mappings for operations
- Push-down operations and link platforms with migrations

S₁
- JSON

S₂
- SQL

S₃

...
Shapes – Sequential

Network:
- Loading complete data set or
- Use streaming (distributed stream support) – Volcano model

Hint:
- Layered topology placement
- Single platform support of (special-filter) operations
- Forward result to next store
Hint:
- Replicated topology placement
- Decomposition for multiple sources (S1 & S2)
- Parallel execution
- Decomposition under actual system deployment (same node, but different stores)

Network:
- Symmetric vs. asymmetric join placement
Shapes – Diamond

Hint:
- Diamond topology
- Same as hierarchy-shape
- Partitioning of single data sets (e.g., reducing-workload-partitioning)
Operator Placement – Approaches

- **Model-based**: different strategies for placement solution
  - Hierarchical Placement
  - Pruned Space Placement
  - Relax-Expand-Solve

- **Model-free**: provide direct placement seek
  - Greedy First-match
  - Local optimization on greedy-first
  - Tabu search

- **ML-based**:
  - Explore placement decision for similar workloads
  - Learn latency of operator mappings
  - Learn cardinalities of topologies (JOP)
Operator Placement – Model-free Tactics

1. Resolve dependencies between platforms and operators

2. Fixed operators as initial placement and greedy expansion along logical plan

3. Co-locate operators on same platforms

4. Move single operator to another location to reduce estimated cost and latency

5. Switch platform by adding migration between source and target

6. Enumerate multiple plans (repeat step 3., 4. and 5. until threshold)

- Local optima problem
- May split co-location

- Terminate, when no further improvement

- Neighbour lookup
Current Systems
### Overview: Polyglot Data Management Systems

<table>
<thead>
<tr>
<th></th>
<th>Multistore</th>
<th>Polystore</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Loosely coupled</strong></td>
<td>PolyBase</td>
<td>Myria</td>
</tr>
<tr>
<td></td>
<td>BigIntegrator</td>
<td></td>
</tr>
<tr>
<td></td>
<td>FORWARD</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Apache Drill (Calcite)</td>
<td></td>
</tr>
<tr>
<td></td>
<td>QoX</td>
<td></td>
</tr>
<tr>
<td></td>
<td>QUEPA</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Odyssey</td>
<td></td>
</tr>
<tr>
<td><strong>Tightly coupled</strong></td>
<td>RHEEM</td>
<td>ESTOCADA</td>
</tr>
<tr>
<td></td>
<td>MuSQLE</td>
<td>Polypheny-DB</td>
</tr>
<tr>
<td></td>
<td>HadoopDB</td>
<td></td>
</tr>
<tr>
<td><strong>Hybrid</strong></td>
<td>CloudMdsQL</td>
<td>BigDAWG</td>
</tr>
<tr>
<td></td>
<td>SparkSQL</td>
<td></td>
</tr>
</tbody>
</table>
PolyBase

- Virtual data integration solution from Microsoft
- Distributed compute engine integrated with MS SQL Server
- Query data where it lives (T-SQL):
  - Oracle
  - MongoDB
  - Teradata
  - Hadoop-Cluster
  - Cosmos-DB
  - S3-compatible Store
  - SAP HANA

PolyBase – Query Concept

- Manual schema definition by Admin
- Create external data source in T-SQL (e.g., MongoDB)
  - Global schema in MS SQL
  - Definition of relational view on source such as MongoDB collection
  - User-defined statistics for source
  - MS SQL applies flattening rules on hierarchial source models
- Bridge the heterogeneity of models

```sql
CREATE EXTERNAL DATA SOURCE external_data_source_name
WITH (LOCATION = '<mongodb://<server>[:<port>]>'
[ [ , ] CREDENTIAL = <credential_name> ]
[ [ , ] CONNECTION_OPTIONS = '<key_value_pairs>'][,...]
[ [ , ] PUSHDOWN = { ON | OFF } ])
[ ; ]
```

```sql
CREATE EXTERNAL TABLE [MongoDbRandomData](
    [id] NVARCHAR(24) COLLATE SQL_Latin1_General_CP1_CI_AS NOT NULL,
    [RandomData_friends_id] INT,
    [RandomData_tags] NVARCHAR(MAX) COLLATE SQL_Latin1_General_CP1_CI_AS)
WITH (LOCATION='MyDb.RandomData',
      DATA_SOURCE=[MongoDb])
```

Example from: PolyBase Extension Group Model: https://docs.microsoft.com/de-de/sql/relational-databases/polybase/polybase-scale-out-groups?view=sql-server-ver16,
Accessed: June 2022
PolyBase – Optimization Model

