Measuring the Scalability of a XML-QL Relational Database Management System

Rebecca J. Cathey, Steven M. Beitzel, Eric C. Jensen, Angelo J. Pilotto,
David Grossman
Information Retrieval Laboratory
Department of Computer Science
Illinois Institute of Technology
Chicago, IL 60616

{cathey, beitzel, jensen, pilotto, grossman}@ ir.iit.edu

Abstract

The explosive growth of XML has led to an increasing need for scalable XML retrieval systems. Our XML retrieval system, the SQLGenerator, stores XML of any schema in a fixed schema relational database and supports a full-featured semistructured query language, XML-QL, through optimized translation of its semantics to relational SQL queries. This paper examines the scalability of this approach with respect to increasing data size. We index four XML collections ranging in size from 500MB to 2GB that were generated using a standard XML generator, XBench. We then compare the execution times of 11 standard XBench queries, covering a wide range of semistructured query features, whose semantics were directly translatable from their original XQuery language to XML-QL. Although it is difficult to estimate the theoretical baseline for scalability of these query features in an RDBMS, many of the queries' runtimes grow linearly with respect to the size of the document collection.

1. Introduction

The eXtensible Markup Language (XML) is playing an increasingly important role in the exchange of a wide variety of data on the Web and elsewhere [4]. XML is a self-describing language, which means that it can define it's own schema. Furthermore, with data of a hierarchical nature, defined schemas allow XML to maintain hierarchical continuity. Combined, the hierarchical and self-describing aspects of XML allow for tremendous flexibility when designing XML collections.

Because of the growth of XML data, efficient methods to store and search XML documents are needed. Our XML-QL to SQL translator, SQLGenerator, indexes XML documents with differing schemas into a static schema relational database. Then it takes an XML-QL query as input and produces equivalent SQL for that query. By using SQL as an intermediate step, we avoid the complexities of building a full database engine such as index structures, storage management, query optimization, and concurrency control. Additionally, the generated SQL is not tied explicitly to any XML schema, DTD, or any other rigid XML specification. This enables the SQLGenerator to work with documents of arbitrary XML schemas without modification of the underlying relational schema.

In this paper, we demonstrate the scalability of the SQLGenerator by testing it against increasingly large XML document collections. In section 2 we discuss previous work in this area. Section 3 gives a brief description of the SQLGenerator. In section 4 we present the results of running specific queries on different document collections of varying sizes. We provide analysis of these results in section 5. Finally, section 6 states our conclusions.

2. Related Work

XML is a semistructured data format. Therefore, it has no rigid, predefined structure or schema. In order to search XML, a query language designed to search semistructured data is needed [9, 5]. Query languages have progressed from simple path based languages to complex query languages that offer a wide range of functionality such as and XML-QL [3]. Most semistructured query languages have similar underlying semistructured query foundations. The SQLGenerator implements XML-QL, a query language initially developed by AT&T Research Laboratories [10].

Due to it's hierarchical nature, there is a diverse body of prior work on methods of storing and indexing XML data. We choose to focus on the method that utilizes a relational database management system (RDBMS). Yoshikawa and Amagasa classify methods for designing a database schema for XML into two categories: structured-mapping approaches and model-mapping approaches [21]. In the structure-mapping approach, a database schema is defined for each XML schema or Document Type Descriptor (DTD)[19, 20, 14]. The model-mapping approach addresses the issue of mapping schemaless XML documents. In this approach, a fixed database schema is used to store the structure of all XML documents. Examples of this include the Edge-oriented approach and the node-oriented approach. The edge-oriented approach developed by Florescu and Kossman is a simple scheme that stores all attributes in a single table [12]. Another variant of the edge approach is to store the attribute names in another table [11] or to store all associations of the same type in the same binary relation [18]. The node-oriented approach maintains nodes rather than edges [21]. With the start and end points of a node it maintains a containment relationship for ancestor-descendent relationships.

