NoSQL Schema Evolution and Big Data Migration at Scale

Uta Störl
Darmstadt University of Applied Sciences, Germany

April 2018

Joint work with Meike Klettke (University of Rostock, Germany) and Stefanie Scherzinger (OTH Regensburg, Germany)
Motivation

• Agile software development with frequently schema changes (weekly up to daily!) ⇒ Schema-flexible NoSQL databases 😊

• However, how to migrate variational data in the productive database?
  – State of the art: Within the application code 😞
  – Schema management for NoSQL database systems necessary!
Schema Management for NoSQL Databases

- (Optional) schema management for NoSQL databases

> Two main tasks
  - Schema evolution management
  - Data migration as safe process based on schema evolution management
Vision: Curating Variational Data End-to-End

Uta Störl, Darmstadt University of Applied Sciences
NoSQL Schema Evolution Language

• Schema evolution language for describing structural changes of the data

• **Single type** operations:
  – *add* property
  – *rename* property
  – *delete* property

• **Multi-type** operations:
  – *copy* property (denormalization)
  – *move* property (refactoring)

Introducing in: S. Scherzinger, M. Klettke, U. Störl: Managing Schema Evolution in NoSQL Data Store, DBPL@VLDB, Italy, 2013
Our Approach: Supporting Schema Management and Data Migration with Darwin

**Darwin** supports the complete schema management life cycle:

- declaring or extracting an initial schema,
- repeatedly applying schema changes,
- proposing mappings between legacy schema versions,
- and migrating legacy data.

The system is implemented for different types of NoSQL database systems.

Vision: Curating Variational Data End-to-End

Top Down Approach
explicitly declared by developer

Bottom Up Approach
obtained by schema extraction

Data Migration
Schema Extraction Approach

- Process of extracting schema versions and evolution operations by reverse engineering

Running Example from Marine Biology

- JSON datasets for *Species* classification of the Baltic Sea and observation *Protocols*

WoRMS: World Register of Marine Species
Step 1: Building the Schema Version Graphs

**Species**
- **id**: [1,3,5,7] (type: number)
- **name**: [1,3,5,7] (type: string)
- **ts**: [1,3,5,7] (type: number)
- **category**: [3] (type: number)
- **WoRMS**: [5,7] (type: number)

**Protocols**
- **id**: [2,4,6,8] (type: number)
- **time**: [2,4,6,8] (type: number)
- **location**: [2,4,6,8] (type: object)
- **spec_id**: [2,4,6,8] (type: number)
- **ts**: [2,4,6,8] (type: number)
- **WoRMS**: [8] (type: number)

**Species**
- **id**: 123, *name*: "Mya arenaria", *ts*: 3, *category*: 141436

**Protocols**
Step 2: Deriving Schema Evolution Operations

**Species**
- **id**: [1,3,5,7] (type: number)
- **name**: [1,3,5,7] (type: string)
- **ts**: [1,3,5,7] (type: number)
- **category**: [3] (type: number)
- **WoRMS**: [5,7] (type: number)

**Protocols**
- **id**: [2,4,6,8] (type: number)
- **time**: [2,4,6,8] (type: number)
- **location**: [2,4,6,8] (type: object)
- **spec_id**: [2,4,6,8] (type: number)
- **ts**: [2,4,6,8] (type: number)
- **WoRMS**: [8] (type: number)

2. **add** integer Species.category
3. **rename** Species.category to WoRMS
   - or
   - **delete** Species.category
4. **add** integer Protocols.WoRMS
   - or
   - **copy** Species.WoRMS to Protocols.WoRMS

where Species.id = Protocols.spec_id
Step 3: Resolving Ambiguities

- Interactively resolving ambiguous schema evolution operations:
  - alternative schema evolution operations
  - specifying join conditions for move or copy operations
Resolving Ambiguities

• **Open questions**
  – Automated choice in case of ambiguities
  – Suggestion of meaningful join conditions

• **Solution approaches**
  – Algorithm for deriving inclusion dependencies from NoSQL datasets proposed in
  – Next steps:
    • Choice of a meaningful join condition for copy and move operations
    • Heuristics for search space reduction
Deriving Schema Versions

```
{ "title": "Protocols",  
  "description": "schema version 3",  
  "type": "object",  
  "properties": {    
    "id": { "type": "number" },    
    "location": { "type": "object",    
      "properties": { ... } },    
    "spec_id": { "type": "number" } } }
```

```
{ "title": "Protocols",  
  "description": "schema version 4",  
  "type": "object",  
  "properties": {    
    "id": { "type": "number" },    
    "location": { "type": "object",    
      "properties": { ... } },    
    "spec_id": { "type": "number" },    
    "WoRMS": { "type": "number" } } }
```

copy Species.WoRMS to Protocols.WoRMS
where Species.id = Protocols.spec_id
Extracted Schema Evolution Operations and Schema Versions