- **Distributed query execution across SQL Servers**
  - Read external partitioning metadata
  - Split MS SQL source and remote source
  - Push-down operations where possible

- **Plugin architecture for SQL-Server**
  - Mapping of T-SQL to stores
  - Scale-out compute node
  - PolyBase waits for source data to be processed

### Overview: Polyglot Data Management Systems

<table>
<thead>
<tr>
<th></th>
<th>Multistore</th>
<th>Polystore</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Loosely coupled</strong></td>
<td>PolyBase</td>
<td>Myria</td>
</tr>
<tr>
<td></td>
<td>BigIntegrator</td>
<td></td>
</tr>
<tr>
<td></td>
<td>FORWARD</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Apache Drill (Calcite)</td>
<td></td>
</tr>
<tr>
<td></td>
<td>QoX</td>
<td></td>
</tr>
<tr>
<td></td>
<td>QUEPA</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Odyssey</td>
<td></td>
</tr>
<tr>
<td><strong>Tightly coupled</strong></td>
<td>RHEEM</td>
<td>ESTOCADA</td>
</tr>
<tr>
<td></td>
<td>MuSQLE</td>
<td></td>
</tr>
<tr>
<td></td>
<td>HadoopDB</td>
<td></td>
</tr>
<tr>
<td><strong>Hybrid</strong></td>
<td>CloudMdsQL</td>
<td>BigDAWG</td>
</tr>
<tr>
<td></td>
<td>SparkSQL</td>
<td></td>
</tr>
</tbody>
</table>
RHEEM – Plan Enumeration

**Input: Directed RHEEM dataflow plan**
- RheemLatin DSL
- RheemStudio
- Java, Scala, Python
- REST Endpoint

**Output: Inflated operator plan with migration steps between platforms**
- Map fix RHEEM operator to execution platform
- Apply mappings between single logical operators to n* execution operators
- Resolve minimum conversion tree to transfer data between multiple platforms

RHEEM

- Decoupling application task from (multi-platform) execution
- Mapping of platform-agnostic operator to platform-specific operators using LAV
- Resolve Migrations using Channel Conversion Graph
- Supports
  - InMemory (Java), GraphChi
  - PostgreSQL
  - Flink, Spark
- Developed as Apache Wayang (Incubating)

RHEEM – Plan Optimization

- **First version:** *genetic* cost-model learner, loss reduction
  - operator execution costs
  - Samples cardinalities and reduce size estimation function e.g., Filter: \( \text{card (Filter)} = c_{in}(\text{Filter}) \cdot \sigma_f \) for selectivity \( f \)

- **ML version (Robopt):** *supervised* fine-level cost-tuning
  - Encodes *logical operator-, platforms and movements* into vectors
  - Vectorized execution plan
  - ML-model selects enumerated vector plans with platform-agnostic operations
  - **Optimizes** the order of executing RHEEM operators

---


Kaoudi et al., ML-based cross-platform query optimization, ICDE, 2020.
## Overview: Polyglot Data Management Systems

<table>
<thead>
<tr>
<th>Coupling Style</th>
<th>Multistore</th>
<th>Polystore</th>
</tr>
</thead>
<tbody>
<tr>
<td>Loose</td>
<td>PolyBase</td>
<td>Myria</td>
</tr>
<tr>
<td></td>
<td>BigIntegrator</td>
<td></td>
</tr>
<tr>
<td></td>
<td>FORWARD</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Apache Drill (Calcite)</td>
<td></td>
</tr>
<tr>
<td></td>
<td>QoX</td>
<td></td>
</tr>
<tr>
<td></td>
<td>QUEPA</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Odyssey</td>
<td></td>
</tr>
<tr>
<td>Tightly</td>
<td>RHEEM</td>
<td>ESTOCADA</td>
</tr>
<tr>
<td></td>
<td>MuSQL</td>
<td>Polypheny-DB</td>
</tr>
<tr>
<td></td>
<td>HadoopDB</td>
<td></td>
</tr>
<tr>
<td>Hybrid</td>
<td>CloudMdsQL</td>
<td>BigDAWG</td>
</tr>
<tr>
<td></td>
<td>SparkSQL</td>
<td></td>
</tr>
</tbody>
</table>
BigDAWG – Overview and Architecture

- Developed at:
  - At: Intel Science and Technology Center for Big Data (MIT)
  - Between: 2015 and 2019

