Universität Rostock

Traditio et Innovatio

1218 HAFEN DER WISSENSCHAFTEN ROSTOCK 1419

Schema Evolution in Scalable Cloud Data Management (SCDM)

Meike Klettke — Universität Rostock

Uta Störl — Hochschule Darmstadt

Stefanie Scherzinger — OTH Regensburg

Motivation

NoSQL databases are often used in Big data applications

- Volume (Scalability)
- **Variety (Schema Flexibility)**

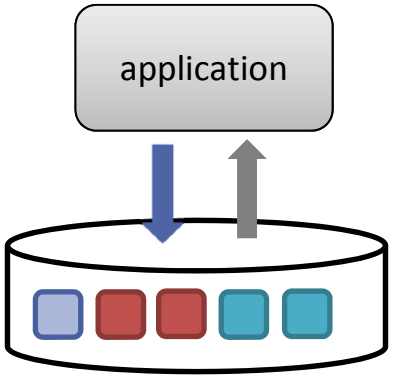
Main Reasons for using NoSQL:

- Scalability
- **Schema-flexibility**

Why Schema Flexibility?

Advantages and Disadvantages of Schema Flexibility

- **Advantages:**
  - Storage of heterogeneous data
  - Storage of different data versions (with structural changes over time)
- **Disadvantage:**
  - high effort to query and process the heterogonous data

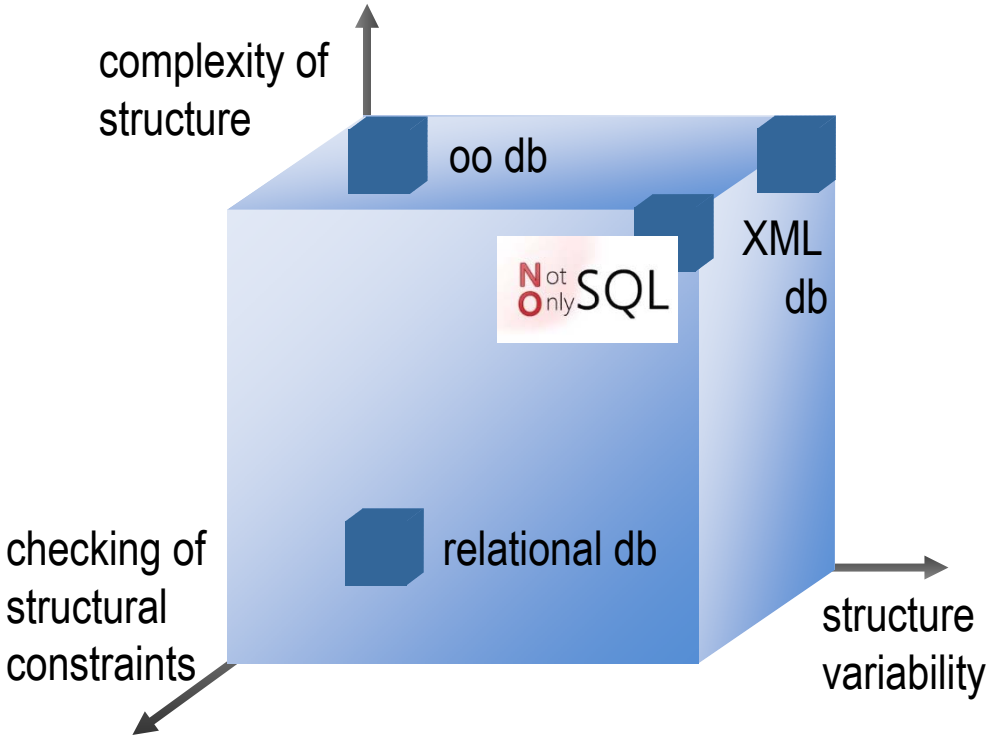


Why Schema Management?

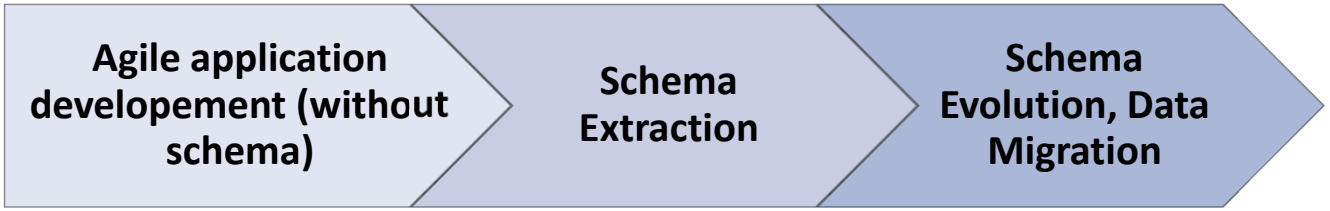
Schema Management

- ***Compromise*** between
  - strict predefined schema (like in relational databases) and
  - schema flexibility (like in NoSQL databases)

- can be realized
  - inside the NoSQL database or
  - on-top of the database system

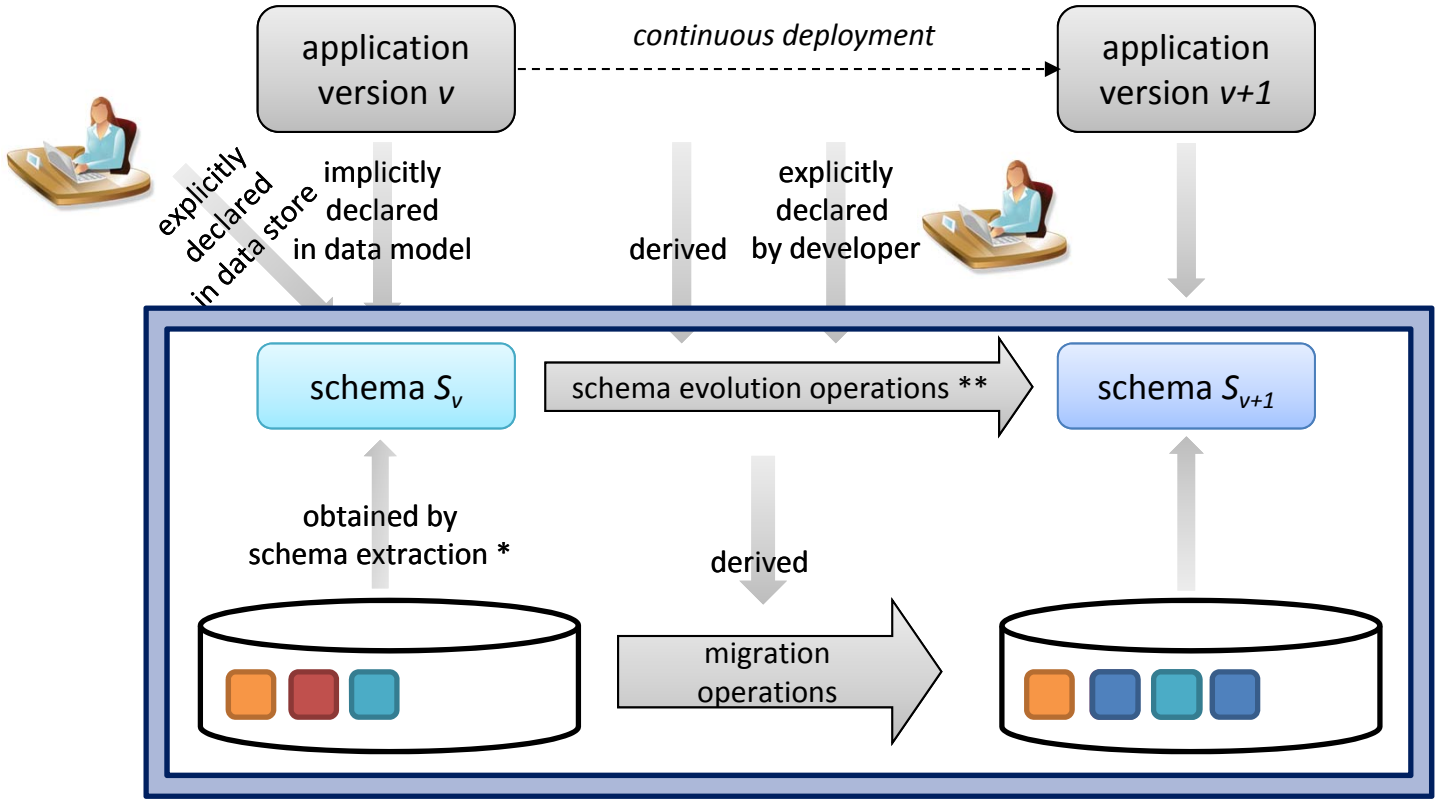


Schema Management Subtasks in Agile Development



1. Usage of NoSQL database for storing data of the agile application (data science)
2. **Extraction of an explicit schema** from available datasets
3. in the following: in case of structural changes: **schema evolution and (scalable) data migration**

Subtasks of Schema Management



* Meike Klettke, Uta Störl, Stefanie Scherzinger: Schema Extraction and Structural Outlier Detection for NoSQL Data Stores, BTW 2015

** Introduced in: Stefanie Scherzinger, Meike Klettke, Uta Störl: Managing Schema Evolution in NoSQL Data Store, DBPL@VLDB, 2013

How to realize Schema Extraction?