Step 1: incremental structure extraction
Step 2: deriving candidates for schema evolution operations
Step 3: resolving ambiguities

List of schema versions and schema evolution operations

Uta Störl, Darmstadt University of Applied Sciences
Vision: Curating Variational Data End-to-End

Application Version n → Application Code Version n + 1

Top Down Approach
explicitly declared by developer

Schema Version n → Schema Version n+1

Schema Evolution Operations

Bottom Up Approach
obtained by schema extraction

Data Migration

Uta Störl, Darmstadt University of Applied Sciences
Data Migration – Dimensions

- **Time**
- **Operation Execution**
- **Location**
- **Amount**

**Operation Execution**
- composite
- stepwise
- application
- database update
- stored procedure
- complete
- partial
- minimal
- eager
- predictive
- incremental
- lazy

**Location and Amount**

Basic Strategies of Data Migration

Eager Migration
- after introduction of a new schema version, all entities are migrated

Advantages:
- all entities are in the current version
- low latency (if entities are accessed)

Disadvantages:
- even entities which are not in use are migrated
- high number (and costs) of migration operations

Lazy Migration
- evolution operations are stored,
- data migration is done on request

Advantages:
- only entities which are in use are migrated
- no unnecessary data migration operations
- composition of operations is possible

Disadvantages:
- entities in the NoSQL database in different versions
- increased latency
How to reduce Costs of Data Migration?

• In case of **large amount of datasets**
  – (system downtime)
  – update operations even for cold data (which are not in use)

• In case of **database as a service**
  – payable operations, monetary costs for all data migrations

• **How to reduce costs of data migration?**
  - Optimize Data Migration
  - Hybrid Migration Approaches
    • Predictive Migration
    • Incremental Migration
Hybrid Migration Strategies

Predictive Migration
- Forecast function, which entities are accessed in near future (bases on heuristics)
- Predictive migration of these entities

Advantages:
+ decreased average latency
+ reduced number of migration operations

Disadvantage:
- additional migration operations in case of wrong predictions

Incremental Migration
- in some version, an eager migration is applied

Advantage:
+ composition of operations is possible

Disadvantage:
- even entities which are not in use are migrated
# Data Migration: Comparison of different Approaches

<table>
<thead>
<tr>
<th>Eager Migration</th>
<th>Predictive Migration</th>
<th>Incremental Migration</th>
<th>Lazy Migration</th>
<th>Versioning*</th>
</tr>
</thead>
<tbody>
<tr>
<td>high</td>
<td>medium</td>
<td>medium</td>
<td>low</td>
<td>none</td>
</tr>
<tr>
<td>none</td>
<td>low</td>
<td>low</td>
<td>medium</td>
<td>high</td>
</tr>
<tr>
<td>high</td>
<td>medium</td>
<td>medium</td>
<td>none</td>
<td>none</td>
</tr>
<tr>
<td>none</td>
<td>medium</td>
<td>medium</td>
<td>medium</td>
<td>high</td>
</tr>
</tbody>
</table>

*Versioning*: Does not migrate data – use query rewriting instead

- **Number of migration operations**
- **Average latency**
- **Application downtime**
- **Effort for query rewriting**

Uta Störl, Darmstadt University of Applied Sciences
Data Migration – Dimensions

- **Time**
- **Operation Execution**
- **Location**
- **Amount**

**Operation Execution**
- Composite
- Stepwise

**Time**
- Eager
- Predictive
- Incremental
- Lazy

**Location**
- Database Update
- Stored Procedure

**Amount**
- Complete
- Partial
- Minimal

Optimization of Data Migration: Composite Migration

**Theory:** If entities are migrated from older versions

⇒ **Composition of evolution operations** is possible in some cases:

