Integrating Network-Bound XML Data

Zachary G. Ives* Alon Y. Halevy Daniel S. Weld

Abstract

Although XML was originally envisioned as a replacement for HTML on the web, to this point it has instead been used primarily as a format for on-demand interchange of data between applications and enterprises. The web is rather sparsely populated with static XML documents, but nearly every data management application today can export XML data. There is great interest in integrating such exported data across applications and administrative boundaries, and as a result, efficient techniques for integrating XML data across local- and wide-area networks are an important research focus.

In this paper, we provide an overview of the Tukwila data integration system, which is based on the first XML query engine designed specifically for processing network-bound XML data sources. In contrast to previous approaches, which must read, parse, and often store XML data before querying it, the Tukwila XML engine can return query results even as the data is streaming into the system. Tukwila features a new system architecture that extends relational query processing techniques, such as pipelining and adaptive query processing, into the XML realm. We compare the focus of the Tukwila project to that of other XML research systems, and then we present our system architecture and novel query operators, such as the x-scan *operator. We conclude with a description of our current research directions in extending XML-based adaptive query processing.*

1 Introduction

The original vision of XML as the data description format that would reshape the web has, to this point, not materialized as expected. HTML documents and dynamic HTML pages predominate, whereas XML documents are rare. Yet behind the scenes, on the Internet and intranets, XML has in fact been widely deployed. Corporations are interchanging data between applications with XML as the common format. Business coalitions and partners are exchanging data and transactions via XML, using standardized DTDs such as those published at OASIS and BizTalk[1]; and Microsoft's .NET "web service" architecture is also based on XML data interchange. Currently, XML has been most successful not as a format for "materialized" or document data, but as a "wire protocol" for exchanging data through virtual XML views.

We are convinced, therefore, that the areas of interoperability (both in terms of schema and data) and data integration are the most significant concerns in XML data management. XML storage and indexing, as well as query processing techniques for local XML repositories, address interesting and relevant problems — yet XML's greatest potential may be as a medium for integrating and exchanging data. Thus we need mechanisms for efficiently processing non-materialized, un-indexed, streaming XML data that is being sent across a network.

Bulletin of the IEEE Computer Society Technical Committee on Data Engineering

*Supported in part by an IBM Research Fellowship..

[1] Found at www.xml.org and www.biztalk.org.

	Tukwila	Niagara	Xyleme	Lore	XFilter	eXcelon/ Tamino	Silkroute/ XPERANTO
Live Source Data	√			Repository		Repository	Relational
Index of Remote Data		√			√		
Warehoused Data			√				
Query Subscription		√	√	√			
Larger than RAM	√				√	√	√
Native XML	√	√	√	√	√	√	Relational

Table 1: Characterization of XML query processing space and systems

The primary focus of the Tukwila data integration system is to develop new techniques for processing dynamically-generated, streaming XML data. Our emphasis is on developing a query processor that operates on queryable XML data sources, requesting the desired data and efficiently modifying, combining, and restructuring it. This data may potentially be larger than memory, so Tukwila also emphasizes support for out-of-core execution.

This focus on large, "live" queryable sources is in contrast to the other XML query processing projects and systems of which we are aware; see Table 1 for a comparison of system features. Niagara [NDM+01] and Xyleme [Aea01] focus on providing query capabilities for XML documents or data files that are locally indexed or warehoused, respectively; both provide support for "subscription" queries whose results update as the underlying data is modified. Lore [GMW99] is a semistructured data repository with XML extensions, and eXcelon [XLN] and Tamino [Tam] are native XML repositories. Silkroute [FTS99, FMS01] and XPERANTO [CFI+00] provide XML publishing capabilities for relational data. XFilter [AF00] (and the associated DBIS system) is not precisely a query processor, but rather an information dissemination system whose goal is to take a set of XPath queries expressing a user's interests, and to "push" or disseminate all documents matching these queries to the user.

