Adaptive Query Processing for Internet Applications

Zachary G. Ives
University of Washington
zives@cs.washington.edu

Alon Y. Levy
University of Washington
alon@cs.washington.edu

Daniel S. Weld
University of Washington
weld@cs.washington.edu

Daniela Florescu
INRIA Rocquencourt
Daniela.Florescu@inria.fr

Marc Friedman
Viathan Corp.
marc@viathan.com

Abstract

As the area of data management for the Internet has gained in popularity, recent work has focused on effectively dealing with unpredictable, dynamic data volumes and transfer rates using adaptive query processing techniques. Important requirements of the Internet domain include: (1) the ability to process XML data as it streams in from the network, in addition to working on locally stored data; (2) dynamic scheduling of operators to adjust to I/O delays and flow rates; (3) sharing and re-use of data across multiple queries, where possible; (4) the ability to output results and later update them. An equally important consideration is the high degree of variability in performance needs for different query processing domains: perhaps an ad-hoc query application should optimize for display of incomplete and partial incremental results, whereas a corporate data integration application may need the best time-to-completion and may have very strict data "freshness" guarantees. The goal of the Tukwila project at the University of Washington is to design a query processing system that supports a range of adaptive techniques that are configurable for different query processing contexts.

1 Introduction

Over the past few years, a new set of requirements for query processing has emerged, as Internet and web-based query systems have become more prevalent. In this emerging data management domain, queries are posed over multiple information sources distributed across a wide-area network; each source may be autonomous and may potentially have data of a different format. In some applications, the Internet query systems' results are fed into other data management tools; in other cases, the system interacts directly with the user in an ad-hoc environment. In certain contexts, the query processing system will handle a small number of concurrent queries; in others, there can be hundreds or even thousands of simultaneous requests. These different Internet query applications have many common requirements, but also require certain context-specific behaviors.

Modern query processors are very effective at producing well-optimized query plans for conventional databases, by leveraging I/O cost information as well as histograms and other statistics to determine the best executable plans. However, data management systems for the Internet have demonstrated a pressing need for new techniques. Since data sources in this domain may be distributed, autonomous, and heterogeneous, the query optimizer will often not have histograms or any other quality statistics. Moreover, since the data is only accessible via a wide area network, the cost of I/O operations is high, unpredictable, and variable.

These factors can be mitigated through *adaptive query processing*, where the query processor adapts its execution in response to data sizes and transfer rates as the query is being executed. Moreover, the high I/O costs suggest that data should be processed as it is streaming across the network (as is done in relational databases with pipelining), scheduling of work should be dynamic to accommodate I/O latencies and data flow rates, and re-use and sharing of intermediate query results should be done wherever possible — both across concurrent queries, and between successive queries that execute within some short time delta of each other.

Adaptive query processing helps solve most of the problems common to Internet query systems. However, another important issue in using adaptive techniques, one that has seldom been considered, is the needs of the specific query context: the performance goals, as well as the applicable adaptive techniques, may vary widely depending on the application. For an ad-hoc, interactive query domain, the user may wish to see incomplete results quickly, but for a business-to-business environment, the emphasis may be on providing complete results as quickly as possible, with strict guarantees about data freshness.

The Tukwila project at the University of Washington is a data integration system, in which we attempt to answer queries posed across multiple, autonomous, heterogeneous sources. All of these data sources are mapped into a common *mediated schema*. The data integration system attempts to reformulate the query into a series of queries over the data sources, then combine the data into a common result. Tukwila's ancestors, the Information Manifold [LRO96] and Razor [FW97], focused on the problems of mapping, reformulation, and query planning; Tukwila attempts to address the challenges of generating and executing plans efficiently with little knowledge and variable network conditions.

The goal of Tukwila is to support efficient query processing of streaming XML data using adaptive query processing techniques, including display of incremental results and the sharing of sub-results across queries. In conjunction with this, we believe there is need for a method of expressing query processing *policies* — providing different query performance behaviors for different contexts.

