XANADUE: A System for Detecting Changes to XML Data in Tree-Unaware Relational Databases

Erwin Leonardi Sourav S. Bhowmick
Singapore-MIT Alliance, Nanyang Technological University, Singapore
School of Computer Engineering, Nanyang Technological University, Singapore
{lerwin,assourav}@ntu.edu.sg

ABSTRACT

Recently, a number of main memory algorithms for detecting the changes to XML data have been proposed. These approaches are not suitable for detecting changes to large XML document as it requires a lot of memory to keep the two versions of XML documents in the memory. We have developed a novel XML change detection system, called XANADUE that uses traditional relational database engines for detecting changes to large XML data. In this approach, we store the XML documents in the relational database and issue SQL queries (whenever appropriate) to detect the changes. This demonstration will showcase the functionality of our system and the effectiveness of XML change detection in relational environment.

Categories and Subject Descriptors: H.2.4 [Database Management]: Systems – *Relational databases*.

General Terms: Algorithms, Design, Experimentation.

Keywords: XML, change detection, relational databases.

1. INTRODUCTION

Detecting changes to XML data is an important research problem. Recently, a number of techniques for detecting the changes to XML data have been proposed. XyDiff [1] and X-Diff [7] are main-memory algorithms for detecting the changes in *ordered* and *unordered* XML documents, respectively. X-Diff detects minimum-cost edit script and has quadratic CPU cost and memory usage. XyDiff, on the other hand, has a linear CPU cost and memory usage but its result quality is relatively worse than X-Diff.

Our study showed that the above main-memory algorithms for XML change detection suffer from the scalability problem as they fail to detect changes to large XML documents due to lack of memory. This demonstration features our XML change detection system XANADUE (**X**ML Ch**a**nge Ma**n**agement in Rel**a**tional **D**atabases having Tree-**U**naware K**e**rnel), to address the scalability issue by detecting changes using the relational database system. The XANADUE project is an investigation of how far we can push the idea of using mature RDBMS technology to design and build a full-fledged XML change management system, without invading the database kernel to make it "tree aware". It is based on our recent work on XML change detection in relational environment as described in [2, 3, 4, 5]. To the best of our knowledge, XANADUE is the first system to detect changes to XML data using relational backends.

SIGMOD'07, June 11–14, 2007, Beijing, China.
ACM 978-1-59593-686-8/07/0006.

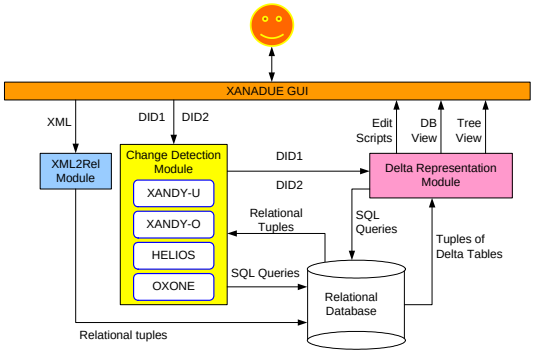

Figure 1: System Architecture of XANADUE.

XANADUE employs the following strategy for change detection. Given the old and new versions of an XML document, it first shreds both documents into sets of relational tuples and stores them in the relational database. Next, a set of SQL queries (wherever appropriate) is issued to detect the changes. This demonstration will show that this line of research has several important benefits and worth to be followed.

2. OVERVIEW AND ARCHITECTURE

Figure 1 depicts the architecture of XANADUE. The system adopts Browser/Server architecture. The browser (the XANADUE *GUI* module) allows user to select two versions of XML documents, detect changes, and visually see various types of changes on these documents. The server consists of three modules: the *XML2Rel* module, the *Change Detection* module, and the *Delta Representation* module. We now discuss these four modules in more detail.

The XML2Rel Module: This component parses two versions of XML documents and loads them into tuples of relational tables in a standard commercial DBMS (the current version of XANADUE uses Microsoft SQL Server 2005). Recent research on relational support for XML data can be classified into two representative types. In a *tree-unaware* approach, the database kernel is not modified to support XML processing. On the other hand, in a *tree-aware* approach, the database engine is invaded to teach it that it is operating on tree-shaped data rather than any arbitrary data points. The current version of XANADUE is built based on "tree-unaware" strategy.