The Tukwila project originated as a relational-model data integration system with a new adaptive architecture to improve query performance [IFF+99]. In our original work, we identified a number of desiderata for a data integration system. Data integration queries are in many cases ad-hoc and interactive (e.g. combining data from several sources to produce a browsable online catalog), so early answers, and thus **pipelining** of results, is desirable. Second, as data is transferred from a source to the integration engine, it is subject to delays and burstiness; hence operators must support **flexible scheduling** so the query processor can process other available tuples while waiting for delayed sources. Finally, since statistics are often incomplete or even nonexistent for remote, dynamically generated data, the query processor should support **runtime re-optimization and re-scheduling** of a query, so it can adjust a query plan as it acquires better knowledge about the data sources. These desiderata are design goals of *adaptive query processing*, and considerable work has recently been done in this area, including query scrambling [UFA98], mid-query re-optimization [KD98], ripple [HH99] and pipelined hash [WA91, IFF+99, UF00] joins, eddies [AH00], dynamic scheduling of pipelines [UF01], and streaming of partial results through blocking operators [STD+00]. (For more details, see the June 2000 issue of the *IEEE Data Engineering Bulletin*.)

In the context of integrating XML data, the data model has changed, but the desiderata remain the same. In fact, XML query languages such as XML-QL and XQuery include operations supported by SQL, such as group-by and join — hence even the same "adaptive" relational techniques should almost directly apply. Yet the XML query model does not operate on tuples in tables — instead, it works on *combinations of input bindings*: patterns are matched across the input document, and each pattern-match binds an input tree to a variable. The query processor iterates through all possible combinations of assignments of input bindings, and the query operators are evaluated against each successive combination. At first glance, this seems quite different from relational-style execution, but a closer examination reveals little difference: if we assign a column in a tuple to each variable, we can view each legal combination of variable assignments as forming a tuple of binding values. Note, however, that this "binding tuple" will likely contain *trees* (or graphs) in its columns, rather than scalar values. Moreover, the content of one column might actually be contained within the tree value of another column.

As a result of this observation, the Tukwila architecture, like a relational data integration system, is based on a tuple-oriented query processing model — but it includes extensions to the tree data type, the ability to incrementally generate binding tuples from an incoming XML stream, and a set of additional XML-specific operations that have no correspondence in a relational system. Our architecture leverages many useful techniques developed for relational data management, while offering the flexibility required to query XML and providing better performance than previous XML query processing approaches.

In the next section, we describe the Tukwila architecture in more detail and present a subset of the experimental results demonstrating Tukwila's impressive performance. We conclude in Section 3 with a discussion of our current research directions in XML and adaptive query processing.

2 The Tukwila System

The Tukwila system supports the majority of features in the XQuery language under development by the World Wide Web Consortium. XQuery includes a tree-structured data model, support for arbitrary recursive functions, conditional queries, and strong typing. Tukwila does not yet attempt to implement the complex sub-typing capabilities of XQuery (which are still under heavy development), and support for recursive functions and conditionals is still under construction (a challenge here is the inherent difficulty of optimizing these queries).

A query is fed into Tukwila's optimizer, which is designed along the lines of Starburst [HFLP89] and Volcano [GM93], with algebraic-level rewrites. The primary difference between Tukwila's optimizer and previous work is in supporting XML-specific operations such as nesting of structure, and in supporting new XQuery features such as arbitrary recursive functions. In our experiments, we have observed that statistics are even harder to obtain for XML data than they are for autonomous relational data sources, and hence it is difficult for any optimizer to generate optimum plans — later in this paper, we present our current focus in attempting to address this problem.

The Tukwila query execution engine features a number of innovations, but preserves several important components from the original relational-model Tukwila engine, including support for multithreaded I/O and operators, plus an event handler for error and exception conditions. In the remainder of this section, we briefly discuss the novel components that provide native XML support; more detail can be found in [IHW01].