In this paper we discuss a number of important areas that must be addressed using adaptive techniques for an effective wide-area XML data integration system. The paper is organized as follows: we begin in Section 2 with an overview of the different dimensions of adaptive query processing, which underly our research agenda for the Tukwila system. Section 3 describes adaptive techniques currently used within the Tukwila data integration system, and explains how they address some of the problems in this domain; in Section 4, we discuss the current focus areas of the Tukwila project. Finally, we conclude in Section 5.

2 Context-Specific Requirements in Adaptive Query Processing

Adaptive query processing encompasses a variety of techniques, some of which date back to the beginnings of relational database technology. These techniques can be classified by the granularities of adaptivity, as in the paper by Hellerstein et al. in this issue. Here we present an orthogonal classification of these techniques, based on the set of dimensions over which applications of adaptive query processing differ. We argue that adaptive query processors should be built in a flexible fashion, so they can be easily configured for any of these different application contexts. Below we begin by identifying the set of dimensions for Internet query processing, and in Sections 3 and 4, we discuss how current and future work on Tukwila addresses the requirements of these dimensions.

Data model To this point, most adaptive query processing techniques have focused on a relational (or object-relational) data model. While there are clearly important research areas within this domain, other data models may require extensions to these techniques. In particular, XML, as a universal format for describing data, allows for hierarchical and graph-structured data. We believe an execution model similar to pipelining is important for the XML realm, as processing of streaming data is of growing impact.

Remote vs. local data Traditional database systems have focused on local data. Recent work has focused on techniques for increasing the performance of network-bound query applications, including [UFA98, UF99, IFF+99, AH00, HH99]. (See the Hellerstein et al paper in this volume for greater detail.)

First vs. last tuple For domains in which the data is used by an application, the query processor should optimize for overall query running time — the traditional focus of database query optimizers. Most of today's database systems do all of their optimization in a single step; the expectation (which does not generally hold for large query plans [AZ96] or for queries over data where few statistics are available) is that the optimizer has sufficient knowledge to build an efficient query plan. The INGRES optimizer [SWKH76] and techniques for mid-query re-optimization [KD98] often yield better running times, because they re-optimize later portions of the query plan as more knowledge is gained from executing the earlier stages. Similar re-optimization techniques can also be applied to interactive domains, as discussed in [IFF+99, AH00, UFA98], because they can often produce output faster by using a superior query plan.

Approximate results In interactive domains, we may wish to see incremental display of the query results, with incomplete or approximate answers that evolve towards their final values. Operators supporting output of partial results have been a focus of recent work in [HHW97], which provided incremental display of approximate results for root-level aggregation, and [STD+00], which proposed a more general approach for providing partial results on demand. However, another important aspect of this area is a method of specifying *when* to provide partial answers, as the user may only want to see tentative results for certain data items. Moreover, a more formal definition is needed for the semantics of when a partial or approximate result is meaningful.

Incremental updates In certain applications where data constantly changes, it is important to be able to execute the query over an initial data set, and thereafter to process "deltas" describing updates to the original data values. Early work in this area includes the partial-results feature of the Niagara system [STD+00].

Number of queries If the domain includes large numbers of similar queries being posed frequently, the query processor should generate query plans with a focus on materialization of partial results for future reuse, and it should make use of common subexpressions. Work in this area includes the NiagraCQ [CDTW00] system at Wisconsin and the OpenCQ and WebCQ projects at Georgia Tech. This problem is similar to that of multi-query optimization [Sel88, RSSB00] but with a more "online" character — as optimization is done for potential *future* reuse of subquery results — and generally larger numbers of queries.

Freshness Data may often be prefetched and cached by the query processor, but the system may also have to provide data freshness guarantees. Caching and prefetching are well-studied areas in the database community. Likewise, the work on rewriting queries over views [Lev00] can be used to express new queries over cached data, rather than going to the original data sources.