The wide use of XML coupled with the multiple storage and querying techniques available has led to the development of numerous XML search systems. The Agora System employs XML as the user interface format, while all the input to the query processor consists of relational tuples [16]. Agora uses a subset of the Quilt query language [17] and stores XML into the relational database using the structure-oriented approach. Agora was demonstrated using several collections of cooking recipes, nutritional information, nutritional information and XML medical files. Dehaan et. al, discusses a system that translates an XQuery query into an SQL query [8]. The XML documents are encoded using dynamic intervals which allow them to represent an iterated application of XQuery expressions on a sequence of XML documents by a single relational query. The performance of two methods utilizing dynamic intervals are compared with several other XQuery processers and native XML database systems. The time over specific queries using different systems is then compared to show the validity of their system. Through experimentation, they determined that the dynamic interval-based plans scale (almost) linearly when enabled by merge-join evaluation strategies. This was a significant improvement over the quadratic behavior exhibited by a number of other XQuery systems. However, the method used to determine linear time is based solely on timings from one query and the methods used to map and query XML documents are XQuery dependent. To show the scalability of our system, we evaluate it using varying test collection sizes. We were unable to locate an XML-QL search engine that that fully supported the functionality of XML-QL and attempts to be scalable. AT&T's reference implementation was developed strictly to show the functionality of XML-QL. Most XML-QL or XQuery engines either did not support all the functionality of our XML-QL system or stored all of the XML data in memory and would not have been able to support the size of our XML document collections.

In addition to evaluation, there has been research comparing the different methods used to implement database schemas. Kudrass analyzes different storage and retrieval methods for schemaless XML documents [15]. By comparing several structure-oriented approaches to storing the entire XML document as one large character object, it was determined that the feasibility of using the structure-oriented approach to map documents into tables is restricted to data-centric documents with little prose. Yoshikawa and Amagasa show advantages of the node-oriented method for the model-based approach of designing a fixed database schema [21]. Jiang et. al, compares the scalability of different model-mapping approaches to a variant of the edge-oriented approach [13]. Based on parent-child relationships, XParent was found to outperform XRel [21] and Edge [11].

The majority of XML collections used for testing XML search systems were typically small, ranging from 7MB to 280MB. In addition, it is sometimes necessary to generate XML documents using benchmarks such as XMark [2] and XBench [1]. The size of collections that can be generated with these tools ranges from 100KB to 10GB. Because of the need to evaluate existing XML systems, the INitiative for the Evaluation of XML retrieval (INEX) [7] aims to provide a large XML test collection with appropriate scoring methods. The INEX document collection consists of $12,107$ documents, totalling 494MB in size. The topics as well as the documents used in INEX are open to INEX participants only, hence we were not able to use them in this study. In order to test the scalability of the SQLGenerator, we needed large XML document collections. Although experiments on smaller XML collections can help to measure performance and relative scalability against other XML retrieval systems, they do not measure the overall scalability of the system. Our evaluation differs from previous ones in that we focus on the scalability aspect of performance by showing the ability of our system to handle large amounts of XML efficiently.

3. System Description

The SQLGenerator is a scalable XML retrieval engine that fully implements the XML-QL query language by translating it to SQL. In addition to its range of capabilities, XML-QL provides an intuitive means of writing semistructured queries resembling SQL that use XML data bindings in a format very similar to the XML documents be-

ing searched. Although we have chosen to use XML-QL, we believe that our relational schema and translation techniques can be applied to similar query languages such as XQuery. The SQLGenerator incorporates XML documents of any schema without requiring modifications to the fixed underlying relational schema they are stored in. Because the documents do not necessarily have defined DTDs or XML schemas, we employ the model-mapping, edge-oriented approach to store the XML documents in the database. More detail on the storage and translation process used by the SQLGenerator are given in [6].

4. Results

We ran experiments on a Sun Fire V880 with 4 750MHz UltraSparc-III CPUs and 8GB of main memory. We implemented SQLGenerator using Java 1.4 and stored the XML document Collections using Mysql 4.0.16. The Mysql database is stored on a 10K RPM fibre channel drive.