2.1 Query Execution Components

The diagram in Figure 1 shows the XML components of the Tukwila system. XML input streams are fed into *x-scan* operators, which match the query's input path expressions against the data and output streams of binding tuples. These binding tuples contain assignments of XML elements (trees) to each of the query's input variables. Since the bound trees may be arbitrarily large and may overlap, binding tuples actually contain *references* rather than the full XML elements; the XML data values are stored separately in an XML Tree Manager that may be paged to disk. The binding tuples are fed into a tree of query operators (described in the next section), which filter, tag, and combine the tuples. Finally, the tagged tuples are passed to an XML Generator, which converts them back into the XML format and returns the result stream to the query initiator.

At first glance it may appear likely that, because of Tukwila's reliance on the Tree Manager, a large XML file could produce "thrashing" in the swap file during query processing, but we have experimental evidence [IHW01] that Tukwila avoids this problem, which we attribute to two factors. First, the system supports "inlining" of scalar values: string, integer, or other "small" data items can be embedded directly in the tuple, avoiding the

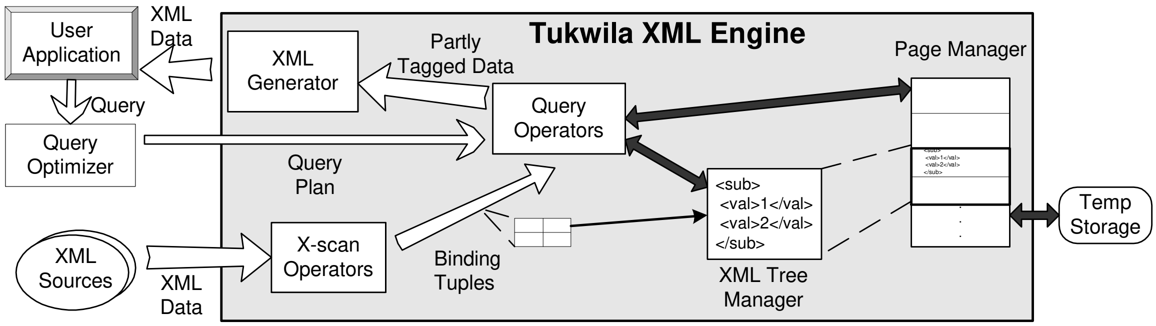


Figure 1: Architecture of the Tukwila query execution engine. After a query plan arrives from the optimizer, data is read from XML sources and converted by x-scan operators into output tuples of subtree bindings. The subtrees are stored within the Tree Manager (backed by a virtual page manager), and tuples contain references to these trees. Query operators combine binding tuples and add tagging information; these are fed into an XML Generator that returns an XML stream.

dereferencing operation. Typical query operations in XML-QL and XQuery are done on scalar rather than complex data (e.g., joining or sorting are frequently based on string values); thus these operations often only need data that has been inlined. Large, complex tree data is typically only required at the XML generation stage, when the final results are returned. A second factor is that many XML queries tend to access the input document in sequential order, and the Tree Manager therefore can avoid re-reading data that has been paged out. For purposes of comparison, we point out that a paged DOM-based approach would have similar behavior to our scheme (except that in-memory representation is larger in a DOM tree); a mapping from XML to relations ("shredded XML") typically requires a significant amount of materialization in the first place, and often incurs heavy costs whenever it needs to perform joins to recreate irregular structure.

2.2 Operations for Processing XML

As was previously suggested, Tukwila includes all of the standard relational database operator implementations, including selection, projection, grouping, sorting, and nested-loops, merge, hash, and double pipelined hash joins. These operators have all been extended to work with data in the Tree Manager, but are otherwise unremarkable. We now provide a brief sketch of the novel XML operators in Tukwila; please see [IHW01] for a more detailed description of each operator.