Heterogeneous JSON documents (blogposts)

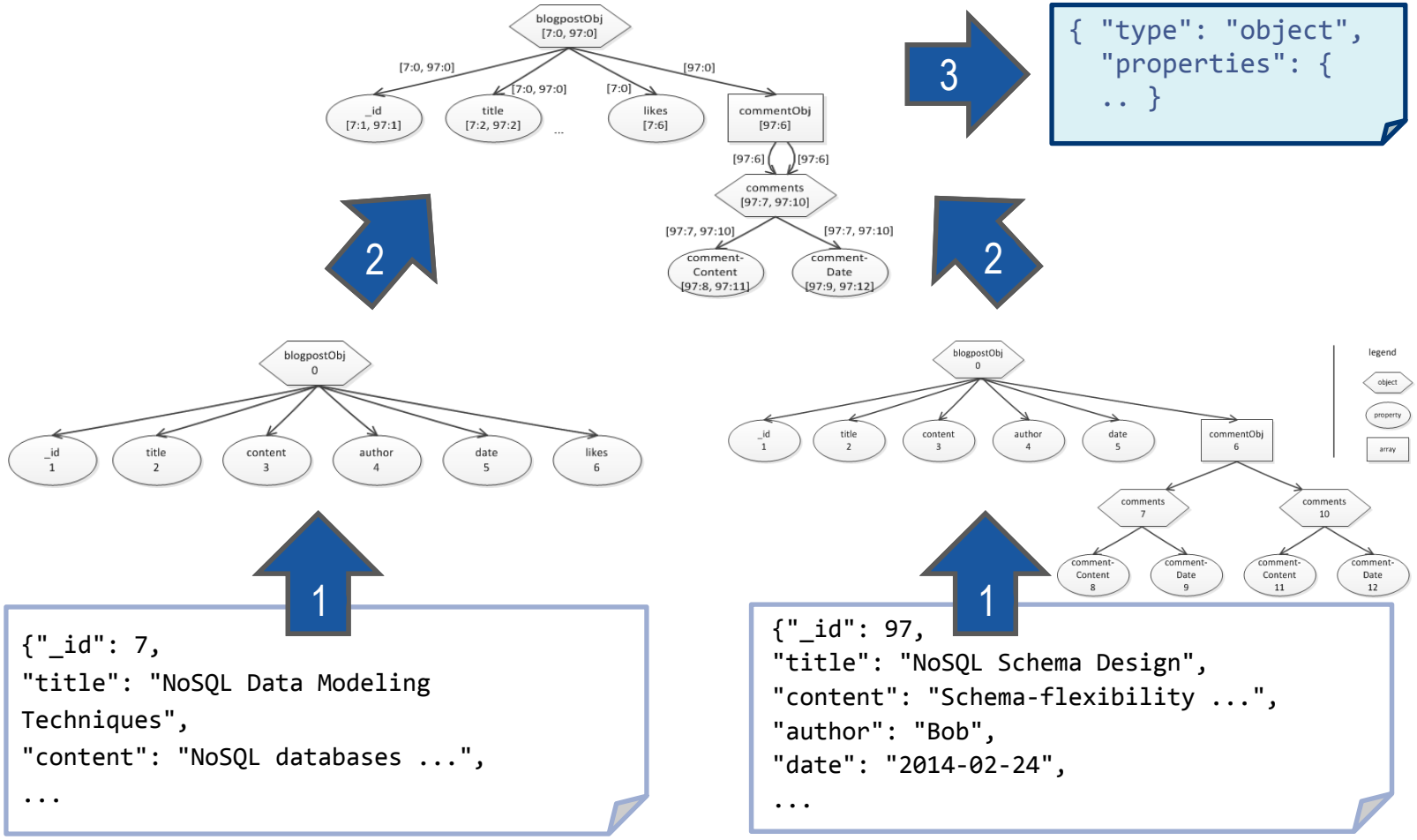
```
{"_id": 7,
"title": "NoSQL Data Modeling
Techniques",
"content": "NoSQL databases ...",
"author": "Alice",
"date": "2014-01-22",
"likes": 34 }
```

```
{"_id": 97,
"title": "NoSQL Schema Design",
"content": "Schema-flexibility ...",
"author": "Bob",
"date": "2014-02-24",
"comments": [
  { "commentContent": "Not sure...",
  "commentDate": "2014-02-24" },
  { "commentContent": "Let me ...",
  "commentDate": "2014-02-26" } ] }
```

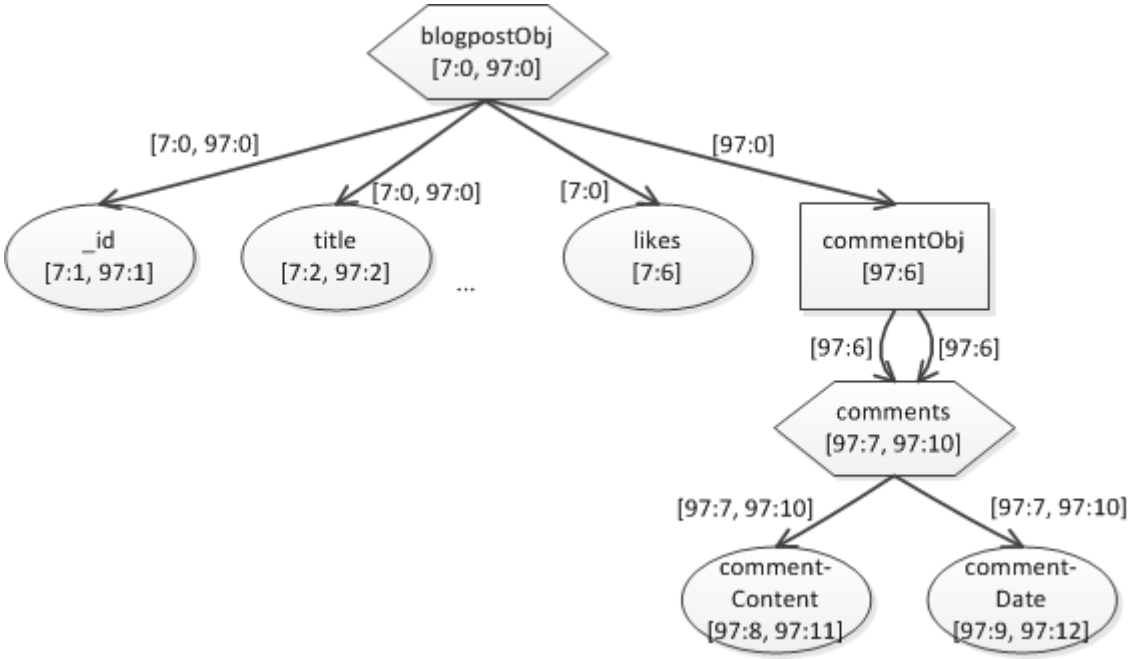
Schema Extraction

```
{ "type": "object",
  "properties": {
  "_id": {"type": "integer"},
  "title": {"type": "string"},
  "content": {"type": "string"},
  ...
  "comments": {"type": "array",
    "items": {"type": "object",
    "properties": {
    "commentContent": {"type": "string"},
    "commentDate": {"type": "string"}}
    "required": ["commentContent",
                 "commentDate"] } } }
  "required": ["title", "content",
               "author", "date"] }
```

Extraction of the Structure Identification Graph



Application of the Structure Identification Graph



We can derive:

→ JSON Schema

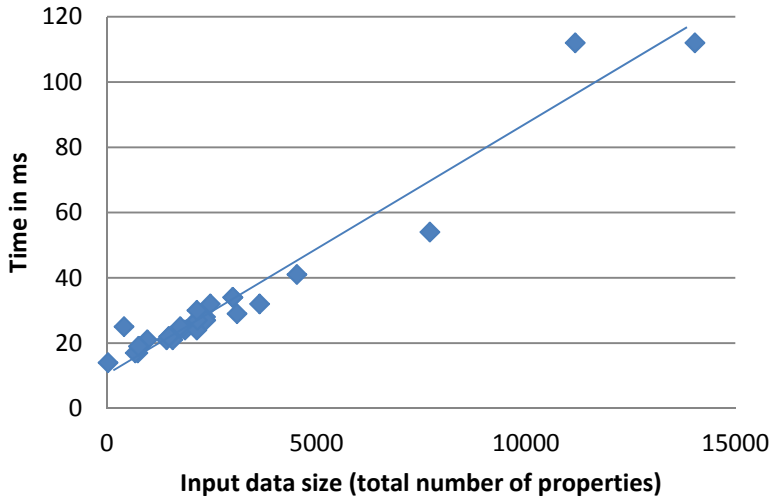
```
{ "type": "object",
  "properties": {
  .. }
```

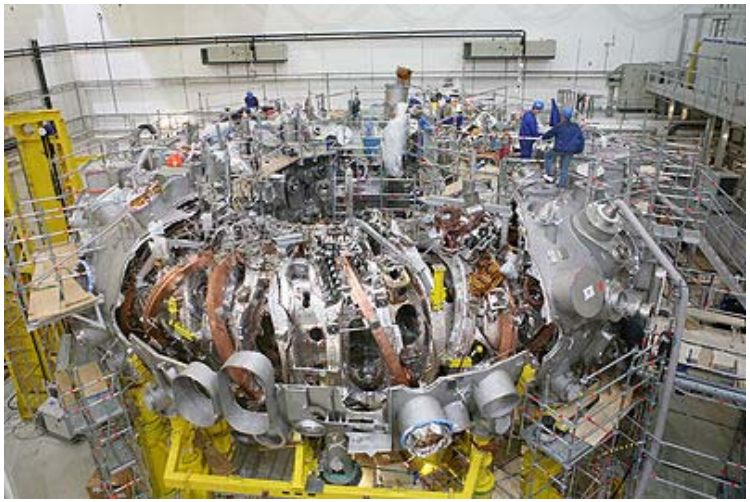
→ Statistics about the Data

→ Outliers

Scalability

- schema extraction is often a preprocessing step
- not time critical
- **bottleneck are the node and edge lists (with references to the id)**

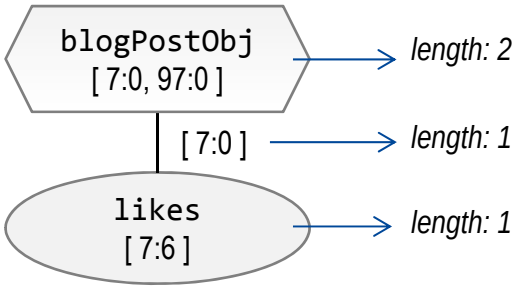




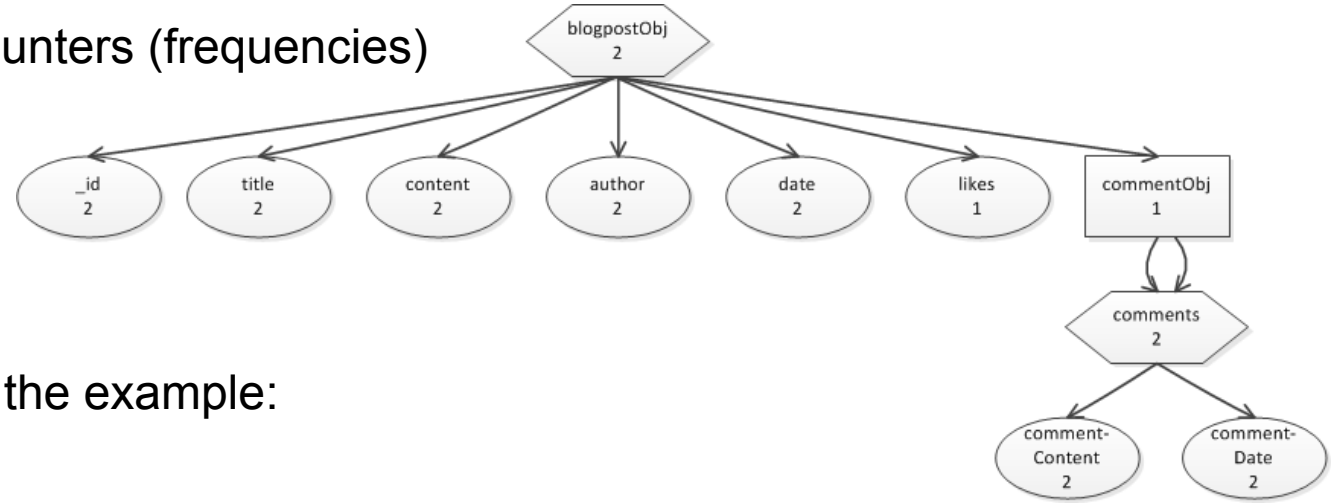
Tested with experimental data from IPP Greifswald, Wendelstein 7-X (*world's largest fusion device*), mongoDB, 180 Collections

Reduced Structure Identification Graph (RG)

- instead of the SG:



- node and edge lists do not contain references to the original documents
- only counters (frequencies)



- RG for the example:

Comparison of both Approaches

	SG – Structure Identifikation Graph	RG – Reduced Structure Identifikation Graph
JSON schema extraction	+	+
Document statistics	+	+
Outlier detection	+	-

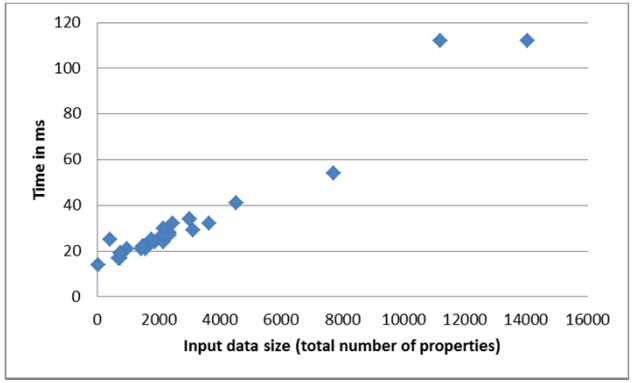
can be found in two steps

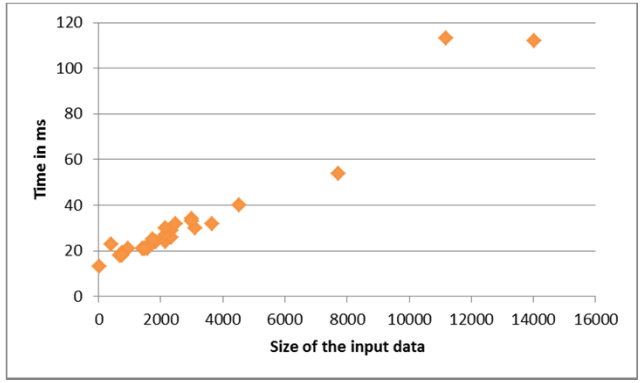
1. Finding document statistics
2. Outlier detection, with queries (e.g. in MongoDB)

```
db.collection.find({pi: {$exists: true}}
db.collection.find({pj: {$exists: false}})
```

Some Experiments

IPP Greifswald (MongoDB):

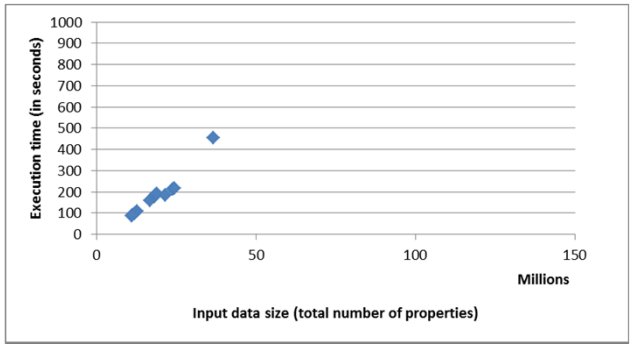


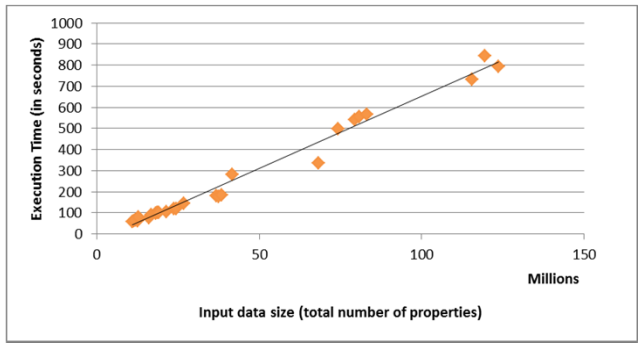


based on the SG (references)

