XML: New Challenges and Opportunities for Database Research

Bongki Moon
Department of Computer Science
University of Arizona
Tucson, AZ 85721, U.S.A.
bkmoon@cs.arizona.edu
Road Map

• Introduction: Basic understanding of XML
• Case Study: Microarray gene expr data
• Challenges and Proposed Solutions
  ▪ XML to RDB Mapping
  ▪ Indexing XML data for Path Queries
    • XISS/Numbering Scheme and Path Joins
  ▪ Selectivity Estimation
• Concluding Remarks
HTML vs. XML

- HTML tags are mostly for formatting purpose. How something should look.
  - Most XML documents are exchanged by applications that do not display them for human consumption.
- HTML tags are not informative about the contents.
  - XML tags are self-describing and can be tailored for different categories of applications.
- HTML files are too forgiving; not suitable for automatic processing.
  - XML data is well-formed and its validity may be checked by applications or processors.
Extensible and Self-describing

- No preset semantics for any XML tag
  - Different semantic layers are supposed to be built atop XML data.
  - Create your own *markup language*. (E.g., GEML, MAML)

- **eXensible**: You can create your own *tags*.

```
<author>Bongki Moon</author>
<picture filename="tiger.jpg" />
<pattern name="XYZ">
  <reporter active_sequence="TCTCACTGGTCA">
    ....
  </reporter>
  <position x="0.3", y="0.5" />
</pattern>
```
Well-formed but Semi-structured

• **Well-formedness**
  - Every XML document has a root element
  - Matching opening-closing tags; properly nested
  - An attribute can occur at most once in an opening tag, and its value must be provided.

• **Tree-structured**
  - Not as structured as relational databases
  - *Need a special-purpose DBMS for XML data?*
More Specifics of XML

- Elements are ordered; attributes are *not* ordered
- *String* is one and only primitive data type in XML; complex data types can be defined in *XML Schema*
- DTD is a set of rules that specify which elements and attributes are allowed in a custom markup language.

```xml
<!DOCTYPE bioinfo_bib [ 
  <!ELEMENT bioinfo_bib (article*)> 
  <!ELEMENT article (author+,title,source)> 
  <!ELEMENT author (#PCDATA)> 
  <!ELEMENT author email_address> 
  .... 
]>
```

- DTD defines the structural relationship between elements.
- A standard maintained by the WWW Consortium ([www.w3c.org/XML](http://www.w3c.org/XML))
Road Map

- Introduction: Basic understanding of XML
- Case Study: Microarray gene expr data
- Challenges and Proposed Solutions
  - XML to RDB Mapping
  - Indexing XML data for Path Queries
    - XISS/Numbering Scheme and Path Joins
  - Selectivity Estimation
- Concluding Remarks
Case Study: *Microarray Data*

- **DNA Microarray technology**
  - Many thousands of DNA samples are arrayed on a glass slide, and competitively hybridized
  - Simultaneously monitor the expression levels of thousands of genes
  - Enormous opportunities to identify target genes for drug development and cancer classification
A Hybridized Mouse Microarray

Courtesy of Dr. James Hoying at Arizona Research Laboratories
Issues and Approaches

• Main challenges in establishing a public microarray gene expr database [Blazma 1999]
  - Annotation of experiments in a standardized and machine-readable way
  - Normalization procedures and standards enabling the comparison of data from different experiments and platforms

• Consensus emerged [MGED and MIAME, etc.]
  - Standards for storing and annotating data can be developed based on the XML technology
Markup Languages and Repositories

- To store data about *patterns, profiles and hybridization* for gene expression analysis
- MAML (MicroArray Markup Language)
  - MGED group ([www.mged.org](http://www.mged.org))
- GEML (Gene Expression Markup Language)
  - Rosetta Inphamatics ([www.geml.org](http://www.geml.org))
- Public repositories
  - GeneX (National Center for Genome Resources)
  - Gene Expression Omnibus (National Center for Biotechnology Information)
  - ArrayExpress (European Bioinformatics Institute)
Loading Gene Expr Data