We generated our XML document collections using XBench. XBench has an option to create data-centric or topic-centric documents. A topic-centric document contains significantly more text than element tags, while a data-centric document dedicates more parts to tags. There is also an option to create single or multiple document collections. We chose to use data-centric documents as they contained data relevant to database applications rather than documents marked up in XML. We also chose to implement multiple document collections because they contained multiple XML schemas. XBench produces a number of XML documents in the e-business domain. There are customer, address, country, item, author, and multiple order XML documents. We generated a 2GB XML document collection. Then we took random subsets of the 2GB XML document collection to obtain 500MB, 1GB, and 1.5GB XML document collections. Using the edge-oriented approach most of the data is stored in a single table called the `pinndx` table. The number of rows for the 500MB, 1GB. 1.5GB. and 2GB databases are 11.32, 22.64, 33.96, and 45.28 million respectively. XBench provides a set of XQuery queries to run against the generated data. A subset of their queries is created for queries against the data centric multiple document collection. We created a corresponding query subset of XML-QL queries to run against the XML collections. We tested our queries on a small XML collection to make sure they returned the same results as a corresponding XQuery query returns. The results were the same for all the queries except Q10 which returns the same results, however, it does not return the results in the same order. The reason for this is because our Q10 uses skolem functions to group by certain attributes. In the original query, however, "order by" was used to order and group results. There were several functionalities present in XQuery that have no equivalent in XML-QL. For example, XQuery contains the "exists" keyword, whereas there is no such functionality for XML-QL. Therefore, we did not include queries that could not be translated correctly.

The queries test the functionality of our system in different ways. The simplest query, Q1, returns a value from a document where a certain attribute has a specific value (id="1"). Q1 tests the ability of our system to handle matching of values on a shallow level. Q3 uses skolem functions to group orders with total amount bigger than 11000.0, by customer id and then calculates the total number of each group using aggregate predicates. This tests the ability of our system to handle both skolem functions and aggregate predicates. Q4 queries for a specific item where the id of the previous one was a specific value. This tests the ability of the system to handle index expressions. Q5 is similar to Q4, however, it returns only the first item, thus performing absolute ordering not relative. Q6 tests the ability of the system to determine whether any item in an invoice has a discount rate higher than 0.02. If such an item exists, the whole invoice is reconstructed, not just the particular item. Q8 and Q9 test the ability of the system to query when the exact path of the item is not known. This is accomplished through the use of regular expressions. Q10 uses skolem functions to group results by the shipping type. Like Q3, this tests the ability of our system to handle skolem functions, however, it also tests the ability of the system to reconstruct large amounts of XML as the result set is large for this query. Q12 and Q16 retrieve large sections of XML where a specific criteria is met. These queries test the ability of the system to reconstruct large portions of XML documents by storing the XML in a variable or using CONTENT_AS. Q17 tests the ability of the system to perform a uni-gram search by only retrieving authors whose biographies contain the word "hockey".

We measured the total time to process a query as well as the time to parse the XML, generate the SQL, execute the SQL, and reconstruct the XML. We also show the number of results returned. The breakdown of times for each query and each data collection are given in Figure 2. We omitted the time to parse the XML and generate the SQL as that was always less than one second. The total times of execution for each query on each XML collection are in Figure 1. Figure 3 shows the execution times of each query as a function of the collection size.

5. Analysis

It is difficult to estimate the theoretical baseline for the scalability of these query features in a RDBMS. However, we expect linear performance is satisfactory. From our experiments we can see that for many of the test queries, execution time grows linearly or super-linearly with respect

	Q1	Q3	Q4	Q5	Q6	Q8	Q9	Q10	Q12	Q16	Q17
500MB	4.46	54.21	27.73	4.36	1980.84	3.57	10.42	196.27	0.99	9.79	2.35
1GB	9.50	108.41	58.37	13.54	3965.71	5.65	14.62	392.61	1.12	11.90	3.92
1.5GB	11.43	19.34	82.67	614.94	5920.28	9.57	22.17	585.49	2.72	12.74	5.57
2GB	16.69	211.56	114.60	592.03	7982.45	36.01	26.02	236.24	3.18	14.90	7.25

Figure 1. Execution times (in seconds)

		Q1	Q3	Q4	Q5	Q6	Q8	Q9	Q10	Q12	Q16	Q17
(a)	XML Result Size	1	3508	1	1	52853	4	2	3517	1	1	2755
	SQL Execution	3.92	52.61	27.20	212.47	18.37	10.82	9.98	194.70	0.45	9.26	1.35
	XML Reconstruction	0.07	0.60	0.01	0.09	1947.49	0.01	0.01	0.85	0.07	0.10	1.35
(b)	XML Result size	1	6917	1	1	105760	2	3	6958	1	1	5476
	SQL Execution	8.93	106.22	57.84	12.98	36.71	40.04	14.54	390.11	0.59	11.40	2.58
	XML Reconstruction	0.07	1.09	0.01	0.08	3881.23	0.01	0.01	1.53	0.07	0.08	0.70
(c)	XML Result size	1	10356	1	1	158545	2	2	10448	1	1	8348
	SQL Execution	10.87	16.26	82.15	614.25	56.03	22.63	21.74	582.16	2.16	12.74	3.86
	XML Reconstruction	0.08	1.47	0.01	0.19	5821.98	0.01	0.01	1.91	0.07	0.01	0.82
(d)	XML Result size	1	13800	1	1	211884	4	4	13971	1	1	11139
	SQL Execution	16.11	208.02	114.07	591.41	128.11	73.72	25.59	232.11	2.64	14.39	5.30
	XML Reconstruction	0.08	1.92	0.01	0.09	7762.38	0.01	0.01	2.79	0.07	0.07	1.14