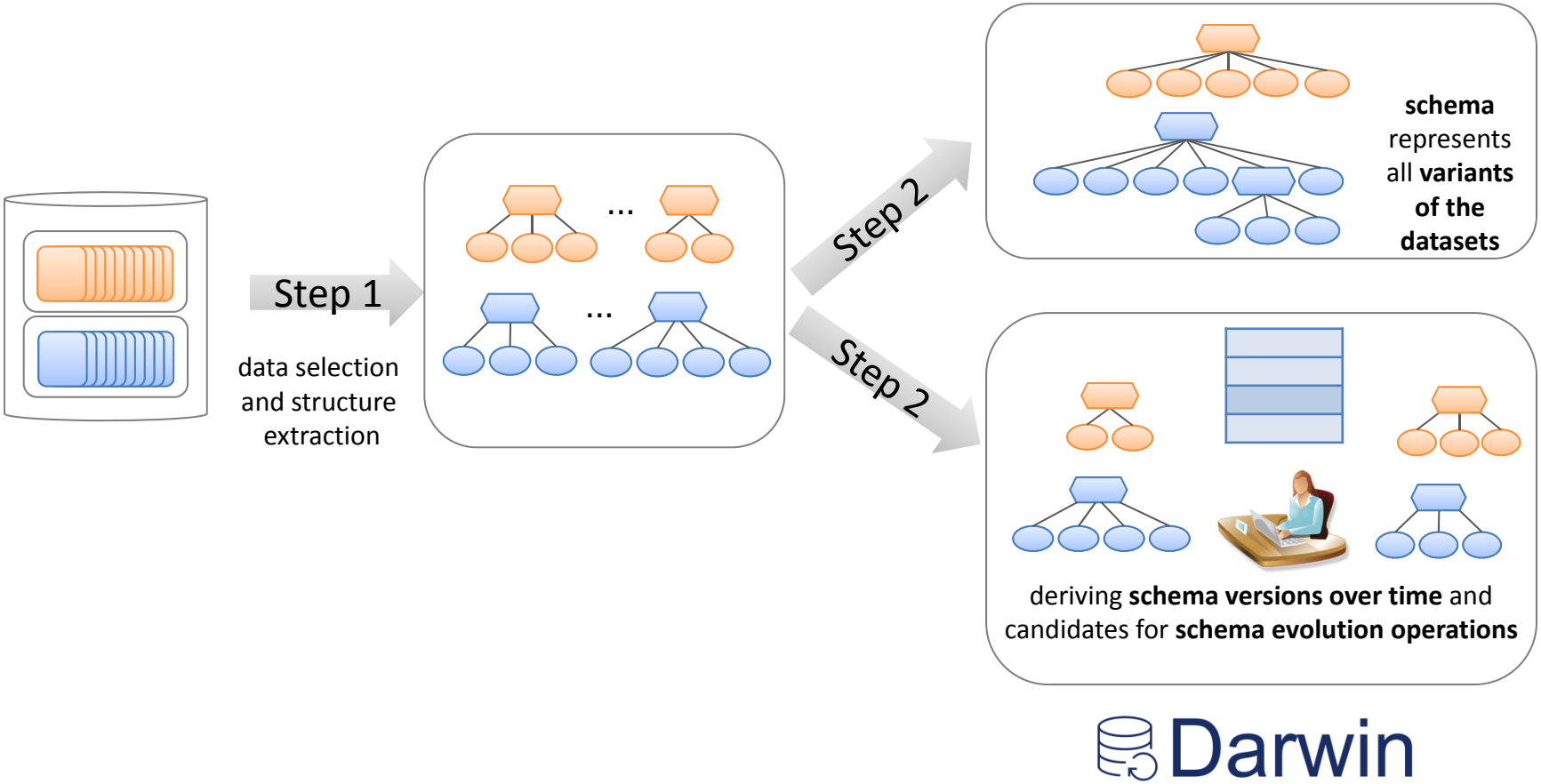
based on the RG (counters)

Web Performance Data (MongoDB):

