Purifying XML Structures
Ph.D. Defense

Taro L. Saito
University of Tokyo
<leo@cb.k.u-tokyo.ac.jp>

http://www.xerial.org/
Outline

Purifying XML Structures: Ph.D. Defense

• Introduction
  – XML and Structural Fluctuation
  – Amoeba Join

• Purifying XML Structures
  – Functional Dependencies for XML
  – Amoeba Join Decomposition
  – Ubiquitous Keys

• Implementation
  – Amoeba Join Processing Algorithms
  – XML Indexing
  – Experimental Results

• Conclusions
  – Applications
  – Summary of Contributions & Future Work

http://www.xerial.org/
Introduction

Purifying XML Structures: Ph.D. Defense

- **XML** (Extensible Markup Language)
  - A markup language representing a tree structure
  - Since 1996, XML has been broadly used as a data representation format

- **Major drawbacks**
  - Hierarchical representation of data is too complex
    - for both of human and computer programs
    - reminiscences of 1970s’ discussion
      - Relational v. s. Hierarchical DB
  - There exist many alternative tree structures
    - to represent a same data model

```xml
<bookstore>
  <order>
    <customer>John</customer>
    <book>
      <title>Data on the Web</title>
    </book>
  </order>
</bookstore>
```
Structural Fluctuation

- **Differently Structured XML Documents**
  - representing a same data model e.g. Amazon.com
    - for order, customer, book nodes
  - The hierarchical order of *order* and *customer* is reversed.
  - The *order* node is behind the *pending* node.

http://www.xerial.org/
Querying Structural Fluctuation

- Standards of XML Processing: XPath, SAX, DOM, etc.
- Many parse states:
  - If we find an order, then parse customer and book
  - or if we first find an customer, then parse pending/order and book …
  - Such query processing is tedious and error-prone!
- Why we need different programs to parse the same meaning XML data?
Structural Fluctuations

- In general, the number of structural fluctuations of \( n \) nodes is \( n^{(n-1)} \)
  - Enumeration of labeled trees of \( n \) nodes
Current Solution

- Disallow structural fluctuations by using a schema
  - XML Schema, DTD, RelaxNG, etc.

- However, fixing a tree structure involves irrelevant work in defining a data model.
  - Why we have to choose only one tree structure?
Heuristic Approach

SLCA (Smallest Lowest Common Ancestor)
- [Li, VLDB2004], [Xu, SIGMOD2005]
- An lca node that does not contain other lca nodes.
- However, it easily leads to unintended results

slca of (customer, book)

http://www.xerial.org/
• **Amoeba Join:** $AJ(\text{order, customer, book})$
  - [WebDB 2006]
  - retrieves node tuple such that
    - one of \((\text{order, customer, book})\) nodes is a common ancestor of the others.
  - Handles every structural fluctuation
Semantics of XML Structures

Semantics implied in XML data
- Each order node should have a single book node
  - Invalid structure might be retrieved without considering such semantics of data.
- Instances of such invalid structures could be numerous

To represent semantics of XML data, we introduce functional dependencies for XML

http://www.xerial.org/
Functional Dependency (FD)

- **Functional Dependency**
  - $X \rightarrow Y$: if two tuples $p, q$ agree with $X$, then also agree with $Y$

<table>
<thead>
<tr>
<th>order</th>
<th>book</th>
<th>title</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>b1</td>
<td>Database Systems</td>
</tr>
<tr>
<td>2</td>
<td>b1</td>
<td>Database Systems</td>
</tr>
<tr>
<td>3</td>
<td>b2</td>
<td>Data on the Web</td>
</tr>
</tbody>
</table>

- **FDs: order $\rightarrow$ book, book $\rightarrow$ title**

<table>
<thead>
<tr>
<th>order</th>
<th>book</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>b1</td>
</tr>
<tr>
<td>2</td>
<td>b1</td>
</tr>
<tr>
<td>3</td>
<td>b2</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>book</th>
<th>title</th>
</tr>
</thead>
<tbody>
<tr>
<td>b1</td>
<td>Database Systems</td>
</tr>
<tr>
<td>b2</td>
<td>Data on the Web</td>
</tr>
</tbody>
</table>

