Adaptability of Methods for Processing XML Data using Relational Databases – the State of the Art and Open Problems

Irena Mlynkova, Jaroslav Pokorny

{irena.mlynkova,jaroslav.pokorny}@mff.cuni.cz

Charles University

Faculty of Mathematics and Physics

Department of Software Engineering

Prague, Czech Republic

Introduction

- XML = a standard for data representation and manipulation
  - Growing demand for efficient managing and processing XML data

⇒ A natural alternative: To exploit tools and functions of (object-)relational database systems ((O)RDMS)

(–) XML trees vs. relations ⇒ inefficiency

(+) theoretical and practical history, mature technology

⇒ The techniques should be further enhanced

Goals of This Presentation

Overview of techniques which improve XML processing based on (O)RDBMS – adaptive / flexible mapping methods

- Reasons why these techniques are important and promising
- Overview and classification of existing approaches
- Discussion of possible improvements and corresponding key problems

Content

1. Why adaptive / flexible methods?
2. Overview of existing approaches
3. Open Issues
4. Conclusion

Managing XML Data

- File system
  - (–) Inability of querying without additional pre-processing of data
- Pure object-oriented approach
  - (–) Efficient and comprehensive tool
- Native methods
  - (+) No need to adapt structures to a new purpose
- (Object-)relational database system
  - (+) Most practically used

Database-Based XML Processing Methods

- Generic – store XML data regardless the existence of corresponding XML schema
  - Store XML documents as general trees (+ numbering schemes)
- Schema-driven – based on structural information from existing schema
  - XML schema $\Rightarrow$ relational schema
- User-defined – leave all the storage decisions in hands of users
  - User defines both target schema and XML-to-relational mapping

Database-Based XML Processing – Problems (1)

Problems:

- **52% of randomly crawled or 7.4% of semi-automatically collected data have no schema**
  - **Problem: schema-driven methods**
- **XSDs are used only for 0.09% of randomly crawled or 38% of semi-automatically collected data**
  - **Problem: XML Schema-driven methods**

Database-Based XML Processing – Problems (2)

- 85% XSDs define so-called local tree grammars, i.e. can be defined using DTD
  - Most common "non-DTD" features: simple types (side optimization effect)
- XML schemes are too general
  - Excessive examples: recursion or * operator
  - Problem 1: Missing structural information $\Rightarrow$ XML documents
  - Problem 2: No information on retrieval frequency $\Rightarrow$ XML queries

Solution

⇒ **A method which is able to exploit the current situation and information**

- **Existing approaches: adaptive methods**
- **Sample XML data and XML queries + XML-to-XML transformations = appropriate database schema**

  (+) Higher performance than fixed methods

  (–) Schema is adapted only once not dynamically

Content

Classification

- Cost-driven
  - Choose the most efficient storage strategy automatically
  - Evaluate a subset of possible mappings $\Rightarrow$ choose the best according to sample XML data, query workload, …
- User-driven
  - Optimization of user-defined methods
  - Default fixed mapping
  - A user can influence the mapping process using annotations = demanded storage strategies

Cost-driven methods

General Characteristics (1)

- Initial XML schema S_{init}
- Set of XML-to-XML schema transformations $T = \{t_1, t_2, ..., t_n\}$
  - $\forall i : t_i$ transforms schema S to schema S_i
- Fixed XML-to-relational mapping function f_{map}
  - Transforms schema S into a relational schema R
- Set of sample data D_{sample} characterizing the application
  - XML documents $\{d_1, d_2, ..., d_k\}$ valid against S_{init}
  - XML queries $\{q_1, q_2, .., q_l\}$ over S_{init} (+ weights $\{w_1, w_2, .., w_l\}$, $\forall i : w_i \in [0,1]$)
- Cost function f_{cost}
  - Evaluates the cost of relational schema R with regard to D_{sample}

General Characteristics (2)

- Result: optimal relational schema R_{opt}
  - $f_{cost}(R_{opt}, D_{sample})$ is minimal
- Naive storage strategy: Exhaustive search through all schema transformations
- The space is theoretically infinite $\Rightarrow$ NP hard problem $\Rightarrow$ suboptimal solution
  - Heuristics, approximation algorithms, terminal conditions…
- Note: Fixed methods = special type of cost-driven methods
  - $T = \emptyset$, $D_{sample} = \emptyset$, $f_{cost}(R, \emptyset) = const\ \forall R$

Hybrid Object-Relational Mapping (1)

- Idea: Decomposition of semi-structured XML parts into relations $\Rightarrow$ inefficient query processing
- Two types of mapping:
  - Structured parts → relations
  - Semi-structured parts → XML data type
    - XML path queries and XML-aware fulltext operations
- Key concern: To determine (semi-)structured parts
- Optimization:
  - Schema graph is first pre-processed
  - Set of transformations T is a singleton
  - Transformation is applied only on semi-structured fragments

Hybrid Object-Relational Mapping (2)

Algorithm:

1. A schema graph $G_1 = (V_1, E_1)$ is built for given DTD
2. For $\forall\ v \in V_1$ a measure of significance ω_v is determined
   - Expresses the type of structure of the fragment
   - Derived from fragment complexity, sample queries, and sample data
3. Each fragment $f \subseteq G_1$ which consists of nodes with ω_v < LOD is replaced with an attribute node having XML data type $\Rightarrow G_2$
4. G_2 is mapped to relational schema using a fixed mapping