3 The Tukwila Data Integration System

In a domain where costs are unpredictable and dynamic, such as data integration for the wide area, a query processing system must react to changing conditions and acquired knowledge. This is the basic philosophy behind the Tukwila project, which focuses on providing a configurable platform for adaptive query processing of streaming data.

In this section, we present an overview of the basic techniques implemented in Tukwila. There are three primary aspects to the Tukwila adaptive framework: an event-condition-action-based rule system, a set of adaptive operators, and the ability to incrementally re-optimize a query plan as greater knowledge about the data is gained. Here we provide a brief overview of these capabilities; for more information, see [IFF+99].

3.1 Controlling Adaptive Behavior

An important need in dealing with network-based query sources is the ability to respond to unexpected conditions: slow data sources, failed sources, amounts of data that are much larger than expected, etc. In order to handle conditions such as these, Tukwila incorporates event-condition-action rules that can respond to execution events such as operator start, timeout, insufficient memory, end of pipeline, and so forth. In response to these events, Tukwila can return to the query optimizer to re-optimize the remainder of a query plan; it can modify memory allocations to operators; it can switch to an alternate set of data sources. Note that these rules are at a lower granularity than triggers or active rules: they respond to events at the sub-operation level, and can also modify the behavior of query plan operators.

3.2 Intra-Operator Adaptivity

The Tukwila system provides two operators that can respond to varying network conditions and produce optimal behavior. The first is an implementation of the pipelined hash join [WA91] with extensions to support overflow of large hash tables to disk; in many ways it resembles the hash ripple join [HH99] and the XJoin [UF99].

A pipelined hash join operates with two hash tables, rather than the single hash table of a typical hybrid hash join. A tuple read from one of the operator's inputs is stored in that input's hash table and probed against the opposite table. Each input executes in a separate thread, and this provides two highly desirable characteristics: it allows overlap of I/O and computation, which is important in an I/O-bound environment, and it produces output tuples as early as possible. The pipelined hash join also adjusts its behavior to the data transfer rates of the sources. The trade-off is that it uses more memory than a standard hybrid hash join; however, this problem can be mitigated with the overflow strategies implemented by Tukwila or the XJoin operator.

In many web applications, there may be multiple sites from which the same input data can be obtained; some of these data sources may be preferable to others, perhaps because of connection speed or cost. Tukwila's *collector* operator provides a robust method for reading data from sources with identical schemas: according to a *policy* specified in Tukwila's rule language, the collector attempts to read from a subset of its sources; if a given source is slow or unavailable, the collector can switch to one or more alternate data sources. This operator allows the query engine to choose data sources based on criteria such as availability or speed.

3.3 Incremental Re-Optimization

Adaptive behavior during query execution is key in situations where I/O costs are variable and unpredictable. When data sizes are also unpredictable, it is unlikely that the query optimizer will produce a good query plan, so it is important to be able to modify the query plan being executed. As a result, Tukwila supports incremental re-optimization of queries during particular plan execution points.

The Tukwila re-optimization model is based on *fragments*, or pipelined units of execution. Fragment boundaries, at which a pipeline is broken and the results are materialized, are chosen by the optimizer according to their cost and potential benefits. In general, a large query plan must already be broken into smaller pipelines so operators will fit into memory; this is particularly true if memory intensive operators such as the pipelined hash join are used. At each materialization point, Tukwila's execution system can check whether the result cardinality was close to that expected by the optimizer; if the cardinality is sufficiently divergent, Tukwila will keep the current query subresults and re-optimize the remainder of the query plan, using the subresults as inputs for a new and better plan.

The Tukwila model for re-optimization is similar to that proposed in [KD98], but it allows the optimizer to choose fragmentation points in an integrated, cost-based approach, rather than adding the capabilities in a separate postprocessing step.

4 Current Areas of Focus in Tukwila

The Tukwila system already supports a number of adaptive techniques, but the system is being extended in a number of ways. Our current work focuses on many of the areas discussed in Section 2.