- **FD is generally used to avoid redundancies of data**
  - Normal Form

http://www.xerial.org/
Data Modeling & FD

- FD has an essential role in data modeling
  - describe one-to-one, one-to-many, many-to-many relationships
    - ex. ER (Entity-Relationship) diagram, UML (Unified Modeling Language)
- Example
  - order \(\rightarrow\) book, order \(\rightarrow\) customer
    - An order has a book. An order has a customer.
    - A book has many orders. A customer has many orders (one-to-many)
  - book \(\rightarrow\) title, title \(\rightarrow\) book
    - A book has a title; a title belongs to a book (one-to-one)
  - customer, book \(\rightarrow\) order
    - An order connects many customers and books (many-to-many)
Functional Dependencies for XML

Previous Work of FDs for XML
- [Buneman et al., WWW2001], [Arenas and Libkin, TODS2004]
- based on fixed paths
  - Because there was no counterpart of relation (tables) in XML
- e.g. /order/book
  - Structural fluctuations are not allowed:

```
<order id="1">
  <book isbn="xx1"/>
  <customer id="c001"/>
</order>
```

```
<book isbn="xx1">
  <order id="1">
    <customer id="c001"/>
  </order>
</book>
```

- In reality, however, the constraint on the path, a book must be a child of an order, is too strong.
- Their definition has no loss-less decomposition
Relation in XML

- Relation in XML allows a zigzag shape
- For an FD: \(\text{book}, \text{customer} \rightarrow \text{order}\)
  - (book, customer, order) must be an amoeba

FD:
- \(\text{book} \rightarrow \text{customer} \rightarrow \text{order}\)
- \(\text{order} \rightarrow \text{book}\)
- \(\text{order} \rightarrow \text{customer}\)

\(D_1 = /\text{book/title}\)
\(D_2 = \{\langle \text{book}, \text{customer}, \text{order} \rangle\}
\& \{\langle \text{order}, \text{book} \rangle\}
\& \{\langle \text{order}, \text{customer} \rangle\}\)
A set of FDs defines XML structures

- **Traditional Approach:**
  - XML data (Structured Data) -> Data Model

- **Our approach:** **Data Model (FD) -> XML Structures**
  - Allows various XML structures to describe a data model
  - Enhancing expressive power of XML databases
Amoeba Join Satisfying FDs

FD:
- book \( \rightarrow \) customer \( \rightarrow \) order
- order \( \rightarrow \) book
- order \( \rightarrow \) customer

- AJ \(_F\) (order, book, customer)
  - retrieves a relation in XML satisfying a set \( F \) of FDs

- Makes easier managing multiple hierarchies of XML tree structures
  - An amoeba join \( AJ \(_F\) \) (order, book, customer) can track \( D_2 \)

http://www.xerial.org/

Amoeba Join Satisfying FDs

http://www.xerial.org/
Amoeba Join Decomposition

\[
\begin{align*}
\text{AJ}_B, c, o \\
\text{AJ}_O, c \\
\text{AJ}_O, B \\
\text{AJ}_O, c \\
\text{AJ}_B, c, o \\
\end{align*}
\]

\[
\begin{align*}
f_1: B, C \rightarrow O \\
f_2: O \rightarrow B \\
f_3: O \rightarrow C
\end{align*}
\]

\[
\begin{align*}
\text{AJ}_F(B, C, O) & = \text{AJ}_B, c, o(\text{AJ}_F(B, C, O)) \\
& = \text{AJ}_B, c, o(\text{AJ}_O, B(\text{AJ}_F(C, O), B)) \\
& = \text{AJ}_B, c, o(\text{AJ}_O, B(\text{AJ}_O, c(C, O), B))
\end{align*}
\]

\[
\begin{align*}
\text{AJ}_F(B, C, O) & = \text{AJ}_O, c(\text{AJ}_F(B, C, O)) \\
& = \text{AJ}_O, c(\text{AJ}_O, B(\text{AJ}_F(B, C, O))) \\
& = \text{AJ}_O, c(\text{AJ}_O, B(\text{AJ}_B, c, o(B, C, O)))
\end{align*}
\]
FD Based XML Query Processing