Note that there are two major approaches for storing XML documents in a "tree-unaware" relational database.

In *schema-conscious approach*, a relational schema is created based on the DTD/schema of the XML documents. In the *schema-oblivious approach*, a fixed schema is maintained which is used to store XML documents. This approach does not require existence of an XML schema/DTD. As the DTD/schema of XML documents may not always be available, we support both approaches in XANADUE.

The Change Detection Module: Once the two versions of a document is stored in the relational database, we are ready to detect changes. The change detection module supports the following four submodules for detecting changes to ordered and unordered XML documents based on schema-oblivious or schema-conscious storage strategy.

- Modules XANDY-O [2] and XANDY-U [3] are used for detecting changes to ordered and unordered XML documents, respectively, using schema-oblivious storage approach.
- Modules OXONE [5] and HELIOS[4] are used for detecting changes to ordered and unordered XML documents, respectively, using schema-conscious storage.

In all of these four modules, the algorithm for change detection consists of two phases: the *best matching subtrees computation* phase and the *change detection* phase. We briefly describe these phases now. The reader may refer to [2, 3, 4, 5] for complete details.

The Best Matching Subtrees Computation Phase: Given two versions of XML documents, T_1 and T_2, stored in a relational database, the objective of this phase is to find the most similar subtrees (hereafter called *best matching subtrees*) in T_1 and T_2. We match the subtrees in T_1 to the ones in T_2. Note that a subtree in T_1 can be matched to more than one subtrees in T_2 and vice versa. We measure the similarity of each matching subtrees by calculating the *similarity score* of these matching subtrees using a set of SQL queries. The most similar subtrees are called *best matching subtrees*.

The process of finding the best matching subtrees is done in a bottom-up fashion. We find the *best matching configurations* at every level of the tree such that the similarity scores of the parent nodes are maximized. The problem of finding best matching configurations is similar to the problem of finding *maximum weighted bipartite matching*. We use a series of SQL queries and a modified Hungarian method [6] to determine best matching subtrees.

Change Detection Phase: In this phase, we use the information on best matching subtrees in order to detect different types of changes by issuing SQL queries (wherever appropriate). First, we determine the changes on internal nodes (both insertions and deletions). Next, the inserted and deleted leaf nodes are detected. Finally, we detect updated leaf nodes and moved nodes (for ordered XML only). The different types of changes are stored in a set of *delta tables*.

Delta Representation Module: Upon successful detection of changes, this module takes the delta tables as input and generates the minimum-cost edit script. It also facilitates meaningful presentation of the changes in two modes: the *tree* view and the *table* view. In the *tree* view, the system will merge T_1 and T_2 and add annotations to the merged tree. The annotations will help the users to know whether a particular node is changed and the change type. Different types of changes will then be displayed with different color codes in the XANADUE GUI (discussed below). In the *table* view, users are able to see the tuples of the delta tables.

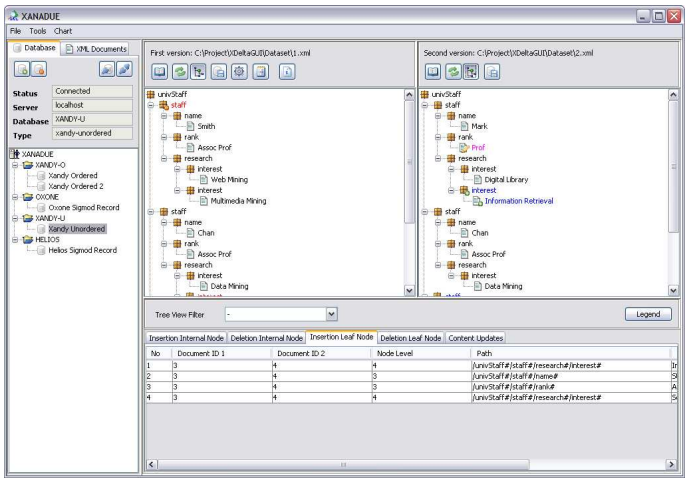

Figure 2: XANADUE GUI.