4.1 XML: a Foundation for Data Integration

Tukwila was initially developed for a relational data model, and required *wrappers* for every data source, to translate data from source formats into the standard Tukwila data format. XML reduces the difficulty of building wrappers, as most data sources have begun to include XML export capabilities, and as various HTML-to-XML wrapper toolkits (e.g. [SA99, LPH00]) have emerged[1]. However, the use of XML, which supports flat, hierarchical, and graph-structured data, has led to a natural extension of the Tukwila data model to fully support semistructured data.

Our data model is based on an ordered, directed-graph approach like that of XML-QL [DFF$^+$99]. This model is powerful enough to support any of the proposed XML query languages, including XML-QL, XQL [RLS98], XPath [CD99], and Quilt [CRF00]. As discussed below, one of our goals in Tukwila is to support the processing of streams of changing data. To this end, we are developing a language for specifying updates to XML documents.

4.2 Processing Streaming XML Data

To this point, XML query processors have worked by mapping XML data into an underlying local store — relational, object-oriented, or semistructured — and have done their processing within this store. For a network-bound domain where data may need to be re-read frequently from autonomous sources to guarantee "freshness," this approach does not produce good performance. Thus it is imperative that an XML data integration system support direct querying of data as it streams in, much as a relational engine can pipeline tuples as they stream in.

In order to provide pipeline-like processing of network data, we must be able to support efficient evaluation of *regular path expressions* over the incoming data, and incremental output of the values they select. Regular path expressions are a mechanism for describing traversals of the data graph using edge labels and optional regular-expression symbols such as the Kleene-star (for repetition) and the choice operator (for alternate sub-paths). Regular path expressions bear many similarities to conventional object-oriented path expressions, and can be computed similarly; however, the regular expression operators may require expensive operations such as joins with transitive closure.

We have developed the *x-scan* operator, which evaluates regular path expressions across incoming XML data, and which binds query variables to nodes and subgraphs within the XML document. X-scan is discussed in greater detail in [ILW00], and includes support for both tree- and graph-structured documents, while preserving document order.

Building upon X-scan, we are developing a complete query processor over streaming XML data. The key additional operators we are considering are a *nest* operator, which nests subelements under parents, in a join-like fashion; and a *fuse* operator, which can be used to support graph-model features in an XML output document by consolidating multiple output nodes.

4.3 Specifying Adaptive Behavior

In Section 2, we discussed a number of dimensions of adaptive query processing. Different data management applications have very different needs within these dimensions. Factors may include whether to optimize for first

[1] Legacy applications will still need wrappers to convert from their formats to XML, but those wrappers should be generic enough to be usable by all XML data consumers, unlike the previous situation where a separate wrapper needs to be created for each data provider and consumer.

or last tuple, how fresh the data from each query must be (and thus how long data can be cached), and whether (and when) the user should see approximate or incomplete data.

Additionally, the query optimizer should behave quite differently if the domain is one in which many similar queries are being posed, rather than one in which a few simple queries are given. For the multi-query case, the query processor should evaluate the potential benefits of materializing subresults for reuse in future queries.

Although optimizing for each of these various needs has been fairly well-studied, much less work has been done on actually expressing the query processing requirements, and on being able to support all of these cases within a single unified framework. This is an area we plan to address within Tukwila: developing a system to support a wide range of applications, and, equally important, providing a configuration language for specifying the requirements of a given domain.

4.4 Increasing Pipelined Behavior

In terms of query execution, an important need for interactive applications is to facilitate output of first tuples — the user should receive results as soon as possible. Tukwila includes adaptive operators whose intent is to address this requirement.

However, these needs must be balanced by the fact that the query optimizer, which does not have good knowledge of the data sources, may have produced a suboptimal plan. The optimizer initially divides the plan into fragments (pipelines with materialization points) based on expected memory usage and other factors such as confidence in its statistics. Each of these materialization points breaks the pipeline, generally slowing time to first tuple — however, if the plan gets re-optimized into a more efficient form, the net result should be a faster time to completion, and potentially even a better time to first tuples.