By courtesy of ROSETTA Inphamatics
GEML Doc Type Definitions

- GEML defines two Document Type Definitions: GEMLPattern.dtd and GEMLProfile.dtd.
  - Pattern DTD describes genes, reporters, and chip layout.
  - Profile DTD describes gene expression data, cell, treatment, and hybridization information.
Road Map

• Introduction: Basic understanding of XML
• Case Study: Microarray gene expr data
• Challenges and Proposed Solutions
  ▪ XML to RDB Mapping
  ▪ Indexing XML data for Path Queries
    • XISS/Numbering Scheme and Path Joins
  ▪ Selectivity Estimation
• Concluding Remarks
Challenges in XML Research

• Storing and Publishing XML
  ▪ Use Relational DBMS to store XML data?
  ▪ Publish relational data in XML format for exchange
  ▪ Store and access Versioned XML data

• Indexing and Querying XML
  ▪ Fast access to XML data via path expressions
    • XPath and XQuery [W3C]
  ▪ XML query optimization, selectivity estimation
Work in Progress

- RDB mapping and XML publication
  - Edge table [INRIA 1999]
  - Inlining and Outer union [VLDB’99 & VLDB’00]
  - XRel path-based storage [TOIT’01]
  - Cost-based storage mapping [ICDE’02]

- Indexing for Path Queries
  - Index Fabric: Block-structured Patricia [VLDB’01]
  - XISS and Path Joins [Li & Moon, VLDB’01]

- Selectivity estimation
  - Path tree and Markov table [Wisc, VLDB’01]
  - Position histogram [Michigan, EDBT’02]

- XML versioning
  - UBCC clustering [UCLA & UCR, VLDB’01, EDBT’02]
XML to RDB Mapping

• **Edge Table [INRIA 1999]**
  - Map each `<parent, child>` pair to a tuple in `Edge(source_id, ordinal, name, flag, target)`
  - **Disadvantages**
    - Storage overhead due to redundant data
    - Too many self-joins are required

• **DTD Graph and Inlining [VLDB’99]**
  - Build a DTD graph from a simplified DTD
  - DFS traversal to create (potentially too many) relational tables
  - Avoid fragmentation problem by inlining descendants of an element into a single relation
XML to RDB Mapping

- **XRel: path-based storage** [TOIT’01]
  - Path Table stores distinct (root-to-any) paths
  - Element Table associates each path with a region where the instances of a path appear
- **Lego: cost-based mapping** [ICDE’02]
  - Transformations at the XML Schema level
  - Apply a greedy heuristic to find an efficient one
    - Feed a relational query optimizer with a relational schema and statistics
    - Compute the expected cost of computing queries
Road Map

- Introduction: Basic understanding of XML
- Case Study: Microarray gene expr data
- Challenges and Proposed Solutions
  - XML to RDB Mapping
  - Indexing XML data for Path Queries
    - XML to RDB Mapping
    - XISS/Numbering Scheme and Path Joins
    - Selectivity Estimation
- Concluding Remarks
Query by Path Expressions

- Path expressions to navigate through and retrieve XML data
  - Q1: `/chapter//figure[@caption="Tree Frogs"]`
  - Q2: `(E_1/E_2)^+ /E_3/((E_4[@A=v])|(E_5//E_6))`
    - Kleene-closure is not in XQuery 1.0.

- Common Feature of XML Query Languages
  - Lorel, XML-QL, XML-GL, XPath, etc.
  - XQuery working draft [W3C; Feb. 2001]
Indexing for Path Queries

- **Index Fabric [VLDB’01]**
  - All (root-to-leaf) paths are encoded as strings and inserted into a Patricia
  - Maintain the Patricia as a block-structured tree like B+-tree; High fan-out can be achieved
  - A sub-path match query is posed as a path query with a wildcard as a prefix

- **Disadvantages?**
  - A sub-path match query is posed a path query with a wildcard as a prefix
    - It could be slow without suffixes of paths in the index
  - No guarantee of the minimum storage utilization
Example: 3-level index fabric