<table>
<thead>
<tr>
<th>$op_2$</th>
<th>$op_1$</th>
<th>rename $E_{B,y}$ to $z$</th>
<th>copy $E_{B,y}$ to $E_{C,z}$</th>
<th>move $E_{B,y}$ to $E_{C,z}$</th>
<th>delete $E_{B,y}$</th>
</tr>
</thead>
<tbody>
<tr>
<td>add $E_{B,y} = \text{default}$</td>
<td>add $E_{B,z} = \text{default}$</td>
<td>-</td>
<td>add $E_{C,z} = \text{default}$</td>
<td>$\epsilon$ (noop)</td>
<td></td>
</tr>
<tr>
<td>rename $E_{B,x}$ to $y$</td>
<td>rename $E_{B,x}$ to $z$</td>
<td>copy $E_{B,x}$ to $E_{C,z}$</td>
<td>move $E_{B,x}$ to $E_{C,z}$</td>
<td>delete $E_{B,x}$</td>
<td></td>
</tr>
<tr>
<td>copy $E_{A,x}$ to $E_{B,y}$</td>
<td>copy $E_{A,x}$ to $E_{B,z}$</td>
<td>-</td>
<td>copy $E_{A,x}$ to $E_{C,z}$</td>
<td>$\epsilon$ (noop)</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>$\text{cond}_1 \land \text{cond}_2$</td>
<td>copy $E_{A,x}$ to $E_{B,y}$</td>
<td></td>
</tr>
<tr>
<td></td>
<td>copy $E_{B,y}$ to $E_{D,u}$</td>
<td>-</td>
<td></td>
<td>move $E_{B,y}$ to $E_{D,u}$</td>
<td></td>
</tr>
<tr>
<td></td>
<td>move $E_{A,x}$ to $E_{B,y}$</td>
<td>-</td>
<td>move $E_{A,x}$ to $E_{C,z}$</td>
<td>delete $E_{A,y}$</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>move $E_{A,x}$ to $E_{C,z}$</td>
<td>$\text{cond}_1 \land \text{cond}_2$</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>move $E_{A,x}$ to $E_{B,y}$</td>
<td>delete $E_{A,y}$</td>
<td></td>
</tr>
</tbody>
</table>

Example:

- copy $A.y$ to $B.y$
  - where $A.id = B.aid$
- delete $A.y$

⇒

- move $A.y$ to $B.y$
  - where $A.id = B.aid$

**Advantages**

⇒ Smaller number of data migration operations

⇒ Lower migration costs

Optimization of Data Migration: Composite Migration

Practice

First (surprising) results:

- Surprisingly, composite migration is not immediately faster.
- Closer analysis reveals that it is the overhead of computing the compositions repeatedly, which falsifies our hypothesis.

U. Störl, A. Tekleab, M. Klettke, S. Scherzinger. In for a Surprise when Migrating NoSQL Data. Lightning Talk@ICDE, Paris, April, 2018.
Optimization of Data Migration: Composite Migration

Practice

After introducing a cache for computed composition formulae:

- By introducing a cache and storing the computed composition formulae, we can effectively reduce the runtime when compared to the stepwise approach.

- We achieve a reduction by about 50% for MongoDB, and by nearly 80% for Cassandra.

U. Störl, A. Tekleab, M. Klettke, S. Scherzinger. In for a Surprise when Migrating NoSQL Data. Lightning Talk@ICDE, Paris, April, 2018.
NoSQL Schema Evolution and Big Data Migration at Scale

Conclusion

- **Objective**: Support for schema-management end-to-end including
  - **Schema evolution management**
    - Schema evolution language
    - Schema extraction
  - **Data migration** as safe process based on schema evolution management
    - Data migrations strategies may be classified and optimized by 4 dimensions
    - Lessons learned:
      - In architecting a scalable data migration tool for NoSQL data stores, careful engineering is called for.
      - In particular, a thorough experimental analysis is required.
NoSQL Schema Evolution and Big Data Migration at Scale

Future Work

• Investigate optimizations so that data migration can be implemented at scale
• Determine the major cost factors for the migration strategies to compile from them a generic cost model
• Craft a dedicated NoSQL migration benchmark
⇒ Design and develop an automated data migration advisor that supports software development teams in managing schema evolution in NoSQL data stores

Acknowledgement
This work is funded by the Deutsche Forschungsgemeinschaft (DFG, German Research Foundation), grant #385808805, since December 2017.
Backup
Supporting Schema Management and Data Migration with Darwin: Measurement Environment

• Big Data Cluster @ Darmstadt University of Applied Sciences
  – 40 Nodes
    • 64 GB / 128 GB RAM; 4 TB storage each
  – MongoDB / Couchbase / Cassandra / ...
  – Apache Hadoop
  – Apache Spark
  – Big Data Competence Centre: http://fbi.h-da.de/bdcc
NoSQL Schema Management and Data Migration: Selected Publications

- U. Störl, A. Tekleab, M. Klettke, S. Scherzinger. In for a Surprise when Migrating NoSQL Data. Lightning Talk @ 34th IEEE International Conference on Data Engineering (ICDE), Paris, April, 2018.
- U. Störl, T. Hauf, M. Klettke und S. Scherzinger: Schemaless NoSQL Data Stores – Object-NoSQL Mappers to the Rescue? BTW 2015, Hamburg, March 2015