Figure 2. Breakdown of time for (a) 500MB (b)1GB (c) 1.5GB and (d)2GB collections (in seconds)

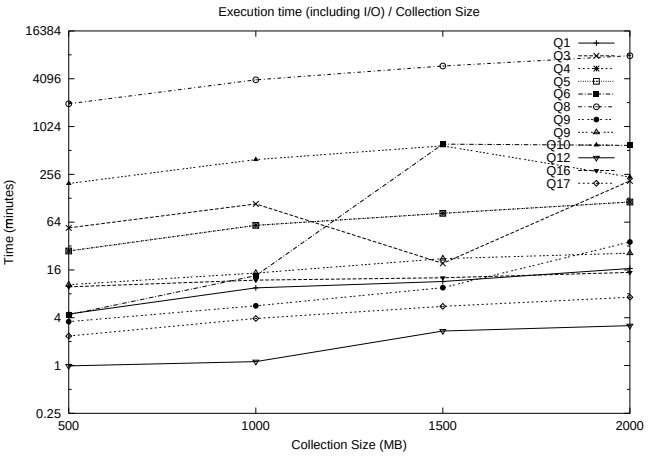

Figure 3. Execution Time per Query

to the size of the XML collection. Several queries had some anomalies that need to be examined. For example, the queries Q3, Q8, and Q10 all run linear on three of the four database sizes. Q3 runs linearly on all of the databases except the 1.5GB database. On this database, the change in time drops. Similarly, Q10 runs linearly on the first three databases. For the 2GB database, the change in time drops. Another similar query is Q8, however on the 2GB database the change in time increases. Q9 is almost linear. There is a slight increase in linearity between the 1GB and the 1.5GB database, however, the other times grow linearly with respect to the size of the XML collection. Q5 differs from all the queries as it exhibits anomalous behavior on all the databases. It runs with increasing time from the 500MB to 1.5GB databases, however, the change in time is exponential. Then for the 2GB database the time decreases.

The time required to run different queries fluctuates quite a bit. For example, the time to run Q6 is much higher than the time to run Q17. By looking at the breakdown of time over all the steps, we can see the times to parse the XML and generate the SQL are fairly static throughout all the queries. The cause of the time fluctuations, therefore, lie in the SQL generation and XML reconstruction steps.

The goal of using a relational database is to move the processing functionality to the database. Because the database does most of the work, the SQL execution step can become very time consuming. For example, an XML-QL query converted to an SQL query requires a search and sometimes multiple self joins of the `pinndx` table. Since the `pinndx` table consists of 10-45 million rows, this can often be a long process. However, optimization techniques in the database, such as the proper use of indexes, can significantly speed up SQL execution time. Even with an op-

timized database, this step can be time consuming. For example, the SQL execution time grows exponentially for Q6. For all other queries, however, the SQL execution time follows the pattern of the behavior exhibited over the total execution time as described above.

The XML reconstruction step is occasionally time consuming because the queries sometimes bind complex variables which require the subtrees of XML to be reconstructed. Other queries in which the XML reconstruction step doesn't take as much time do not require large sections of XML to be reconstructed. Another factor that effects the XML reconstruction time is the number of results returned. Notice that the time to reconstruct XML is sometimes linear and sometimes constant. For example, Q1 and Q10 have constant XML reconstruction times over all the XML document collections whereas, Q3 and Q12 experience linear growth within the XML reconstruction step. The reason for the change in growth orders can be attributed to the size of the result set returned. The XML reconstruction step grows linearly for Q3, Q6, Q12, and Q17. Notice that these four queries all have a result set that also grows linearly with respect to the document size. In addition, these four queries are the only four queries in our collection that have non constant result set sizes. Therefore, when a large number of results are returned, the XML reconstruction time grows linearly with respect to the size of the XML collection.