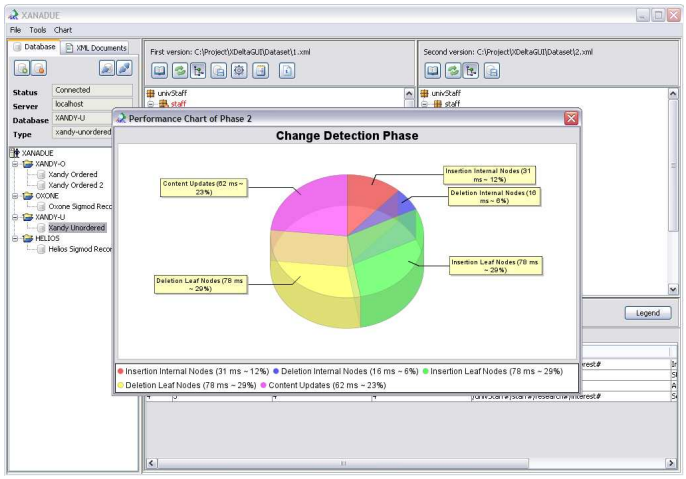

Figure 3: Performance visualization.

The XANADUE GUI Module: Figure 2[1] depicts the screen dump of the current version of XANADUE visual interface. It consists of three panels. The left panel displays change detection modules for ordered and unordered XML based on schema-conscious and schema-oblivious storage approaches. The user chooses one of these modules for detecting changes. The top panel displays the side-by-side tree representations of two versions of an XML document. The changes to these versions are shown in different colors. The bottom panel displays the table view of different types of changes. There is a one-to-one correspondence between a tuple in the table view and a changed node in the tree view. User can right click on a tuple in the table view to view the corresponding changed node in the tree view. User can also select only a specific type of changes (e.g., insertion or deletion) to view only those types of changes in the tree view. Finally, we also provide visualization of the performance of various phases in the change detection process using pie charts (Figure 3).

[1]Note to readers: We recommend viewing the screenshots presented directly from the color PDF, or from a color print copy, since the grayscale version may not clearly register the various features.

3. SCALABILITY AND PERFORMANCE

XANADUE is entirely implemented in Java on top of a commercial RDBMS. Our implementation is meant to assess the viability of using relational back-ends for XML change detection. RDBMSs are known to scale well with increasing data volumes – an inevitable feature of XML change detection systems, if they are to support large XML documents.

For our experimental study, we use Microsoft SQL Server 2000 for storing shredded XML documents. The experiments were conducted on a Microsoft Windows XP Professional machine having Pentium 4 1.7 GHz processor with 512 MB of memory. We used a set of synthetic XML data based on SIGMOD DTD (http://www.sigmod.org/record/). The numbers of nodes are varied from 331 nodes to 620,223 nodes. The second versions of the XML documents are generated by using our XML change generator. We compared the performance of various submodules of XANADUE to the Java version of X-Diff [7] (downloaded from http://www.cs.wisc.edu/~yuanwang/xdiff.html), and C version of XyDiff [1] (downloaded from http://pauillac.inria.fr/cdrom/www/xydiff/index-eng.htm). Note that despite our best efforts (including contacting the authors), we could not get the Java version of XyDiff. The C version of XyDiff was run in a Pentium 4 1.7 GHz processor with 512 MB of memory with Red Hat Linux 9 operating system.

Figures 4(a) and (b) depict the performance of our approaches compared to memory-based approaches in detecting the changes to ordered and unordered XML documents, respectively. The percentage of changes is set to 3%. We observe that XANADUE is more scalable than X-Diff and XyDiff. X-Diff is unable to detect the changes to XML documents that have more than 6,000 nodes due to lack of main memory. When we run the C version of XyDiff implementation, it cannot detect the changes to XML documents that have more than 355,000 nodes as its process was killed by Linux kernel. Note that the performance of XANADUE is slower than main memory-based approaches for smaller data sets as the database I/O cost is more expensive. However, for larger datasets, HELIOS and XANDY-U are orders of magnitude faster than X-Diff. Also, OXONE has comparable response time with XyDiff for large XML documents. Finally, schema-conscious approaches (HELIOS and OXONE) perform better than schema-oblivious approaches (XANDY-U and XANDY-O) in XANADUE.