The novel operator most fundamental to Tukwila's operation, which enables the entire tuple-oriented query processing model, is the *x-scan* operator. XML-QL and XQuery path expressions are related to regular expressions over the XML structure, and the first phase of executing a query consists of matching the desired path expressions against the XML data: this is the operation x-scan performs, while the document is still being read and parsed. X-scan combines and returns the results as a stream of binding tuples. The core of an x-scan operator is a set of dependent finite state machines that match the query's path expressions. Whenever an element open-tag is parsed, this represents a forward-traversal in a state machine; whenever the close-element is encountered, the state machine's previous state is restored. An accept state signifies that a binding to the current XML element should be produced. Finally, separate variable bindings must be combined to produce the appropriate binding tuples.

The x-scan operator is much like an index scan or sequential scan in a relational database — it is given a file and retrieves the desired components. However, in a network-based context, where data from one source might describe a set of inputs to be provided to another query, another operator is very useful: one that takes a series of data values, combines them to produce a new request for data from a different source, and then performs an x-scan operation over the answers to that request. We call this operation *indirect-scan*, and it performs very

Nbr.	Class	Input	Query
Q1	Extract	0.9MB	Suras in Quran with “The” in title (large trees)
Q2	Extract	1.5MB	Chapters after Chapter 7 in Book of Mormon (medium trees)
Q3	Extract	1.5MB	Sura titles with “Mormon” from Book of Mormon (single result)
Q4	Extract	39MB	DBLP papers that reference conferences (select by existential)
Q5	Extract	9MB	DBLP papers with URLs – across wide area from `nbci.com`
Q6	Integrate	200KB x 9MB	Nest DBLP papers in their conference entries ($1 : n$ nesting)

Table 2: Queries for extraction (Q1-5) and combination (Q6) of XML

similarly to a dependent join in a relational database; it retrieves data, sends a request to a dependent source, and then combines the returned results with the current data. However, there are two key differences: (1) indirect-scan can request data from a different web source on each iteration, since it generates a complete request (and not simply an assignment of input parameter values) each time[2]; and (2) indirect-scan must perform an x-scan operation over the dependent data source before joining the results with those from the independent source.

The other novel XML operators in the Tukwila engine relate to creation of XML structural information. There are operators for enclosing data in new XML elements or attributes, and for outputting literal and computed values in the XML stream. We also include operators for nesting child subquery data within a parent query's output (a very common operation in XML query languages) and for separating attributes into hierarchical layers. All of these operators annotate the query processor's binding tuples with structural information that is not part of the data but controls the operation of the XML generator.

2.3 Performance

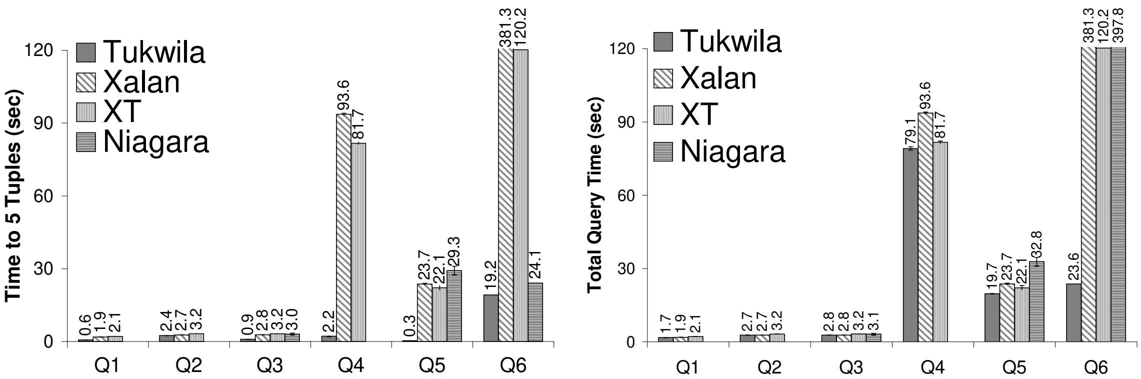
We have performed extensive experimental studies of Tukwila's performance and scalability, which can be found in [ILW00] and [IHW01]. Due to space limitations, in this paper we provide only a brief discussion of our findings, which are that the Tukwila architecture provides superior performance and scales well to the largest XML documents we could find on the Web.