LegoDB Mapping

- True cost-driven approach
- Non-trivial set of transformations:
  - Inlining/outlining, splitting/merging of a shared element, associativity, commutativity, exploitation of equations (a,(b|c)) = ((a,b)|(a,c)), (a+) = (a,a*), …
- Optimization: greedy search strategy
  - Choice of least expensive transformation at each iteration
  - Termination of searching if there exists no transformation t $\in$ T that can reduce the current (sub)optimum
- Evaluation of f_{cost} is optimized using statistics framework
  - The cost of schema R is estimated without its expensive construction

Other Approaches

- Similar to the previous case
- Differ in the chosen heuristics and evaluation of f_{cost}
  - Genetic algorithms
  - Hill climbing algorithm - a variation of greedy search

User-driven methods

General Characteristics

- Idea: A user is expected to help the mapping process, not to perform it
  - Optimization of user-defined strategies
- Initial XML schema S_{init}
- Set of allowed fixed XML-to-relational mappings $\{f^i_{map}\}_{i=1,\dots,n}$
- Set of annotations A, each of which is specified by name, target, allowed values, and function
- Default mapping strategy f_{def} for not annotated fragments

Mapping Definition Framework (MDF)

- **Annotations:**
  - **Storage strategy for a fragment - inlining/outling, Edge mapping, CLOB column**
  - **Storage strategy for whole schema - key/foreign key, interval encoding, Deway decimal classification**
  - **Names of tables or columns, SQL types of columns**
- **Default mapping f_{def} is user-defined**

XCacheDB System

- Annotations:
  - Similar to the previous case
  - Allowed redundancy - storing in both set of tables and a BLOB column
- Authors analyse redundancy theoretically
  - Define classes of schema decompositions
  - Finding 1: Fixed mapping strategies consider only 4NF decompositions which are least space-consuming
  - Finding 2: With further information the choice of other type of decomposition leads to more efficient query processing
    - At the cost of a certain level of redundancy

Summary (1)

1. **Storage strategy has a crucial impact on query-processing $\Rightarrow$ fixed mapping is not universally efficient**
2. **The choice of an optimal mapping strategy for a particular application is not an easy task $\Rightarrow$ it is not advisable to rely only on user's experience**
3. **The space of possible XML-to-relational mappings is theoretically infinite, most of the subproblems are NP-hard $\Rightarrow$ the exhaustive search is impossible $\Rightarrow$ necessary to define search heuristics, approximation algorithms, …**
4. **The choice of an initial schema can strongly influence the efficiency of the search algorithm $\Rightarrow$ reasonable to start with at least "locally good" schema**

Summary (2)

5. A strategy of finding a (sub)optimal XML schema should take into account the given schema, query workload, sample data, possible query translations, cost metrics, …
6. Cost evaluation of an XML-to-relational mapping should not involve expensive construction of relational schema, loading sample XML data and analyzing the resulting relational structures $\Rightarrow$ it can be optimized using cost estimation
7. A user should be allowed to influence the mapping strategy, but the approach should not demand a full schema specification
8. Even thought a storage strategy is able to adapt to a given sample of schemes, data, queries, etc., its efficiency is still endangered by later changes of the expected usage

Content

1. Why adaptive / flexible methods?
2. Overview of existing approaches
3. **Open Issues**
4. Conclusion

Open Issues (1)

- Problem of missing input data
  - Schema S_{init}:
    - Is the user-given schema a good candidate for S_{init}?
    - How can we measure this quality?
    - How can we find a better candidate?
    - Can we find it for schema-less XML documents?

      $\Rightarrow$ automatic construction of schema
  - XML documents $\Rightarrow$ analyses of real XML data
    - Default settings
  - XML queries $\Rightarrow$ crucial problem
    - No analyses on real XML queries (no source to analyze)
    - Collect dynamically?

Open Issues (2)

- Efficient solution of subproblems
  - Numerous simplifications
    - Omitting of ordering, mixed contents, recursion, …
  - Default mapping of schema-driven methods is fixed ⇒ combination with cost-driven idea
- Deeper exploitation of user-given information ⇒ pattern matching
  - Idea: User-given schema annotations = "hints" how to store particular XML patterns
  - Similar fragments should be stored similarly

    ⇒ exploitation in searching an efficient mapping for not annotated parts

Open Issues (3)

- Theoretical analysis of the problem
  - No theoretic study of XML-to-XML transformations (classification, characteristics, …) + NP-hardness
- Dynamic adaptability
  - Changes of queries / data $\Rightarrow$ crucial worsening of efficiency $\Rightarrow$ dynamic adaptability
    - Problem of missing input queries
  - Requirement: To avoid reconstruction of whole schema
    - Idea 1: Local changes $\Rightarrow$ local reconstructions + similarity and pattern matching
    - Idea 2: Analysis of gradual minor changes vs. not often greater reconstructions

Content

1. Why adaptive / flexible methods?
2. Overview of existing approaches
3. Open Issues
4. **Conclusion**

Solved Open Issues

- Proposal of an adaptive schema-driven mapping method
  - Able to find fragments similar to the user-specified annotations
  - Able to map the remaining parts of the schema adaptively using the user-specified annotations
  - Proposal of a similarity measure and its tuning based on statistics of real-world data
- Further improvements:
  - Combination with true cost-driven methods
    - User provides annotations + sample data and queries
  - Dynamic adaptability

Thank you