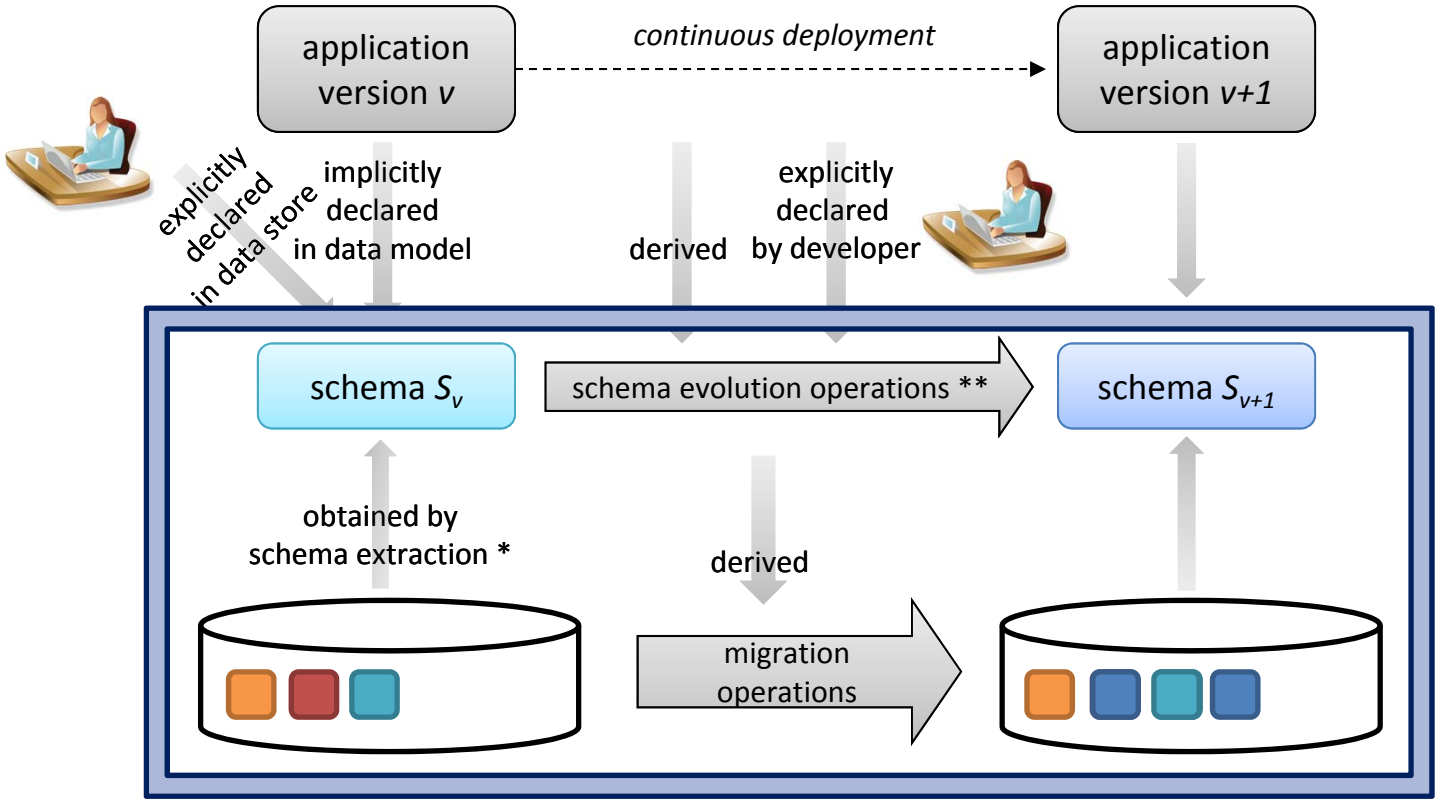




Two different interpretations of structural differences



Subtasks of Schema Management



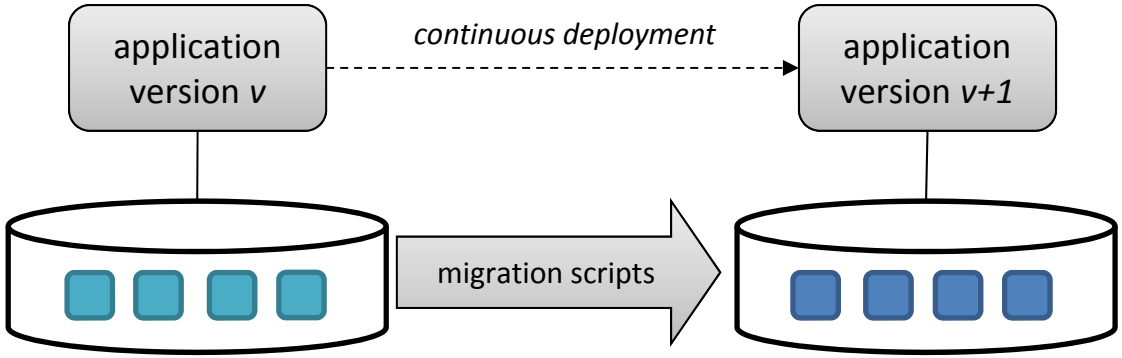
** Meike Klettke, Uta Störl, Stefanie Scherzinger: Schema Extraction and Structural Outlier Detection for NoSQL Data Stores, BTW 2015*

*** Introduced in: Stefanie Scherzinger, Meike Klettke, Uta Störl: Managing Schema Evolution in NoSQL Data Store, DBPL@VLDB, 2013*

How to realize Evolution?

State-of-the-Art and Motivation

- State-of-the-Art in NoSQL databases:
  - hand-written migration scripts

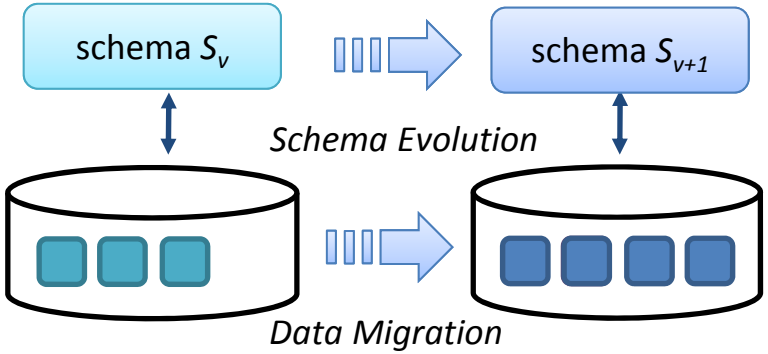


- In relational databases:
  - schema evolution is a well studied field
  - (`alter table` ..)
- In NoSQL:
  - consider data volume and data migration costs

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)



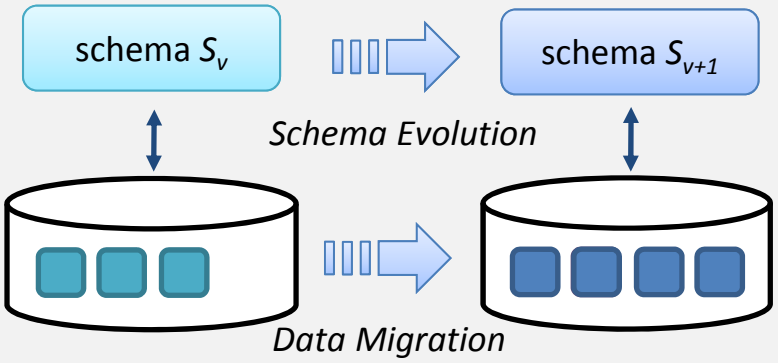
Introduced in: S. Scherzinger, M. Klettke, U. Störl: Managing Schema Evolution in NoSQL Data Store, DBPL@VLDB, 2013, implemented in Darwin

How to **scale** Data Migration?