4.4.1 Dynamically Choosing Materialization Points

Clearly, there is a trade-off between the number of materialization points and the query processing time. Unfortunately, with few statistics available, the optimizer is unlikely to be able to choose good materialization points; it is likely to have too many, too few, or poorly placed breaks in the pipeline. (This problem is also present in traditional systems with quality statistics, appearing for complex queries with many join operations.) We are investigatating the performance implications of choosing the materialization points adaptively, during plan execution. In our approach, the query optimizer creates long pipelines; when these pipelines run out of memory (e.g. a join algorithm needs to overflow to disk), the execution engine will, using "hints" provided by the optimizer, insert a new materialization point into the middle of the pipeline. All operators "upstream" of this new materialization point will flush their results to disk; execution of the operators below the materialization point will continue. Once they complete, the upstream operators will reload their intermediate results, begin reading from the materialized file, and resume normal operation.

We expect that there will be several benefits to this approach. First, early results will likely be able to percolate through the entire long pipeline before the system runs out of memory — this speeds time to initial tuples. Second, the system will only insert materialization points where necessary, something that is extremely difficult to do statically. Third, the cost of breaking a pipeline should generally be less than that of having multiple join algorithms simultaneously overflowing, as it allows the query processor to "stage" portions of the data that exceeds memory.

4.4.2 Returning Incremental Results

Another important feature that is important for interactive applications is the ability to display approximate results incrementally. Initial work in this area has been done in the context of the Niagara system [STD$^+$00], which allows the user to request partial results at any time, and also within the CONTROL project [HHW97] for top-level aggregate operators.

Our focus in this area is on two important problems. First, an interactive query system would ideally provide incremental results for all query types in an interactive, browsable window, where the query processor "focuses" on finalizing the results currently in the user's view. Second, it will often be the case that for a given domain or query, approximate or partial results are only useful for certain items within the data set. A system that supports partial results should also support a mechanism for expressing which data items should be approximated. We believe this should be one aspect of the configuration language discussed in Section 4.3.

5 Conclusions

Adaptive query processing is a rapidly growing field, as evidenced by this special issue. Certain aspects of this work go back to the early days of relational databases, but the evolution of data integration and data management systems for the Internet has led to a number of recent developments.

We believe that one of the most important areas of future exploration should be in developing a system flexible enough to meet the wide range of domain-specific needs, and providing a means of specifying the relevant parameters to the system. The Tukwila project is attempting to address aspects of both of these problems, using XML as the standard data format and data model. We believe that the current system has taken a number of steps in this direction, and that our current and future work will take us much closer to a comprehensive data management solution for Internet-based data.

References

[AH00] Ron Avnur and Joseph M. Hellerstein. Continuous query optimization. In *SIGMOD 2000, Proceedings ACM SIGMOD International Conference on Management of Data, May 14-19, 2000, Dallas, Texas, USA*. ACM Press, 2000.

[AZ96] Gennady Antoshenkov and Mohamed Ziauddin. Query processing and optmization in Oracle Rdb. *VLDB Journal*, 5(4):229–237, 1996.

[CD99] James Clark and Steve DeRose. XML path language (XPath) recommendation. http://www.w3.org/TR/1999/REC-xpath-19991116, November 1999.

[CDTW00] Jianjun Chen, David DeWitt, Feng Tian, and Yuan Wang. NiagaraCQ: A scalable continuous query system for internet databases. In *SIGMOD 2000, Proceedings ACM SIGMOD International Conference on Management of Data, May 15-18, 1999, Dallas, TX, USA*, 2000.

[CRF00] Donald D. Chamberlin, Jonathan Robie, and Daniela Florescu. Quilt: An XML query language for heterogeneous data sources. In *WebDB (Informal Proceedings) 2000*, pages 53–62, 2000.

[DFF$^+$99] Alin Deutsch, Mary F. Fernandez, Daniela Florescu, Alon Levy, and Dan Suciu. A query language for XML. In *Proceedings of the International Word Wide Web Conference, Toronto, CA*, 1999.