- No explicit path structures are required
- Examples:
  - FDs
    - book, customer ↦ order • order ↦ book • order ↦ customer
  - A query for book and order node: $AJ_F(\text{book, order})$
    - book and order nodes compose amoebas
  - A query for book and customer nodes:
    - $AJ_F(\text{book, customer})$
      - book and customer nodes might be connected through order nodes
    - Thus, $AJ_F(\text{book, customer, order})$ is evaluated
- Relation in XML is dynamically determined according to query targets
Functional Dependencies and Keys

- **Key is a special case of a functional dependency**
  - e.g. order (id) → book, customer
    - order (id) is a key

<table>
<thead>
<tr>
<th>order</th>
<th>book</th>
<th>customer</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>b1</td>
<td>John</td>
</tr>
<tr>
<td>2</td>
<td>b2</td>
<td>Lucy</td>
</tr>
</tbody>
</table>

- **Using a relation in XML, we can define keys for XML**
  - [order@id] → book, customer
    - Given an order id, we can uniquely determine book and title nodes
    - XML structures: <<order, book>>, <<order, customer>>

- **More general description of keys**
  - In [Buneman, et al. WWW2001], it is not allowed to reverse the position of order and book nodes

http://www.xerial.org/
Ubiquitous Keys

Purifying XML Structures: Ph.D. Defense

FD:

book → order
order → book
customer → order

(Q1): for $x$ in /seller/book_list/book
    $y$ in /seller/customer_list/customer/pending/order/book
    where $x/@isbn = y/@isbn$
    return $x/title$

Querying without using Structures

- **AJ**(book, [pending, order, title])
  - book nodes are merged using ubiquitous keys
Amoeba Join Processing
**Sweep Amoeba Join Algorithm**

**Purifying XML Structures: Ph.D. Defense**

- Fetch all input nodes
  - \(AJ(\text{org, manager, location})\)
  - Sort input nodes in their document orders.

- **Sweep sorted input nodes**
  - Assume the smallest node in the input nodes as an amoeba root.
  - Search their descendant regions for components of amoebas.

---

http://www.xerial.org/

---

http://www.xerial.org/
Disk I/O Optimization

Purifying XML Structures: Ph.D. Defense

- \( \text{AJ(\text{org, manager, location = "Tokyo"})} \)
  - Choose pivot nodes from a small input domain
  - Traverse upward to find amoeba root candidates
  - Search space for amoeba is localized under the amoeba root candidates.

http://www.xerial.org/
XML Indexing
History of XML Indexing

A hundreds of XML indexing papers ...

- tailored to specific queries
  - XPath query, structural-join (A//D), twig-queries, text search, etc.
- from many research areas
  - Database Community
    - Node labeling (static or updatable)
  - Information Retrieval (IR)
    - inverted indexes for text data. SLCA (2005)
  - Compressed Index
    - XBW (Ferrangina, WWW2006)

http://www.xerial.org/
Multidimensional Aspects of XML

Purifying XML Structures: Ph.D. Defense

- Tree-Structure Index
  - Ancestor, Descendant (subtree), Sibling
- Path-Structure Index
  - Suffix-path (//headline/item)
- An XML Index that can process both of the structures simultaneously is strongly required

://www.xerial.org/
Our Approach

Purifying XML Structures: Ph.D. Defense

- [DASFAA2007]
- Integrating tree-structure and path indexes
  - As a multidimensional index
    - (start, end, level, path)
  - It can be implemented on top of the B+-tree

Why B+-trees?