- Use Cases:
  - Medical applications (MIMIC II)
  - Ocean Metagenomic Analysis

Architecture figure: Gadepally et al., The BigDAWG polystore system and architecture, IEEE HPEC, 2016.
BigDAWG – Islands of Information

- 3 components of virtual islands:
  - Data model
  - Query language
  - Storage engines

- Degenerate islands to achieve semantic completeness

- **Shims**: semantical mapping between island and data store

- **Casts and Scope**: accessing multiple islands

- Extensible by implementing new islands

BigDAWG – Performance Profiling

- **Training mode:**
  - all plans of a query are executed
  - the best is stored in the preference matrix

- **Optimized mode:**
  - either the best plan from the preference matrix
  - or a random plan is executed

- **Opportunistic mode:**
  - Similar to optimized mode
  - Additional evaluations during times of low system utilization
  - Additional evaluations if new stores become available
BigDAWG – Semantic Equivalence

- „Semantically equivalent queries [...] are substitutable“
- Encode intersecting sets of semantic capabilities using a **semantic lattice**
- Capture semantic equivalent (sub-)queries in a semantic dictionary (Equivalence Rule)
- 3 types of semantic containment:
  - Order of result entries
  - Expressivness of semantics
  - Backward compatibility for primitive types

---

## Overview: Polyglot Data Management Systems

<table>
<thead>
<tr>
<th></th>
<th>Multistore</th>
<th>Polystore</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Loosely coupled</strong></td>
<td>PolyBase</td>
<td>Myria</td>
</tr>
<tr>
<td></td>
<td>BigIntegrator</td>
<td></td>
</tr>
<tr>
<td></td>
<td>FORWARD</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Apache Drill (Calcite)</td>
<td></td>
</tr>
<tr>
<td></td>
<td>QoX</td>
<td></td>
</tr>
<tr>
<td></td>
<td>QUEPA</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Odyssey</td>
<td></td>
</tr>
<tr>
<td><strong>Tightly coupled</strong></td>
<td>RHEEM</td>
<td>ESTOCADA</td>
</tr>
<tr>
<td></td>
<td>MuSQLE</td>
<td>Polypheny-DB</td>
</tr>
<tr>
<td></td>
<td>HadoopDB</td>
<td></td>
</tr>
<tr>
<td><strong>Hybrid</strong></td>
<td>CloudMdsQL</td>
<td>BigDAWG</td>
</tr>
<tr>
<td></td>
<td>SparkSQL</td>
<td></td>
</tr>
</tbody>
</table>
ESTOCADA

- Developed by University of San Diego and INRIA*

- Focus on view-based Query Rewriting (Local-as-view)

- Leveraging possible data redundancy and previously computed query results for improving performance

- Can be built into existing Polystores (e.g., BigDAWG, SparkSQL, Tatooine)

- Functional demonstration based on MIMIC III dataset

* Institut national de recherche en sciences et technologies du numérique

Architecture figure: Bugiotti et al., Invisible Glue: Scalable Self-Tuning Multi-Stores, CIDR, 2015.
ESTOCADA – Virtual Views

- Relational model as pivot model
- Virtual views on underlying models (encoded *relationally*)
- Differences in semantics modeled by integrity constraints
  - tuple-generating dependencies
  - equality-generating dependencies
- Encodings/Models hidden (only necessary for query rewriting)

Example: Alotaibi et al., Towards Scalable Hybrid Stores: Constraint-Based Rewriting to the Rescue, SIGMOD, 2019.
ESTOCADA – Query Language and Rewriting

QBT$^\text{XM}$:

- Block-based integration language
- Each block contains native query language
  - FOR clause: Bind variables from stores
  - WHERE clause: Selections on bound variables
  - RETURN clause: Construct new data based on variable bindings

Query Rewriting:

- Optimized version of PACB algorithm
- Query rewriting using all virtual as well as materialized views

Logical Query Plan:

- Translation of PACB result into logical plan
  - Subqueries and supported operators pushed down to stores
  - Handling of unsupported operators and cross-store-joins by the integration layer