Figure 2 compares the performance of Tukwila with that of the Niagara query processor[3] and with two XSLT query engines, XT and Xalan. We experimented with a range of queries, including selection-oriented document queries and integration queries combining data between multiple sources. The graphs show the running times for the queries of Table 2. The results demonstrate that Tukwila produces initial answers (2a) significantly earlier than the other query processors, largely due to its streaming input model that allows for pipelining. Moreover, Tukwila's overall query completion times (2b) were also faster, particularly if the query combined multiple input documents or restructured them.

Additional experiments have shown that the XSLT engines and the Niagara system do not scale to documents larger than about 25% of available memory, when they fail because the document's memory representation fills available RAM and they do not have mechanisms for overflow to disk. In contrast, the Tukwila engine only showed a small performance loss when querying the largest complex well-formed XML document we could find on the web, at 160MB, when given only 20MB of memory. For these results and others, please see the cited papers.

[2] This allows for greater expressiveness in queries, as one data source can describe a list of alternate sources to be combined in a single query.

[3] We used the version available at www.cs.wisc.edu/niagara during December, 2000.



(a) First 5 tuples (b) Total query time

Figure 2: Experimental comparison of XML queries shows that Tukwila has equally good total running time (and better time to first tuples) for simple extraction queries over small documents (Q1-Q3), and first tuples emerge significantly faster for larger documents both on the local area (Q4) and Internet (Q5). Tukwila performance is dramatically better than other systems when combining and restructuring XML data (Q6).

3 Conclusions and Future Directions

The query execution model of the Tukwila data integration system seems very well-suited to our target domain: potentially large, queryable XML data sources. The pipelined execution model and adaptive query processing operators produce good performance even under varying network conditions, and the approach also scales well.

We believe that we have found a good execution model for XML query processing, and now the next major challenge lies in effective *optimization* of XML queries. In our context, this is an extremely challenging problem: data sources may not have statistics, and in fact statistics and cost models for XML are not yet well-developed even for XML repositories. Worse, XML queries may include operations whose costs are difficult to model, such as recursive function calls, conditional queries, and references to external files.

It is our belief that the only real solution to this problem lies in further increasing the adaptive behavior of the query processing system, and in allowing the query processor to frequently re-optimize the query as it executes and gains increasing knowledge of costs and data characteristics. Our original Tukwila system provided mechanisms to re-optimize at query materialization points; later work by the authors of the Telegraph project [AH00] demonstrated that adjustments to a query plan can sometimes be made in the middle of a tuple stream, and their eddy operators performed flow-based tuple routing. We believe that these ideas must be carried even further to support efficient XML query processing: query execution and re-optimization should occur on a nearly continual basis, allowing the system to constantly adjust the query plan to varying conditions, to infer data patterns and possible optimizations, and even to share work between concurrent queries or to distribute a query across multiple nodes — while preserving prior work. It is this problem of integrated, feedback-directed query evaluation upon which we are now focusing.

References

[Aea01] Serge Abiteboul and et al. A dynamic warehouse for XML data of the web. *IEEE Data Engineering Bulletin*, June 2001.

[AF00] Mehmet Altinel and Michael J. Franklin. Efficient filtering of XML documents for selective dissemination of information. In *VLDB '00*, 2000.

[AH00] Ron Avnur and Joseph M. Hellerstein. Continuous query optimization. In *SIGMOD '00*, 2000.

[CFI+00] Michael J. Carey, Daniela Florescu, Zachary G. Ives, Ying Lu, Jayavel Shanmugasundaram, Eugene Shekita, and Subbu Subramanian. XPERANTO: Publishing object-relational data as XML. In *ACM SIGMOD WebDB Workshop '00*, 2000.