- Index structures and transaction management, recovery, logging, caching etc. are interdependent.
- We already have many transaction management techniques on B+-trees
  - Transaction management on R-tree is not seriously supported.

http://www.xerial.org/
Inverted-Path Index

Purifying XML Structures: Ph.D. Defense

Align inverted paths in the lexicographical order
- facilitates suffix path queries

Examples (suffix-path query range):
- //item [6, 11)
- //headline/item [6, 8)
**Z-Order**

- Align multidimensional points (nodes) in z-order
  - Interleave function gives z-order in the multidimensional space
- Each step in z-orders splits slices into two

---

http://www.xerial.org/

---

Xerial
An Innovation of the XML DBMS
Range Query

- Traverse B+-tree in the order of z-order

http://www.xerial.org/
Experimental Results
Implementation

Purifying XML Structures: Ph.D. Defense

- **Xerial**
  - [http://www.xerial.org/](http://www.xerial.org/)
  - XML Database Management System
    - XML data is multi-dimensionally indexed
    - supporting amoeba joins & XPath queries
  - Implemented in C++
    - about 150,000 lines of codes
      - Query compiler & scheduler, query processing algorithms
      - Database indexing, XML processor, etc.

- **Machine environment for experiments**
  - Windows XP notebook
  - Pentium M 2GHz, 1GB Main Memory
  - 5,400 rpm HDD (100GB)
Database Size

Purifying XML Structures: Ph.D. Defense

• Data set: XMark Benchmark XML Document
• Xerial is space-efficient

http://www.xerial.org/
Suffix-Path Query Performance

Purifying XML Structures: Ph.D. Defense

http://www.xerial.org/

- Xerial & path-start index is fastest
Subtree Retrieval Performance

All of the indexes show similar performance
- XML nodes are sorted in the order of start values

http://www.xerial.org/
Ancestor Retrieval

- The number of the previous nodes of a context node affects the ancestor-query performance.

http://www.xerial.org/
Sibling Query Performance

- Without indexes for level-values, retrievals of sibling nodes are inefficient

http://www.xerial.org/
# Amoeba Join Performance

## Purifying XML Structures: Ph.D. Defense

<table>
<thead>
<tr>
<th></th>
<th>XMark (factor = 0.1, 12M)</th>
<th>XMark (factor = 0.5, 57M)</th>
<th>XMark (factor = 1.0, 114M)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>QK  SW/I  SW/S  BF/I  BF/S</td>
<td>QK  SW/I  SW/S  BF/I  BF/S</td>
<td>QK  SW/I  SW/S  BF/I  BF/S</td>
</tr>
<tr>
<td>Q1</td>
<td>2.71 0.39 5.47 &gt; 8d  &gt; 8d</td>
<td>22.91 1.97 30.81 &gt; 3y  &gt; 3y</td>
<td>62.20 4.17 69.09 &gt; 24y  &gt; 24y</td>
</tr>
<tr>
<td>Q2</td>
<td>0.06 0.32 5.57 106.75 115.94</td>
<td>0.05 1.20 29.34 &gt; 0.5h  &gt; 0.5h</td>
<td>0.06 2.67 67.12 &gt; 11h  &gt; 11h</td>
</tr>
<tr>
<td>Q3</td>
<td>0.05 0.11 5.43 20.02 26.42</td>
<td>0.07 3.97 29.41 &gt; 0.1h  &gt; 0.1h</td>
<td>0.06 8.95 66.02 &gt; 0.5h  &gt; 0.5h</td>
</tr>
<tr>
<td>Q4</td>
<td>0.06 0.41 7.98 &gt; 30y  &gt; 30y</td>
<td>0.05 10.96 43.41 &gt; 162c  &gt; 162c</td>
<td>0.07 22.12 90.95 &gt; 2631c &gt; 2631c</td>
</tr>
</tbody>
</table>

Q1: AJ(emph, bold, keyword)  
Q2: AJ(emph, bold, keyword=>"aboard notes")  
Q3: AJ(item, @id="item100", description)  
Q4: AJ(item, @id="item100", description, location, text)  

- **Algorithm**  
  - QK: Quicker, SW: Sweep, BF: Brute Force  
- **Index**  
  - I: Index Scan, S: Sequential Scan  
- **Quicker algorithm is fastest when we can localize search regions**