Let us now observe the *result quality* of XANADUE for various percentages of changes. The *result quality* is defined as the ratio of the number of edit operations in a delta detected by a particular approach to the number of edit operations in the optimal delta. Figure 4(c) shows that XANADUE is able to produce superior deltas compared to XyDiff. Figure 4(d) depicts the result quality comparison between XANADUE and X-Diff. When the percentage of change is large, XANADUE produces better quality of delta. For other cases, we observe that the result quality of XANADUE and X-Diff is comparable.

Figures 4(e) and 4(f) show the effects of percentage of changes to the performances of XML change detection. We use data sets containing 890 and 2718 nodes for ordered and unordered XML change detection, respectively. The percentages of changes are equally distributed to different types of changes. Overall, percentage of changes slightly affects the performance of all change detection approaches.

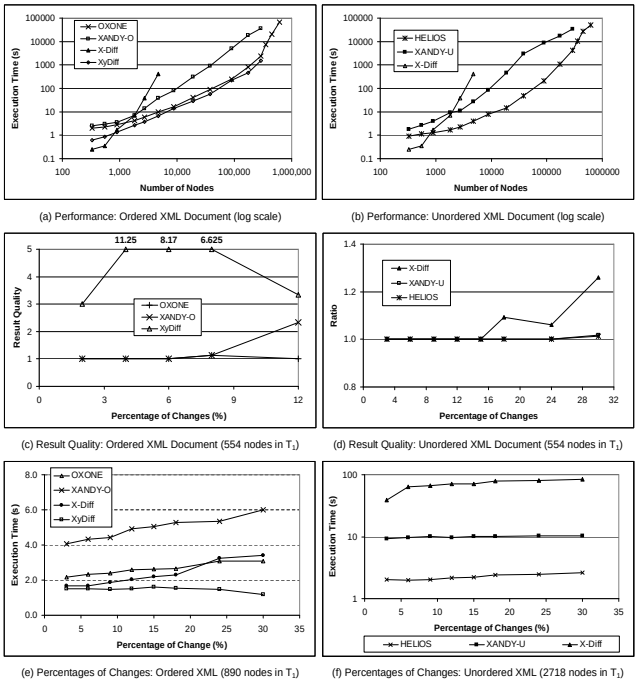

Figure 4: Performance results.

4. DEMONSTRATION

Our demonstration aims to showcase the functionality and effectiveness of the XANADUE system in detecting changes to XML documents. We will showcase the followings.

- How relational backend is used to detect different types of changes to ordered and unordered XML in XANADUE. We will show how this process is simplified by the XANADUE GUI.
- Demonstrate the cases when the result quality of XANADUE is comparable to X-Diff and cases when it is better than X-Diff and XyDiff.
- Demonstrate better scalability of XANADUE compared to main memory algorithms (say X-Diff). We will show when HELIOS and XANDY-O are significantly faster than X-Diff. We will also show cases where X-Diff fails to detect changes due to lack of memory but XANADUE is able to detect these changes.

5. REFERENCES

[1] G. COBENA, S. ABITEBOUL, A. MARIAN. Detecting Changes in XML Documents. *In ICDE*, 2002.

[2] E. LEONARDI, S. S. BHOWMICK. XANDY: A Scalable Change Detection Technique for Ordered XML Documents Using Relational Databases. *DKE Journal*, 59(2), Elsevier Science, 2006.

[3] E. LEONARDI, S. S. BHOWMICK, S. MADRIA. XANDY: Detecting Changes on Large Unordered XML Documents Using Relational Databases. *In DASFAA*, 2005.

[4] E. LEONARDI, S. S. BHOWMICK. Detecting Changes on Unordered XML Documents Using Relational Databases: A Schema-Conscious Approach. *In CIKM*, 2005.

[5] E. LEONARDI, S. S. BHOWMICK. OXONE: A Scalable Solution for Detecting Superior Quality Deltas on Ordered Large XML Documents. *In ER*, 2006.

[6] C. PAPADIMITRIOU, K. STEIGLITZ. Combinatorial Optimization: Algorithms and Complexity. Prentice-Hall, Englewood Cliffs, NJ, 1982.

[7] Y. WANG, D. J. DEWITT, J. CAI. X-Diff: An Effective Change Detection Algorithm for XML Documents. *In ICDE*, 2003.