RightOrder Inc., 2001
Road Map

- Introduction: Basic understanding of XML
- Case Study: Microarray gene expr data
- Challenges and Proposed Solutions
  - XML to RDB Mapping
  - Indexing XML data for Path Queries
    - XISS/Numbering Scheme and Path Joins
  - Selectivity Estimation
- Concluding Remarks
XISS and Path Joins [VLDB’01]

- Design a numbering scheme
  - Based on *Extended Preorder*
  - Determine ancestor-descendant relationship

- Propose *Path-Join* algorithms
  - Conventional tree traversal is slow
  - Join algorithms to avoid tree traversal

- Design indexing and storage structures
  - *XISS*: Element index, attribute index, structure index, etc.
Numbering Scheme

- XML objects are modeled by a tree structure
  - Nodes are XML elements and attributes
  - Parent-child represents nesting between objects
- To process path expression queries
  - (e.g.) chapter3/section, chapter3//figure
  - Conventional approach: traverse XML trees
  - New Approach:
    - Collect two object sets
    - Determine A-D relationship between objects
Extended Preorder

• **Annotate a node with a pair of** \(<\text{order, size}>\)
  - **For** \(Y\) **and its parent** \(X\),
    - **order**\((X) < \text{order}(Y)\) and
    - **order**\((Y) + \text{size}(Y) \leq \text{order}(X) + \text{size}(X)\)
  - **For sibling** \(X\) **and** \(Y\), **if** \(X\) **is before** \(Y\) **in preorder,**
    - **order**\((X) + \text{size}(X) < \text{order}(Y)\)

• **Lemma:** \(X\) **is an ancestor of** \(Y\) **iff**
  \[
  \text{order}(X) < \text{order}(Y) \leq \text{order}(X) + \text{size}(X)
  \]

• **Size can be pre-allocated to accommodate future insertion.**
Extended Preorder: Examples

- $(1,100)$ is an ancestor of $(17,5)$
  - $1 < 17$
  - $17 < 1+100$

- $(10,30)$ is not an ancestor of $(45,5)$
  - $45 > 10+30$
Index and Data Organization

• Operations to support
  ▪ For a name string, find a list of elements (or attributes) having the same name string.
  ▪ For a given object, find its parent and children.

• Two supplementary structures
  ▪ Name index maps a name string to nid
    • Minimize storage and computational overhead
    • Implemented as a B⁺-tree
  ▪ Value table stores all string values
XISS Structure Overview

XML Raw Data

Document Loader

Query Processor

Query

Result

XISS

Element Index

Attribute Index

Structure Index

Name Index

Value Table

Paged File
Element and Attribute Index

• Element Index maps \textit{nid} to a list of element records grouped by \textit{did} (document id).
  - Implemented as a \textit{B}⁺-tree
  - An element record is fixed-length containing:
    • <\textit{order}, \textit{size}>, depth, parent id

• Quickly find all elements having the same name string.

• Attribute Index is identical to E. index except
  • \textit{vid} (value id) to attribute value in the Value Table.
Structure Index

- Structure Index maps **did** to a list of element and attribute records.
  - Implemented as a B⁺-tree
  - Each record is fixed-length containing:
    - nid, <order,size>, parent-order, first-child-order, first-attribute-order, etc.
- Quickly locate an object record using **did** and **nid**
Structure Index: illustration

Document ID (did)

B⁺-tree

Array of All Elements and Attributes in the Same Document

nid,
<Order, Size>,
Parent order,
Child order,
Sibling order,
Attribute order...