## Overview: Polyglot Data Management Systems

<table>
<thead>
<tr>
<th>Coupling Type</th>
<th>Multistore</th>
<th>Polystore</th>
</tr>
</thead>
<tbody>
<tr>
<td>松散耦合</td>
<td>PolyBase</td>
<td>Myria</td>
</tr>
<tr>
<td></td>
<td>BigIntegrator</td>
<td></td>
</tr>
<tr>
<td></td>
<td>FORWARD</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Apache Drill (Calcite)</td>
<td></td>
</tr>
<tr>
<td></td>
<td>QoX</td>
<td></td>
</tr>
<tr>
<td></td>
<td>QUEPA</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Odyssey</td>
<td></td>
</tr>
<tr>
<td>紧耦合</td>
<td>RHEEM</td>
<td>ESTOCADA</td>
</tr>
<tr>
<td></td>
<td>MuSQLE</td>
<td></td>
</tr>
<tr>
<td></td>
<td>HadoopDB</td>
<td></td>
</tr>
<tr>
<td>混合</td>
<td>CloudMdsQL</td>
<td>BigDAWG</td>
</tr>
<tr>
<td></td>
<td>SparkSQL</td>
<td></td>
</tr>
</tbody>
</table>
CloudMdsQL

- Functional SQL-like language implemented in LeanXcale (Research system)

- Multistore with (current) support for
  - PostgresQL
  - Apache Spark
  - MongoDB
    - (Python)

- Abstraction layer for data retrieval
  - Preserves the semantics of the underlying data stores
  - A query may contain embedded (native) subqueries
  - Python functions to query API-only query interfaces

- Mediator/wrapper architecture

- Relational model as internal data model

---

CloudMdsQL – Query Execution

- Queries usually consist of subqueries and an integration SELECT statement

  ```
  T1(x int, y int)@rdb = ( SELECT x, y FROM A )
  T2(x int, z array)@mongo = {*
    db.B.find( {$lt: {x, 10}}, {x:1, z:1, _id:0} )
  *}
  SELECT T1.x, T2.z
  FROM T1, T2
  WHERE T1.x = T2.x
  ```

- The system creates query execution plans (QEPs)
  - Subqueries are pushed down to the wrappers/stores
  - Subquery results are transformed into a relational format
  - Relational data is combined using Bind Joins

CloudMdsQL – Query Optimization

- The optimization search space for consists of all query rewritings by
  - Pushing down select operations
  - Expressing Bind Joins
  - Join ordering

- Search space is small, thus a simple exhaustive search strategy is used

- Usage of a simple catalog for comparing rewritten queries:
  - Data collection cardinalities
  - Indexes
  - Attribute selectivities
  - Simple cost models

- Local cost models provided by probing and sampling by the wrappers
### Wrap Up: Polyglot Data Management Systems

<table>
<thead>
<tr>
<th>Polyglost</th>
<th>Data virtualization, single T-SQL interface, data fabric (Microsoft)</th>
</tr>
</thead>
<tbody>
<tr>
<td>PolyBase</td>
<td>Automatic cross-platform optimization, operator placement &amp; data migration</td>
</tr>
<tr>
<td>Multistore</td>
<td>Virtual views, constrained based data transformation &amp; block-based integration language</td>
</tr>
<tr>
<td>Multistore</td>
<td>Application-driven, customizable data islands &amp; semantic equivalence</td>
</tr>
<tr>
<td>PolyBase</td>
<td>Odyssey</td>
</tr>
<tr>
<td>BigIntegrator</td>
<td>RHEEM</td>
</tr>
<tr>
<td>Odyssey</td>
<td>ESTOCADAADA</td>
</tr>
<tr>
<td>RHEEM</td>
<td>CloudMdsQL</td>
</tr>
<tr>
<td>CloudMdsQL</td>
<td>BigDAWG</td>
</tr>
<tr>
<td>CloudMdsQL</td>
<td>Single Interface Multistore</td>
</tr>
<tr>
<td>SparkSQL</td>
<td>Single Interface Multistore</td>
</tr>
<tr>
<td>Hybrid</td>
<td>Single Interface Multistore</td>
</tr>
</tbody>
</table>

**Polyglot Data Management: State of the Art & Open Challenges**
Open Challenges
Open Challenges: Overview

- **Unified Access vs. Unique Features**
  How to design a suitable interface?

- **Ad Hoc Data Manipulation**
  How to push user updates to the stores?

- **Adaptive Reconfiguration**
  How to react to changing requirements?

- **Cross-System Query Optimization**
  How to find the optimal query plan?

- **Streaming & Real-Time Readiness**
  How to integrate real time requirements?

- **Multi-Model Schema Management**
  How to update schema mappings?
Open Challenges: (i) Unified Access vs. Unique Features

„smallest common denominator“

✓ Simple to build
✔ Not very powerful
✔ Loss of semantic/functional features

Mediator query language with embedded store query languages

✓ Easily extensible
✓ Full semantic/functional complexity
✔Does not hide complexity
✔Prevents intra store optimization potential

All-powerful query language

✓ Hidden complexity
✓ Full semantic/functional complexity
✔ Super complex to build
✔ Extensibility challenging
✔ Feasible?