---

http://www.xerial.org/
Improvement by AJ Decomposition

Purifying XML Structures: Ph.D. Defense

<table>
<thead>
<tr>
<th>query statement</th>
<th>1M</th>
<th>#result</th>
<th>5M</th>
<th>#result</th>
<th>25M</th>
<th>#result</th>
<th>50M</th>
<th>#result</th>
</tr>
</thead>
<tbody>
<tr>
<td>Q1 AJ(book, order)</td>
<td>0.24</td>
<td>9403</td>
<td>1.14</td>
<td>46146</td>
<td>5.60</td>
<td>223637</td>
<td>11.34</td>
<td>447174</td>
</tr>
<tr>
<td>Q2 AJ(customer, order)</td>
<td>0.14</td>
<td>9403</td>
<td>0.64</td>
<td>46146</td>
<td>3.24</td>
<td>223637</td>
<td>6.44</td>
<td>447174</td>
</tr>
<tr>
<td>Q3 AJ(order, book, customer)</td>
<td>3.00</td>
<td>583603</td>
<td>14.89</td>
<td>2857746</td>
<td>Out of Memory</td>
<td>Out of Memory</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Q4 AJF(order, book, customer)</td>
<td>0.51</td>
<td>9403</td>
<td>2.11</td>
<td>46146</td>
<td>10.41</td>
<td>223637</td>
<td>21.25</td>
<td>447174</td>
</tr>
<tr>
<td>Q5 AJ*(book, order)</td>
<td>0.32</td>
<td>16094</td>
<td>1.51</td>
<td>79102</td>
<td>7.46</td>
<td>383420</td>
<td>15.17</td>
<td>766740</td>
</tr>
<tr>
<td>Q6 AJ*(book, [order, pending, title])</td>
<td>1.17</td>
<td>66094</td>
<td>7.74</td>
<td>359102</td>
<td>112.08</td>
<td>1753420</td>
<td>486.66</td>
<td>3516740</td>
</tr>
<tr>
<td>Q7 AJ*F(book, [order, pending, title])</td>
<td>0.91</td>
<td>16594</td>
<td>4.30</td>
<td>81902</td>
<td>21.19</td>
<td>397120</td>
<td>88.51</td>
<td>792420</td>
</tr>
</tbody>
</table>

- Without decomposing amoeba joins, the number of XML structures to be retrieved explodes.
Perspectives
Applications

Our methods can be applied various XML databases

Examples of promising applications

File Systems
- Represent files with XML format
  - reorganization and enhancing information of files with tags

Bioinformatics
- Reorganization of data is frequent
  - Statistical analysis (classification, transformation, cleansing, etc.)
- Integration of various data sources is required
**SCMD**

**SCMD** (Saccharomyces Cerevisiae Morphological Database)
- [NAR04], [NAR05], [PNAS05]
Deep Copies of XML Data

Purifying XML Structures: Ph.D. Defense

- Many duplicates (deep copies) of data

Xerial
An Innovation of the XML DBMS

http://www.xerial.org/
Shallow Copies of XML Data

Purifying XML Structures: Ph.D. Defense

- Graph-structured data model can be decomposed into several trees
- To connect nodes in trees, we need shallow copies of nodes.

With FD-based query processing
- It becomes easier to manage shallow-copy representation of XML data

http://www.xerial.org/
Future Work

Purifying XML Structures: Ph.D. Defense

• **Query Optimization**
  - Efficient amoeba join decomposition scheduling
  - Integration of index-lookups and cost-based optimization
  - Indexes for amoeba structures

• **More complex semantics**
  - Ownerships of nodes
  - Scope of attributes

• **Updates of XML Data**
  - Detecting violation of FDs
  - Automatically constructs XML structures
    - From unstructured data

```xml
<dept name="R&D">
  <manager>David</manager>
  <section name="Biology">
    <manager>Kevin</manager>
  </section>
</dept>
```

http://www.xerial.org/
Our Contributions

- **Amoeba Join**
  - Tracks various XML structures
- **Functional Dependency**
  - defines XML structures of interest
  - Conceptual change: *Data model (FD) defines XML structures*
- **Amoeba Join Decomposition**
  - makes faster the FD-based query processing
- **XML Indexing**
  - A space-efficient XML indexing technique
Thank you!

This is the end of the presentation

http://www.xerial.org/