Path Expression Queries

Q1: document("*.xml")//chapter//figure[@caption="Tree Frogs"]

- Conventional Approaches [McHugh, VLDB’99]
  - **Top-Down Traversal**
    - Check all downward paths from `chapter`
  - **Bottom-Up Traversal**
    - Check all upward paths from `figure` with `@caption="Tree Frogs"`
  - **Hybrid**
    - Meeting in the middle of a path
Path-Join Algorithms

- **Apply Path-Join algorithms to a sub-expression**
  - A single element or attribute
    - Use the element index or attribute index
  - A pair of element and attribute
    - Use *EA-Join* algorithm
  - A pair of two elements
    - Use *EE-Join* algorithm
  - A Kleene closure (+,*) of a subexpression
    - Use *KC-Join* algorithm
  - A Union of two subexpressions
    - Merge two intermediate results and group by did
Path-Join Algorithms

- Decompose a path expression
  - Q2: \((E_1/E_2)^*/E_3/((E_4[@A=v])|(E_5//E_6))\)
EA-Join Algorithm

- Join an element set and an attribute set by A-D
  - (e.g.) figure[@caption="Tree Frogs"]
- Input
  - {... E_i ...}, E_i is a set of elements from a document did
  - {... A_j ...}, A_j is a set of attributes from a document did
- Output
  - A set of (e,a) pairs such that e is parent of a
- Algorithm
  
  foreach $E_i$ and $A_j$ with the same did do
  
  foreach $e \in E_i$ and $a \in A_j$ do
  
  if (e is parent of a) then output (e,a);
EA-Join: single-scan requirement

- Attribute nodes **before** their sibling elements in the order by the numbering scheme.

```
chapter <1, 3>  
  chapter <2, 1>  
    name <3, 0>  
    name <4, 0>  
  name <2, 0>  
  chapter <3, 1>  
```

---- Element Node
------ Attribute Node
EE-Join Algorithm

- Join two element sets by A-D relationship
  - (e.g.) chapter/figure
- Input
  - \{.. E_i ..\} and \{.. F_j ..\}, E_i, F_j are a set of elements from the same document did
- Output
  - A set of (e,f) pairs such that e is an ancestor of f
- Algorithm
  
  foreach $E_i$ and $F_j$ with the same did do
  
  foreach $e \in E_i$ and $f \in F_j$ do
  
  if (e is ancestor of f) then output (e,f);
EE-Join: Multiple Scans

• Use **depth** to process fixed-length path expression queries
  - (e.g.) chapter/*//*/figure

• Join by A-D relationship
  - Viewed as a join of a range set and a point set
    - Range: [order(chapter), order(chapter)+size(chapter)]
    - Point: order(figure)
  - Multiple scans are unavoidable
  - Still, very efficient for long or unknown-length paths
EE-Join: an extreme case

- chapter <1, 90>
- chapter <2, 80>
- chapter <8, 20>
- chapter <9, 10>
- figure <10, 0>
- figure <11, 0>
- figure <19, 0>
KC-Join Algorithm

- (e.g.) chapter*, figure+, chapter/chapter
- **Input**
  - {.. E_i ..}, E_i is a set of elements from a document **did**
- **Output**
  - A Kleene closure of {.. E_i ..}
- **Algorithm**
  
  ```
  i = 1; K_i = {.. E_i ..};
  repeat
    i = i+1; K_i = EE-Join(K_{i-1}, K_1);
  until (K_i is empty)
  output union of K_1, K_2, .., K_{i-1};
  ```
Experiment Settings

- **XISS prototype system using C++ and GiST**
- **Sun Ultrasparc-II running Solaris 2.7**
  - 256 MB Memory, 20 GB Disk w/ Ultra 10 EIDE
- **Data sets**
  - **Real-world: Shakespeare’s Plays, SIGMOD**
    - Shakespeare’s Plays: 327K(22) elements, 0(0) attributes
    - SIGMOD Record: 839K(47) elements, 4775(3) attributes
    - DBLP/conference: 2666K(29) elements, 199K(3) attributes
  - **Synthetic: XML generator (IBM)**
    - NITF (News Industry Text Format) as a DTD
      - NITF100: 63K(124) elements, 263K(142) attributes
      - NITF1: 38K(86) elements, 171K(106) attributes
EE-Join Results

- Measured total elapsed time; IO time was dominant.
- An order of magnitude faster than bottom-up approaches.
EE-Join: Speed-up