[FMS01] Mary F. Fernandez, Atsuyuki Morishima, and Dan Suciu. Efficient evaluation of XML middle-ware queries. In *SIGMOD '01*, May 2001.

[FTS99] Mary Fernandez, Weng-Chiew Tan, and Dan Suciu. SilkRoute: Trading between relations and XML. In *Ninth International World Wide Web Conference*, November 1999.

[GM93] Goetz Graefe and William J. McKenna. The volcano optimizer generator: Extensibility and efficient search. In *ICDE '93*, pages 209–218, 1993.

[GMW99] Roy Goldman, Jason McHugh, and Jennifer Widom. From semistructured data to XML: Migrating the Lore data model and query language. In *ACM SIGMOD WebDB Workshop '99*, pages 25–30, 1999.

[HFLP89] Laura M. Haas, Johann Christoph Freytag, Guy M. Lohman, and Hamid Pirahesh. Extensible query processing in Starburst. In *SIGMOD '89*, pages 377–388, 1989.

[HH99] Peter J. Haas and Joseph M. Hellerstein. Ripple joins for online aggregation. In *SIGMOD '99*, pages 287–298, 1999.

[IFF+99] Zachary G. Ives, Daniela Florescu, Marc T. Friedman, Alon Y. Levy, and Daniel S. Weld. An adaptive query execution system for data integration. In *SIGMOD '99*, pages 299–310, 1999.

[IHW01] Zachary G. Ives, Alon Y. Halevy, and Daniel S. Weld. An XML query engine for network-bound data. Submitted for publication. Available at data.cs.washington.edu/integration/tukwila/xmlengine.pdf., 2001.

[ILW00] Zachary G. Ives, Alon Y. Levy, and Daniel S. Weld. Efficient evaluation of regular path expressions over streaming XML data. Technical Report UW-CSE-2000-05-02, University of Washington, May 2000.

[KD98] Navin Kabra and David J. DeWitt. Efficient mid-query re-optimization of sub-optimal query execution plans. In *SIGMOD '98*, pages 106–117, 1998.

[NDM+01] Jeffrey Naughton, David DeWitt, David Maier, Ashraf Aboulnaga, Jianjun Chen, Leonidas Galanis, Jaewoo Kang, Rajasekar Krishnamurthy, Qiong Luo, Naveen Prakash, Ravishankar Ramamurthy, Jayvel Shanmugasundaram, Feng Tian, Kristin Tufte, Stratis Viglas, Yuan Wang, Chun Zhang, Bruce Jackson, Anurag Gupta, and Rushan Chen. The Niagara Internet query system. *IEEE Data Engineering Bulletin*, June 2001.

[STD+00] Jayavel Shanmugasundaram, Kristin Tufte, David J. DeWitt, Jeffrey F. Naughton, and David Maier. Architecting a network query engine for producing partial results. In *ACM SIGMOD WebDB Workshop '00*, pages 17–22, 2000.

[Tam] Tamino: The information server for electronic business. http://www.softwareag.com/tamino.

[UF00] Tolga Urhan and Michael J. Franklin. XJoin: A reactively-scheduled pipelined join operator. *IEEE Data Engineering Bulletin*, 23(2), June 2000.

[UF01] Tolga Urhan and Michael J. Franklin. Dynamic pipeline scheduling for improving interactive performance of online queries. In *VLDB '01*, September 2001.

[UFA98] Tolga Urhan, Michael J. Franklin, and Laurent Amsaleg. Cost based query scrambling for initial delays. In *SIGMOD '98*, pages 130–141, 1998.

[WA91] Annita N. Wilschut and Peter M. G. Apers. Dataflow query execution in a parallel main-memory environment. In *Proc. First International Conference on Parallel and Distributed Information Systems (PDIS)*, pages 68–77, December 1991.

[XLN] eXcelon: The XML application development environment. http://www.odi.com/excelon/main.htm.