6. Conclusions

In this paper we show that our XML retrieval system, SQLGenerator, is a reliable and scalable method for storing and searching a collection of XML documents. We tested the scalability of the SQLGenerator using collections of XML documents ranging in size from 500MB to 2GB. From our experiments we determined that, for most of our queries, the time of our system grows linearly with respect to the size of the XML document collection.

Acknowledgements

This work is supported by BIT Systems, Inc. Special thanks to Fred Ingham and Tim Lewis for their assistance with the design and testing of various parts of the SQLGenerator.

References

[1] Xbench - a family of benchmarks for xml dbmss. http://db.uwaterloo.ca/ ddbms/ projects/xbench/index.html.

[2] Xmark - an xml benchmark project. http://monetdb.cwi.nl/xml/index.html.

[3] Xml-ql: A query language for xml. http://www.w3.org/TR/1998/NOTE-xml-ql-19980819.

[4] Xml query requirements. http://www.w3.org/TR/xmlquery-req.

[5] S. Abiteboul. Querying semi-structured data. In *ICDT*, pages 1–18, 1997.

[6] S. Beitzel, E. Jensen, A. Pilotto, R. Cathey, O. Frieder, and D. Grossman. Xml retrieval in the classroom. *IIT IR Lab technical report*, July 2003. http://ir.iit.edu/publications/downloads/ SQLGeneratorTechnicalReport.pdf.

[7] S. Dagstuhl. Proceedings of the first workshop of the initiative for the evaluation of xml retrieval (inex).

[8] D. DeHaan, D. Toman, M. Consens, and M. Özsu. A comprehensive xquery to sql translation using dynamic interval encoding. *ACM SIGMOD/PODS 2003 Conference*, 2003.

[9] A. Deutsch, M. Fernandez, D. Florescu, A. Levy, D. Maier, and D. Suciu. Querying xml data. *IEEE Data Engineering Bulletin*, 22(3):12–20, 1999.

[10] A. Deutsch, M. Fernandez, D. Florescu, A. Levy, and D. Suciu. A query language for xml. *International World Wide Web Conference*, 1999.

[11] D. Florescu and D. Kossman. A performance evaluation of alternative mapping schemes for storing xml data in a relational database. Technical report, INRIA, France, 1999.

[12] D. Florescu and D. Kossman. Storing and querying xml data using an rdbms. *IEEE Data Engineering Bulletin*, 22(3):27–34, 1999.

[13] H. Jiang, H. Lu, W. Wang, and J. X. Yu. Path materialization revisited: An efficient storage model for xml data. *Proceedings of the Thirteenth Australian Conference on Database Technologies*, 5:85–94, 2002.

[14] L. Khan and Y. Rao. A performance evaluation of storing xml data in relational database management systems. *Proceeding of the third international workshop on Web information and data management*, pages 31–38, 2001.

[15] T. Kudrass. Management of xml documents without schema in relational database systems. *Information & Software Technology*, 44(4):269–275, 2002.

[16] I. Manolescu, D. Florescu, D. Kossmann, F. Xhumari, and D. Olteanu. Agora: Living with XML and relational. In *The VLDB Journal*, pages 623–626, 2000.

[17] J. Robie, D. Chamberlin, and D. Florescu. The quilt query language for semistructured data and xml. *In Proceedings of the International Workshop on the Web and Databases*, 2000.

[18] A. Schmidt, M. Kersten, M. Windhouwer, and F. Waas. Efficient relational storage and retrieval of XML documents. *Lecture Notes in Computer Science*, 1997:137+, 2001.

[19] J. Shanmugasundaram, K. Tufte, G. He, C. Zhang, D. DeWitt, and J. Naughton. Relational database for querying xml documents: Limitations and opportunities. In Proc. of VLDB, 1999.

[20] F. Tian, D. DeWitt, J. Chen, and C. Zhang. The design and performance evaluation of alternative xml storage strategies. *ACM SIGMOD Record*, 31(1):5–10, 2002.

[21] M. Yoshikawa and T. Amagasa. Xrel: A path-based approach to storage and retrieval of xml documents using relational databases. *ACM Transactions on Internet Technology (TOIT)*, 1(1):110–141, August 2001.