Data Sets

- Shakespeare
- SIGMOD
- NITF-100
- NITF-1
- DBLP

Speedup

EA/EB
EA/*/EB
EA/*/EB
EA/EB

Invited Talk for DEWS’2002, March 4-6, 2002, Kurashiki, Japan
EE-Join: Scale-up

- NITF data (229MB)
EA-Join Results

- Measured total elapsed time; IO time was dominant.
- SIGMOD data has a small number of attributes; bottom-up was slightly better than EA-Join.
- NTIF has much more attributes than elements.
EA-Join: Speed-up

Data Sets

- SIGMOD
- NITF-100
- NITF-1
- DBLP

Y-axis: Speedup

Top-down vs. Bottom-up
EA-Join: Scale-up

- NITF data (229MB)
Road Map

• Introduction: Basic understanding of XML
• Case Study: Microarray gene expr data
• Challenges and Proposed Solutions
  ▪ XML to RDB Mapping
  ▪ Indexing XML data for Path Queries
    • XISS/Numbering Scheme and Path Joins
  ▪ Selectivity Estimation
• Concluding Remarks
Selectivity Estimation

- **Path Tree [VLDB’01]**
  - Path Tree is a condensed form of XML data tree
    - Each node labeled with a tag name and a frequency
    - Use *-nodes to coalesce nodes and frequencies
  - To estimate, take the frequency sum, or average if there is a *-node in the path.

![XML data tree](image1)

![Path tree](image2)
Selectivity Estimation

- **Markov Table [VLDB’01]**
  - Store frequencies of paths no longer than $M$
  - Use *short memory* assumption to estimate frequencies for paths longer than $M$

- $\text{Freq}(A/B/C/D) = \text{freq}(A/B/C) \cdot \Pr(D \mid B/C)$
  - $\text{freq}(A/B/C) \cdot \frac{\text{freq}(B/C/D)}{\text{freq}(B/C)}$

- $\text{freq}(A/B/C) = \frac{\text{freq}(A/B)}{\text{freq}(B) \cdot \text{freq}(C/B)} = 9$

- $\text{freq}(A/B/C/D) = \frac{\text{freq}(B/C) \cdot \text{freq}(C/D)}{\text{freq}(B) \cdot \text{freq}(C)} = 4.8$
Selectivity Estimation

- Positional Histogram [EDBT’02]
  - Query: u[P₁]/v[P₂]
  - Associate an extended preorder with each node
    - Each node mapped into a point in a 2-dim space
    - Intervals either separate or fully contained
    - Ancestors in the upper-left region, and descendants in the lower-right region
  - To estimate the selectivity of u[P₁]/v[P₂]
    - Estimate the number of ancestors (or descendants) for each instance of v (or u) using the 2-dim histogram
Concluding Remarks

• **XML for a wide spectrum of applications**
  - Bioinformatics, Multimedia (Mpeg), Chemistry
  - Literature, Patent/Financial/Legal, Metadata Exchange
  - *The list of applications keeps increasing!!*
  - See *XML Cover Page* for more applications
    - CML (Chemical Markup Language)
    - HDML (Hand-held Device Markup Language)
    - WIDL (Web Interface Definition Language)

• **XML poses new challenges**
  - Relational vs. Semi-structured: Mapping required
  - Indexing and Querying XML data
  - XML query optimization and selectivity estimation
Still Going On and On...

- **ACM SIGMOD’2002**
  - 10 out of 42 research papers related to XML data
    - Indexing for path queries, joins, storing in RDB, semi-structured, statistics, etc.
  - 1 industrial paper and 4 demos

- **At the U of Arizona**
  - Optimization of Path Joins with Long paths
  - Comparative performance study (e.g., Index Fabric)
  - Implementation of XISS and Path Joins on RDBMS
  - Launch the MDR (Microarray Data Repository) project in collaboration with researchers at AHSC
Questions

For more information,

www.cs.arizona.edu/~bkmoon

THANK YOU!