Basic Strategies of Data Migration

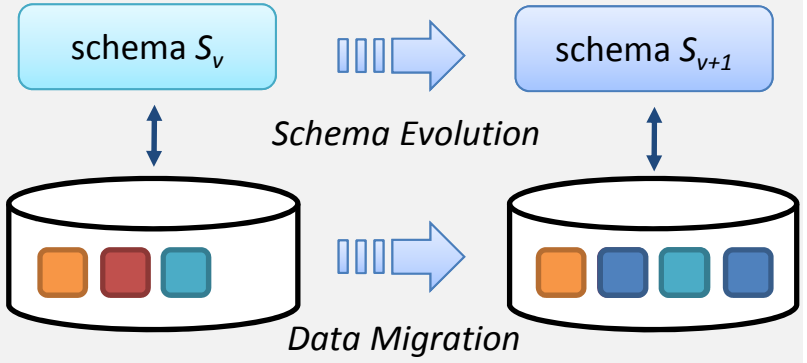
Eager Migration

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



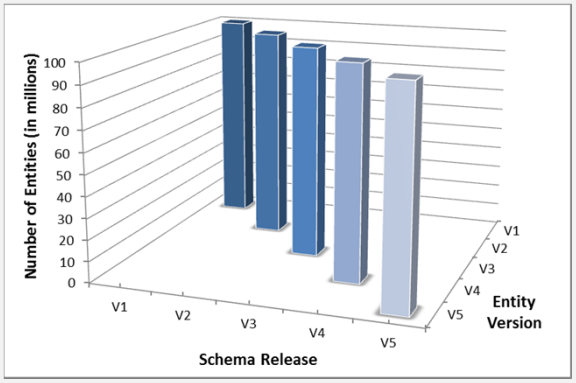
Lazy Migration

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

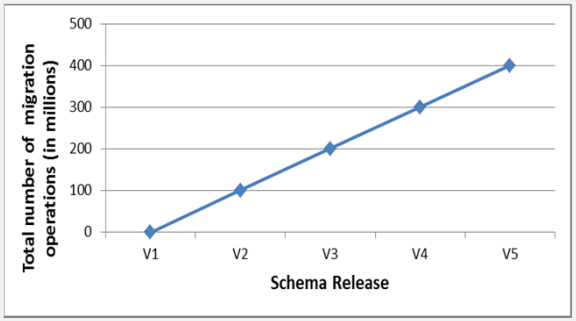


Basic Strategies of Data Migration

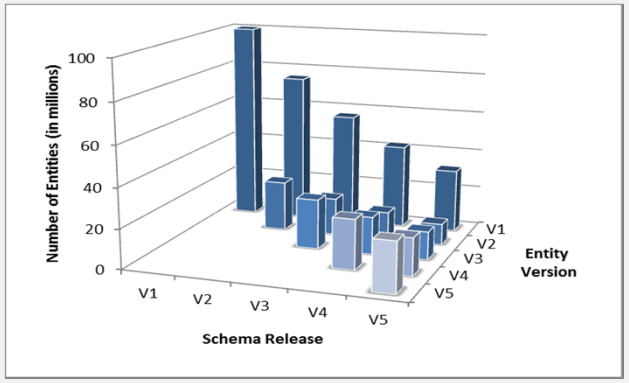
Eager Migration



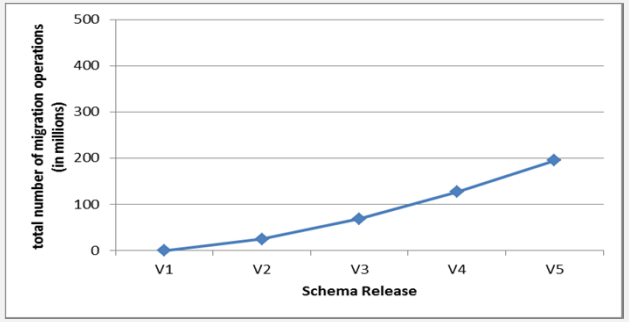
100 million entities, 5 versions



Lazy Migration



100 million entities, 5 versions,
access to 25 million entities per version



Advantages and Disadvantages

Eager Migration

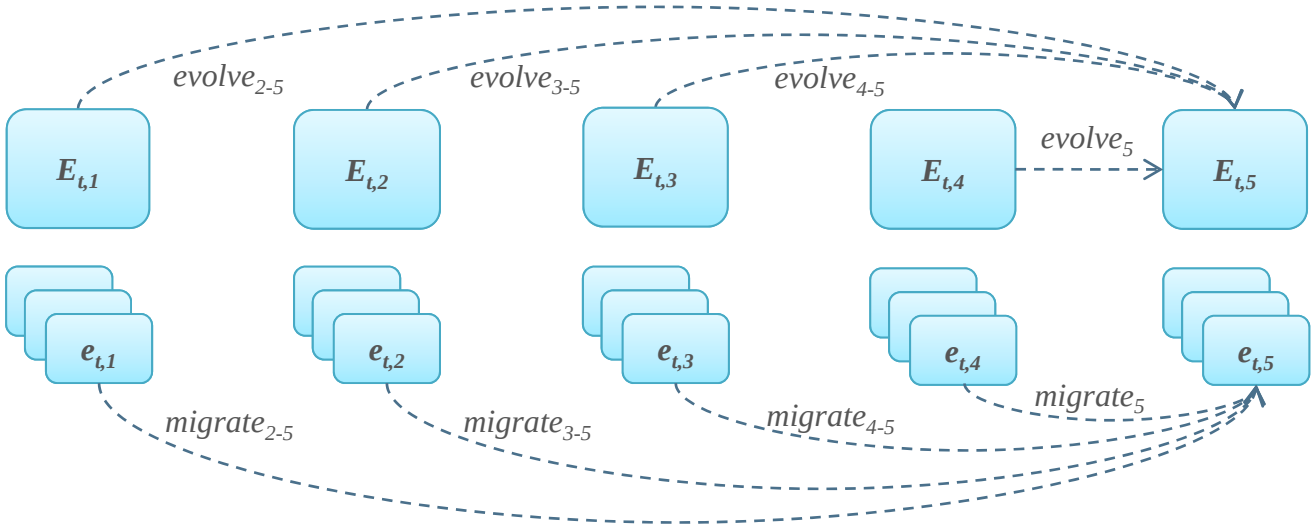
- **Advantages:**
  - \+ all entities are in the current version
  - \+ **low latency** (if entities are accessed)
- **Disadvantages:**
  - – even entities which are not in use are migrated
  - – **high number (and costs) of migration operations**

Lazy Migration

- **Advantages:**
  - \+ only entities which are in use are migrated
  - \+ **no unnecessary** data migration **operations**
  - \+ **composition of operations** is possible *(see next slides)*
- **Disadvantages:**
  - – entities in the NoSQL database in different versions
  - – increased **latency**

Optimizing Lazy Migration

In case, entities are migrated from older versions:



- **Composition of operations** is possible in some cases:
  - + **composition of evolution operations**
  - + ***smaller number of data migration operations (lower migration costs)***

Meike Klettke, Uta Störl, Manuel Shenavai, Stefanie Scherzinger: NoSQL Schema Evolution and Big Data Migration at Scale, 4th Scalable Cloud Data Management Workshop (SCDM) @ IEEE Big Data Conference, Washington, D.C., 2016

Example for **Lazy composed migration**

$\{$precondition: points $\notin E_{\texttt{blogpost}}\}$
$evolve_2$: **add** blogpost.points = 42
$\{$postcondition: points $\in E_{\texttt{blogpost}}\}$

$\{$precondition: points $\in E_{\texttt{blogpost}}$, likes $\notin E_{\texttt{blogpost}}\}$
$evolve_3$: **rename** blogpost.points **to** likes
$\{$postcondition: points $\notin E_{\texttt{blogpost}}$, likes $\in E_{\texttt{blogpost}}\}$

can be composed to:

$\{$precondition: points $\notin E_{\texttt{blogpost}}$, likes $\notin E_{\texttt{blogpost}}\}$
$evolve_{2\_3}$: **add** blogpost.likes = 42
$\{$postcondition: points $\notin E_{\texttt{blogpost}}$, likes $\in E_{\texttt{blogpost}}\}$

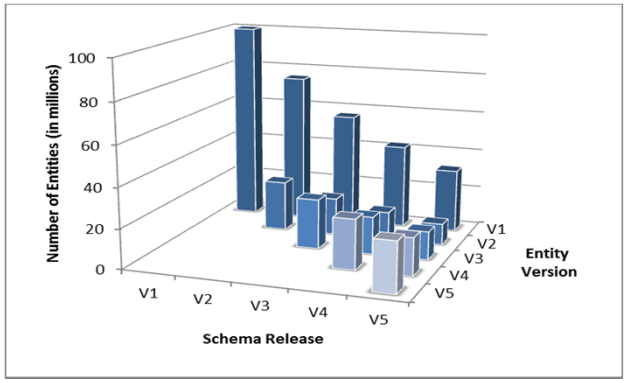
Rules for **Lazy composed migration**

op1 \ *op2*	**rename** $E_B.y$ **to** z	**copy** $E_B.y$ **to** $E_C.z$ $cond_2$	**move** $E_B.y$ **to** $E_C.z$ $cond_2$	**delete** $E_B.y$
add $E_B.y$ = *default*	**add** $E_B.z$ = *default*	-	**add** $E_C.z$ = *default* $cond_2$	ϵ (noop)
rename $E_B.x$ **to** y	**rename** $E_B.x$ **to** z	**copy** $E_B.x$ **to** $E_C.z$ $cond_2$	**move** $E_B.x$ **to** $E_C.z$ $cond_2$	**delete** $E_B.x$
copy $E_A.x$ **to** $E_B.y$ $cond_1$	**copy** $E_A.x$ **to** $E_B.z$ $cond_1$	-	**copy** $E_A.x$ **to** $E_C.z$ $cond_1 \wedge cond_2$	ϵ (noop)
copy $E_B.y$ **to** $E_D.u$ $cond_3$	-	-	-	**move** $E_B.y$ **to** $E_D.u$ $cond_3$
move $E_A.x$ **to** $E_B.y$ $cond_1$	**move** $E_A.x$ **to** $E_B.z$ $cond_1$	-	**move** $E_A.x$ **to** $E_C.z$ $cond_1 \wedge cond_2$	**delete** $E_A.y$

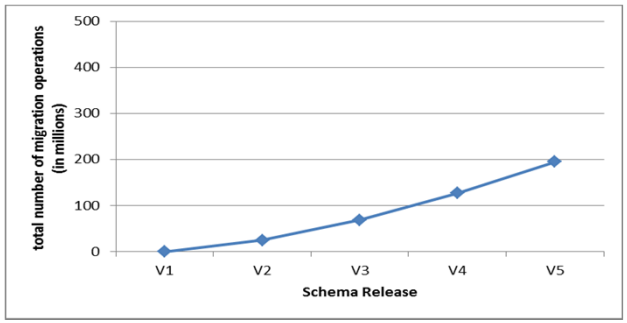
- Two schema evolution operations can be composed if they match *op1* and *op2*
- result of the composition is given in the table cells

Effects of Composition

Lazy stepwise Migration



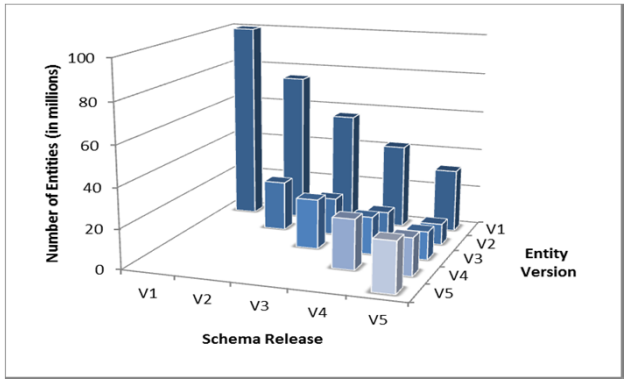
100 million entities, 5 versions,
access to 25 million entities per version



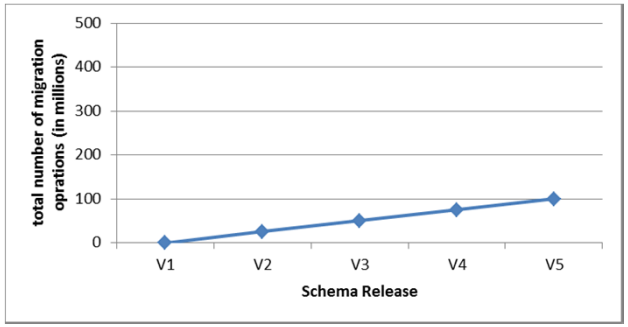
=

≠

Lazy composed Migration



100 million entities, 5 versions,
access to 25 million entities per version



Hybrid Migration: Predictive Migration

Basic idea:

- ***forecast function*** delivers ***predictions***
- ***Predictive migration*** combines ***in each version***
  - ***eager migration*** (for the predictions) and
  - ***lazy migration*** (for all other entities)

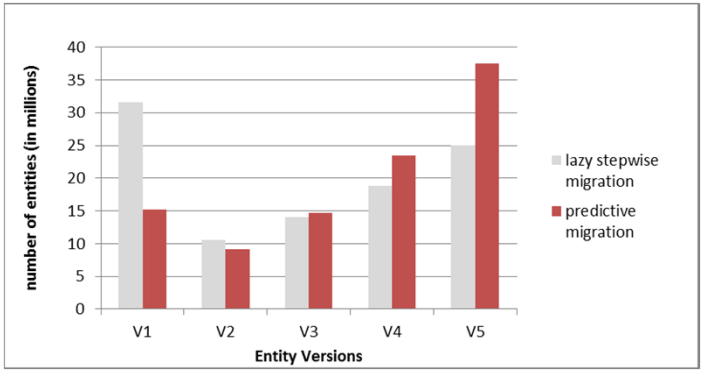
- Success depends on the quality of the forecast function
- forecast function has to consider:
  - access statistics in past
  - Relationships between entities

Meike Klettke, Uta Störl, Manuel Shenavai, Stefanie Scherzinger: NoSQL Schema Evolution and Big Data Migration at Scale, 4th Scalable Cloud Data Management Workshop (SCDM) @ IEEE Big Data Conference, Washington, 2016

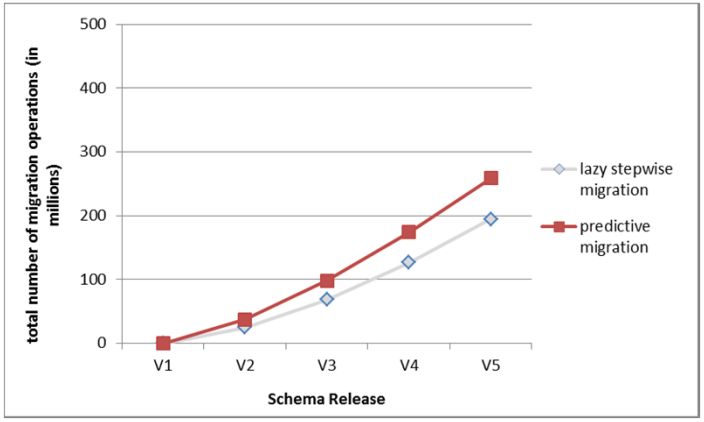
A Combination of Eager and Lazy Migration: Predictive Migration

- ***Forecast function***, which entities are accessed in near future (bases on heuristics)
- ***predictive migration*** of these entities in current version ***in bulk***

- **Advantage:**
  - decreased average latency
- **Disadvantage:**
  - additional migration operations in case of wrong predictions

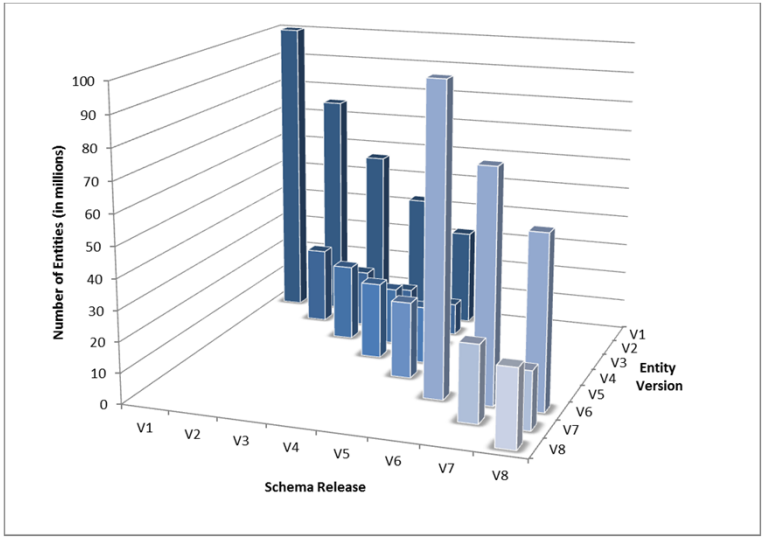


100 million entities, 5 versions,
access to 25 million entities per version,
prediction of 25 million entities, 50% correct

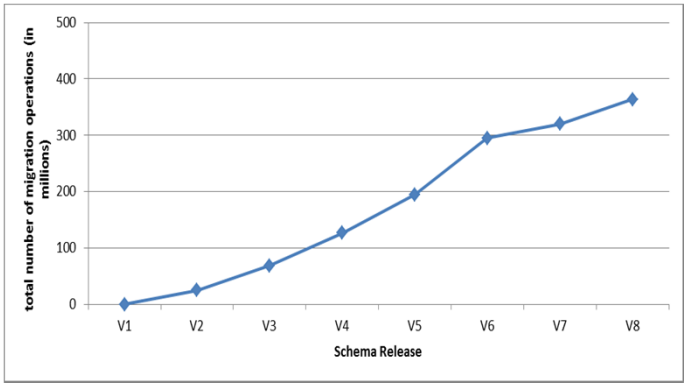


Another Hybrid Migration: Incremental Migration

- **Lazy data migration** is applied (stepwise or composed)
- ***in some version***, an **eager migration** is applied, in case of
  1. disruptive schema changes
  2. "technical debts" of the NoSQL database



- needs a function for evaluating the
  1. amount of changes
  2. state of the NoSQL database



Comparison of different approaches

1. Number of migration operations

Eager Migration	Predictive Migration	Incremental Migration	Lazy Migration	Versioning
high	medium	medium	low	none

2. Effort for query rewriting

none	medium	medium	medium	high

3. Average latency

none	low	low	medium	high

4. Application downtime

high	medium	medium	none	none

Conclusion

- Schema management components support application development with NoSQL databases
- Two different methods for **schema extraction**
- especially realizing structural changes
  - **schema evolution operations**
  - **data migration strategies**
    - eager migration (in bulk)
    - lazy migration (on request)
    - combined (hybrid migrations)

Future Work

Provide ***Schema Evolution and Data Migration*** for a **wide range of applications**

- from eager migrations up to lazy migrations
- even versioning and rewriting of queries an results

Development of an **advisor** for

- the data migration strategy
- its parametrization, and
- choice of the NoSQL database system

for the next 3 -4 years

Schema Evolution and Data Migration

- ***predictive migration -*** development and proving of a forecast function
- ***incremental migration -*** cost function for decision about lazy or eager data migration in each version

Schema Extraction

- finding and consideration integrity constraints (keys, inclusion dependencies, foreign keys, ..)

for the next year

More details?