FROM readings AS r
GROUP BY r.gas AS g
SELECT ELEMENT {
  gas: g,
  count: count(group),
  avg: avg( *
    FROM group AS p
    SELECT ELEMENT p.r.num)
}

SQL++

@eq{
  complex : yes,
  null_eq_null : null,
  null_and_true : null
} (r=r)

CloudMdsQL

T1(x int, y int)@rdb = ( SELECT x, y FROM A )
T2(x int, z array)@mongo = {*
  db.B.find( {$lt: {x, 10}}, {x:1, z:1, _id:0} ) *
}
SELECT T1.x, T2.z
FROM T1, T2
WHERE T1.x = T2.x

Ong et al., The SQL++ Semi-structured Data Model and Query Language. arxiv.org, 2014.
Open Challenges: (ii) Ad Hoc Data Manipulation

**Two round approach**
1. Determine all records to be updated
2. Update each record based on its ID

Is there any efficient one round approach?
Open Challenges: (ii) Ad Hoc Data Manipulation

UPDATE T1
SET A = A * 1.1 + B
WHERE C = „WI“;

Mediator

S1
ID
1 11
2 9.5
3 8.5
...

S2
ID
1 11
2 4
3 6
...

S3
ID
1 RS
2 WI
3 WI
...

Further Challenges:
How to ensure cross-store
• **Atomicity, Isolation & Durability**
  • logging
  • locking
  • recovery
• **Consistency**
  • check constraints
  • referential integrity

if individual stores do not support such mechanisms?
Open Challenges: (iii) Adaptive Reconfiguration

- Detecting changing requirements or workloads?
  - Fluctuating traffic throughout the day
  - Singular events (e.g. Black Friday)
  - Additional users in a multi-tenant environment

- Adapting/reconfiguring the system
  - Adding or removing resources
  - Reorganization (e.g. splitting a hot range)

- Changing the system topology
  - Data migration between stores
    (e.g. write-heavy data to main-memory database)

at runtime
Open Challenges: (iv) Cross-System Query Optimization

- Operator Placement
  - Data vs. Operator Shipping
  - Migration Paths

- Pareto Optimum of Objectives
  - Latency
  - Throughput
  - Planning
  - Application Objective

- ML-based optimization
  - Hard constraint for query correctness in optimization
  - Join-Ordering for sub-query
Open Challenges: (v) Streaming & Real-Time Readiness

- Streaming Workloads
  - Expose streaming capabilities
  - Integrate streaming with storage systems

- Push-based features
  - Triggers, ECA rules
  - Change notifications

- Caching
  - Materialized views
  - Cache coherence / cache invalidation
Open Challenges: (vi) Multi-Model Schema Management

- Mappings between global & local Schemas
  - fundamental for query rewriting
  - cross-model (e.g., SQL ↔ graph)
  - via wrapper

- Update of Mappings
  - Evolution of global schema
  - Evolution of local schema
  - Migration of data between stores

- Composition/Extraction of Mappings for data migration
Towards Polyglot Data Stores
Overview and Open Research Questions

DANIEL GLAKE*, FELIX KIEHN*, and MAREIKE SCHMIDT*, Universität Hamburg
FABIAN PANSE and NORBERT RITTER, Universität Hamburg

Nowadays, data-intensive applications face the problem of handling heterogeneous data with sometimes mutually exclusive use cases and soft non-functional goals such as consistency and availability. Since no single platform copes everything, various stores (RDBMS, NewSQL, NoSQL) for different workflows and use cases have been developed. However, since each store is only a specialization, this motivates progress in polyglot data management emerging new systems called Multi- and Polystores. They are trying to access different stores transparently and combine their capabilities to achieve one or multiple given use cases. This paper describes representative real-world use cases for data-intensive applications (OLTP and OLAF). It derives a set of requirements for polyglot data stores. Subsequently, we discuss the properties of selected Multi- and Polystores and evaluate them based on given needs illustrated by three common application use cases. We classify them into functional features, query processing technique, architecture and adaptivity and reveal a lack of capabilities, especially in changing conditions tightly integration. Finally, we outline the benefits and drawbacks of the surveyed systems and propose future research directions and current challenges in this area.

CCS Concepts: - Information systems → DB engine architectures.

Additional Key Words and Phrases: polyglot persistence, multi-/polystore, data management, adaptivity, query processing.

Further Readings

Thanks …
Slides available at: vldb2022.dbis.hamburg