[FW97] Marc Friedman and Daniel S. Weld. Efficiently executiong information-gathering plans. In *Proceedings of the Fifteenth International Join Conference on Artificial Intelligence*, pages 785–791, 1997.

[HH99] Peter J. Haas and Joseph M. Hellerstein. Ripple joins for online aggregation. In Alex Delis, Christos Faloutsos, and Shahram Ghandeharizadeh, editors, *SIGMOD 1999, Proceedings ACM SIGMOD International Conference on Management of Data, June 1-3, 1999, Philadephia, Pennsylvania, USA*, pages 287–298. ACM Press, 1999.

[HHW97] Joseph M. Hellerstein, Peter J. Haas, and Helen Wang. Online aggregation. In Joan Peckham, editor, *SIGMOD 1997, Proceedings ACM SIGMOD International Conference on Management of Data, May 13-15, 1997, Tucson, Arizona, USA*, pages 171–182. ACM Press, 1997.

[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 1999, Proceedings ACM SIGMOD International Conference on Management of Data, June 1-3, 1999, Philadephia, Pennsylvania, USA*, pages 299–310, 1999.

[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 Laura M. Haas and Ashutosh Tiwary, editors, *SIGMOD 1998, Proceedings ACM SIGMOD International Conference on Management of Data, June 2-4, 1998, Seattle, Washington, USA*, pages 106–117. ACM Press, 1998.

[Lev00] Alon Y. Levy. Answering queries using views: A survey, 2000. Manuscript available from `www.cs.washington.edu/homes/alon/views.ps`.

[LPH00] Ling Liu, Calton Pu, and Wei Han. XWRAP: An XML-enabled wrapper construction system for web information sources. In *Proceedings of the 16th International Conference on Data Engineering, San Diego, CA USA*, pages 611–621, 2000.

[LRO96] Alon Y. Levy, Anand Rajaraman, and Joann J. Ordille. Querying heterogeneous information sources using source descriptions. In T. M. Vijayaraman, Alejandro P. Buchmann, C. Mohan, and Nandlal L. Sarda, editors, *VLDB'96, Proceedings of 22th International Conference on Very Large Data Bases, September 3-6, 1996, Mumbai (Bombay), India*, pages 251–262. Morgan Kaufmann, 1996.

[RLS98] Jonathan Robie, Joe Lapp, and David Schach. XML Query Language (XQL). http://www.w3.org/TandS/QL/QL98/pp/xql.html, September 1998.

[RSSB00] Prasan Roy, S. Seshadri, S. Sudarshan, and Siddhesh Bhobe. Efficient and extensible algorithms for multi query optimization. In *SIGMOD 2000, Proceedings ACM SIGMOD International Conference on Management of Data, May 14-19, 2000, Dallas, Texas, USA*. ACM Press, 2000.

[SA99] Arnaud Sahuguet and Fabien Azavant. Building light-weight wrappers for legacy web data-sources using W4F. In *VLDB'99, Proceedings of 25th International Conference on Very Large Data Bases, September 7-10, 1999, Edinburgh, Scotland, UK*, pages 738–741. Morgan Kaufmann, 1999.

[Sel88] T. K. Sellis. Multiple-query optimization. *ACM Transactions on Database Systems*, 13(1):23–52, 1988.

[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 *WebDB (Informal Proceedings) 2000*, pages 17–22, 2000.

[SWKH76] Michael Stonebraker, Eugene Wong, Peter Kreps, and Gerald Held. The design and implementation of INGRES. *TODS*, 1(3):189–222, 1976.

[UF99] Tolga Urhan and Michael J. Franklin. XJoin: Getting fast answers from slow and bursty networks. Technical Report CS-TR-3994, University of Maryland, College Park, February 1999.

[UFA98] Tolga Urhan, Michael J. Franklin, and Laurent Amsaleg. Cost based query scrambling for initial delays. In *SIGMOD 1998, Proceedings ACM SIGMOD International Conference on Management of Data, June 2-4, 1998, Seattle, Washington, USA*, 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.