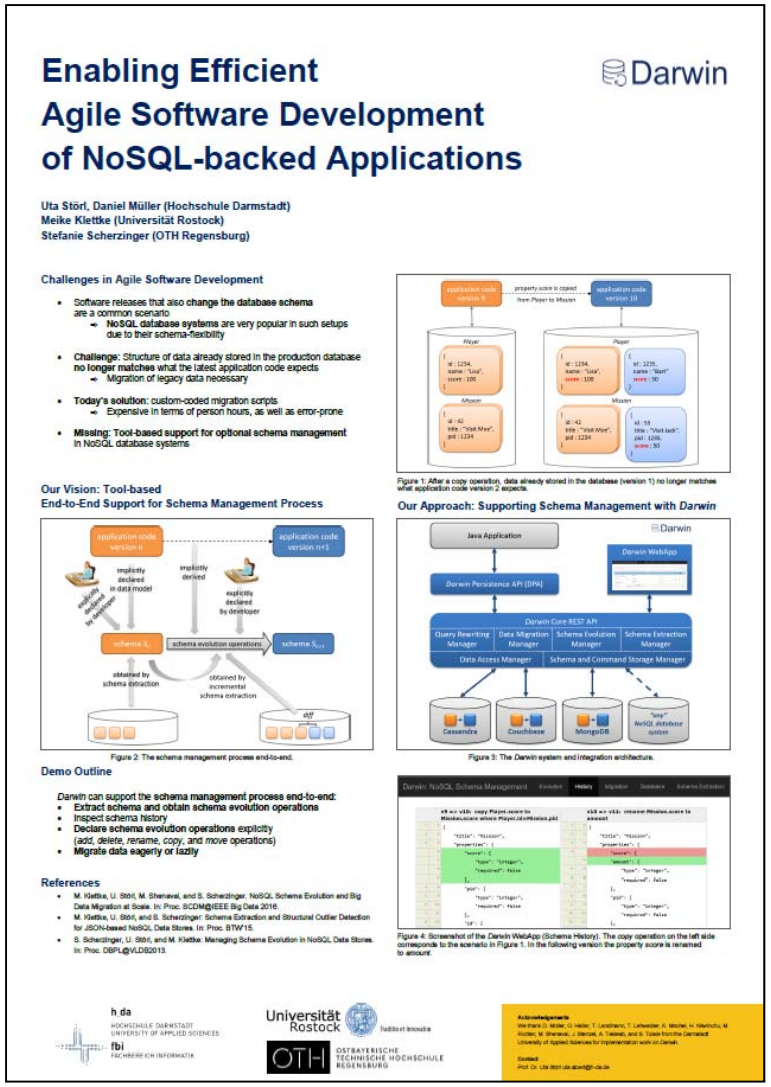
Demo Sessions

Please visit our
software demonstration

Darwin

at BTW 2017

Wednesday,
8. March: 6:00 pm – 7:30 pm
Thursday,
9. March: 10:45 am – 12:15 am



Enabling Efficient Agile Software Development of NoSQL-backed Applications

Darwin

Uta Störl, Daniel Müller (Hochschule Darmstadt)
Meike Klettke (Universität Rostock)
Stefanie Scherzinger (OTH Regensburg)

Challenges in Agile Software Development

- Software releases that also change the database schema are a common scenario
  - NoSQL database systems are very popular in such setups due to their schema-flexibility
- Challenge: Structure of data already stored in the production database no longer matches what the latest application code expects
  - Migration of legacy data necessary
- Today's solution: custom-coded migration scripts
  - Expensive in terms of person hours, as well as error-prone
- Missing: Tool-based support for optional schema management in NoSQL database systems

Our Vision: Tool-based End-to-End Support for Schema Management Process

Figure 2: The schema management process end-to-end.

Our Approach: Supporting Schema Management with *Darwin*

Figure 3: The *Darwin* system and integration architecture.

Demo Outline

Darwin can support the schema management process end-to-end:

- Extract schema and obtain schema evolution operations
- Inspect schema history
- Declare schema evolution operations explicitly (add, delete, rename, copy, and move operations)
- Migrate data eagerly or lazily

Figure 4: Screenshot of the *Darwin* WebApp (Schema History). The copy operation on the left side corresponds to the scenario in Figure 1. In the following version the property score is renamed to amount.

References

- M. Klettke, U. Störl, M. Shenavai, and S. Scherzinger. NoSQL Schema Evolution and Big Data Migration at Scale. In: Proc. SCDM@IEEE Big Data 2016.
- M. Klettke, U. Störl, and S. Scherzinger. Schema Extraction and Structural Outlier Detection for JSON-based NoSQL Data Stores. In: Proc. BTW'15.
- S. Scherzinger, U. Störl, and M. Klettke. Managing Schema Evolution in NoSQL Data Stores. In: Proc. DBPL@VLDB2013.

Literature (NoSQL databases)

- J. Baker, C. Bondç, J. C. Corbett, J. J. Furman, et al. Megastore: Providing Scalable, Highly Available Storage for Interactive Services. In Proc. CIDR'11, 2011.
- Apache Cassandra, 2013. http://cassandra.apache.org/.
- V. Benzaken, G. Castagna, K. Nguyen, and J. Simeon. "Static and dynamic semantics of NoSQL languages". In Proc. POPL, pages 101–114, 2013.
- M. Brown. Developing with Couchbase Server. O'Reilly, 2013.
- R. Cattell. "Scalable SQL and NoSQL data stores". SIGMOD Record, 39(4):12–27, 2010.
- F. Chang, J. Dean, S. Ghemawat, W. C. Hsieh, et al. "Bigtable: A Distributed Storage System for Structured Data". In Proc. OSDI, pages 205–218, 2006.
- K. Chodorow. MongoDB: The Definitive Guide. O'Reilly, 2013.
- Couch Potato, 2010. https://github.com/langalex/couch_potato/issues/14.
- Google Inc. Google Datastore Admin, 2013. https://developers.google.com/appengine/docs/adminconsole/datastoreconsole.
- Google Inc., "Google Cloud Datastore: Pricing and Quota," Oct. 2016, https://cloud.google.com/datastore/docs/pricing
- S. Tiwari. Professional NoSQL. John Wiley & Sons, 2011.

Literature (Schema Management)

- JSON Schema, 2013. http://json-schema.org/.
- MongoDB, “MongoDB Manual Version 3.2: Document Validation,” 2016, https://docs.mongodb.com/manual/core/document-validation/
- U. Störl, T. Hauff, M. Klettke, and S. Scherzinger, “Schemaless NoSQL Data Stores – Object-NoSQL Mappers to the Rescue?” in Proc. BTW’15, 2015
- Morphia. A type-safe java library for MongoDB, 2013. http://code.google.com/p/morphia/.

Literature (Schema Extraction)

- C.-H. Moh, E.-P. Lim, and W. K. Ng. DTD-Miner: A Tool for Mining DTD from XML Documents. In WECWIS, pages 144{151, 2000.
- Meike Klettke, Uta Störl, Stefanie Scherzinger: *Schema Extraction and Structural Outlier Detection for JSON-based NoSQL Data Stores*, Wissenschaftliches Programm, BTW 2015

Literature (Schema Evolution)

- D. L. Parnas. “Software Aging”. In Proc. ICSE, pages 279–287, 1994.
- G. Dong and J. Su. First-Order Incremental Evaluation of Datalog Queries. In DBPL’93, 1993.
- Objectify AppEngine. Migrating Schemas, 2012. https://code.google.com/p/objectify-appengine/wiki/SchemaMigration.
- J. McWilliams and M. Ivey. Google Developers: Updating your model’s schema, 2012. https://developers.google.com/appengine/articles/update_schema.
- Meike Klettke: Modellierung, *Bewertung und Evolution von XML-Dokumentkollektionen.* Habilitationsschrift, Universität Rostock, Fakultät für Informatik und Elektrotechnik, 2007
- S. Scherzinger, M. Klettke, and U. Störl. Managing Schema Evolution in NoSQL Data Stores. In Proc. DBPL’13, 2013.
- U. Störl, T. Hauff, M. Klettke, and S. Scherzinger, “Schemaless NoSQL Data Stores – Object-NoSQL Mappers to the Rescue?” in Proc. BTW’15, 2015
- T. Cerqueus, E. C. de Almeida, and S. Scherzinger, “Safely Managing Data Variety in Big Data Software Development,” in Proc. BIGDSE’15, 2015
- S. Scherzinger, U. Störl, and M. Klettke, “A Datalog-based Protocol for Lazy Data Migration in Agile NoSQL Application Development,” in Proc. DBPL’15, 2015
- Stefanie Scherzinger, Meike Klettke, Uta Störl: *Cleager: Eager Schema Evolution in NoSQL Document Stores*, Demoprogramm, BTW 2015

Literature (Schema Evolution, cont.)

- Meike Klettke, Stefanie Scherzinger, Uta Störl, Stephanie Sombach, Katharina Wiech: Challenges with Continuous Deployment of NoSQL-backed Database Applications, LWDA 2016
- Stefanie Scherzinger, Stephanie Sombach, Katharina Wiech, Meike Klettke and Uta Störl: Datalution: A Tool for Continuous Schema Evolution in NoSQL-backed Web Applications, QUDOS 2016

Literature (Optimization)

- C. A. R. Hoare, “An Axiomatic Basis for Computer Programming,” Commun. ACM, vol. 12, no. 10, 1969
- S. Abiteboul and V. Vianu, “Equivalence and Optimization of Relational Transactions,” J. ACM, vol. 35, no. 1, Jan. 1988
- M. Liu. Extending Datalog with Declarative Updates. In M. Ibrahim, J. Köng, and N. Revell, editors, Database and Expert Systems Applications, volume 1873 of LNCS, pages 752–763. Springer, 2000
- Meike Klettke, Uta Störl, Manuel Shenavai, Stefanie Scherzinger: NoSQL Schema Evolution and Big Data Migration at Scale, 4th Scalable Cloud Data Management Workshop (SCDM) @ IEEE Big Data Conference, Washington, D.C., USA, December 2016