Processing XML Data Streams: A Survey

J. Ulrych

Charles University in Prague, Faculty of Mathematics and Physics, Department of Software Engineering, Czech Republic.

Abstract. Data streams have been gaining importance over last few years as the information systems have become more complex and users have required up-to-date data. Processing data streams, however, causes difficulties since traditional data management systems are not optimized for particular features of data streams. A flexible data item structure in XML data streams, varying arrival rate and bursts of data items, blocking operators in queries and system scalability; these are only few of issues that are need to be solved before the XML data stream management becomes common part of information systems. The purpose of this paper is to summarize current results on XML data stream processing, provide real-world motivation to study XML data streams, show the main problems in XML stream processing (e.g. approximate results in querying and data mining) and point out open issues for future research.

Introduction

In last few years, information systems have been becoming more complex, volumes of data being processed are on increase and the data become dynamic, i.e. updates are more common. There are several issues that make it difficult to process dynamic data using traditional tools for data processing — Database Management Systems (DBMS). Enormous size and number of updates are the most obvious of them. Hence, it is necessary to study new data models — dynamic data models.

One kind of dynamic data models are *data streams*. In contrast to traditional static data models which operate on the data physically stored in persistent computer memory and therefore can be accessed off-line in random-access manner, the data stream model characteristics are [36]:

- The data arrive from a data stream continually.
- No assumptions on the data stream ordering can be made. In practice, data items in the stream are ordered by their *timestamp*. Each data item is assigned a timestamp when it is obtained, measured or enters the stream. However, data arrive from a data stream in order they entered it.
- The data stream length is unbounded.
- The data item is deleted (removed) as soon as it is received from the data stream and processed. There is no way to restore deleted data item.
- Structure of data items in a data stream can change in time (stream is evolving).

In traditional DBMS, answers to the queries are required to be exact. When working on data streams, the unboundness of a stream makes it nearly impossible to provide exact answers. In fact, exact answers are not required under all circumstances. In many applications approximated answers, which are easier to obtain, are sufficient.

Extensible Markup Language (XML) [1] is a general specification for creating special purpose markup languages. It is a text-based markup language designed to be human-legible. The extensibility lies in a capability to define so-called *elements* by users which makes the XML self-describing. This means that we not only encode data themselves, but also we encode metadata describing them which makes it easier to process them. However, notice, that metadata do not define semantics. XML technology is a complex set of tools for processing XML documents.

XML streams have not been created from scratch, but rather emerged as a secondary product of information exchange in XML among systems. Each XML message sent between such systems can be processed as a standalone XML document. However, when we need to analyze long sequence of messages often (e.g. statistical queries), traditional one-by-one message processing is not efficient. Therefore, there is a need to study stream processing. Especially XML stream processing, i.e. long-running queries and transformations provide many open problems for future research.

The rest of this paper is organized as follows. Section *XML streams applications survey* contains a survey of real-world XML data stream applications, Section *Streamed XML* explains XML data streams and theirs differences to traditional data streams and points out benefits of the algorithms developed specially for XML data streams processing. In Section *Querying XML data streams* we provide a brief survey of problems concerning querying the XML data streams. Similarity evaluation on XML data streams is addressed in Section *Data Integration*. Finally, Section *Conclusion* summarizes and concludes the paper.

XML streams applications survey

In this section, we provide a brief survey of XML data stream applications to show the real world motivation to study XML data streams. It is not intended to be exhaustive.

- One source of an XML data stream is a Radio Frequency Identification (RFID) [25]. RFID chips can be used in supermarkets as a replacement for bar codes, in libraries to identify the books, in hospitals for patient and drug identification, car identification for electronic toll collection and many others.

- Recently, a great deal of attention in the mobile-computing has been directed towards building networks of ad-hoc collections of sensor data [30]. The sensor networks can be considered as DBMS. However, there are two main differences between well-established DBMS and sensor networks. First, sensors have only limited processor and power resources, e.g. batteries. The second difference is that sensors provide data continuously in streams, usually in well-defined time intervals, without being asked for that data. Moreover, these data needs to be processed in real time because they represent real-world events, e.g. temperature measurements, traffic accidents, battlefield surveillance, pollution monitoring, etc.

- Network traffic management [38] and content based routing (multicasting) of time-critical data [37] are another uses of XML data streams. The main purpose of content based routing is to prioritize packets carrying time critical data (usually multimedia data, e.g. a video or audio stream) on time. That means to maintain constant delay and jitter of the transfer rate. In contrast, the data that are not time critical may experience varying transfer rate during the transmission. Hence, network routers need to read and process the data item content. XML is convenient for packet content description and further processing — routing.

- Information dissemination has been gaining popularity over past few years. It involves distribution of data to a large number of clients. It comprises delivery of personalized newspapers, entertainment content, stock tickers [23], etc. For example, NewsML [2] and NITF [3] are XML-based formats for marked-up news articles which are used by news agencies to broadcast their articles. XML allows encoding the document structure and use this information to create more precise user profiles which are not based on the keyword text search only. User is then being delivered only information regarding his or her interests. Well known information dissemination implementation is the filtering system XFilter [7]. XFilter matches incoming XML documents to high number of profiles of users registered in the system. User interests are specified as queries in XPath [5] language. In the same time, there is another filtering system called YFilter [16], which combines multiple queries into a single nondeterministic finite automaton eliminating redundant processing of similar queries.

- Many of current information systems are based on Service Oriented Architecture (SOA) [4]. The functions of the SOA applications are separated into distinct units, called *services*, which are loosely coupled and can be distributed over a network and reused by many applications. XML has been used extensively in SOA [20] as a format of messages sent between the services due to its non-proprietary and self-descriptive nature as well as for its flexible structure (in contrast to relational and object-oriented model). XML is also an excellent choice for defining views over distributed collections of both XML and relational data.

Streamed XML

The traditional DBMSs as they have been developed for more than 40 years are well explored during this time. It would be natural to use the theoretical results or DBMS itself in data stream management systems (DSMS). However, DBMS are not well suited for data stream processing. The main reason that different requirements are imposed on both systems [28], as shown in Table 1. When working on XML data, situation is even worse because there is no widely accepted DBMS for XML documents. There have been many works in progress proposing various methods for evaluation of XPath and XQuery queries on XML data streams as will be described later in this section. However, most of those implementations restrict expressive power of these languages.

Table 1. The main differences between traditional DBMS and DSMS

Comparison	DBMS	DSMS
data model	set of tuples	tuple sequences
database update types	arbitrary modifications	appends
query types	one-time, transient	persistent, continuous
query answer	exact	approximate
query plan	optimized at beginning	adaptive query plans
data access	arbitrary data access	one pass algorithms
computational resources	rich	limited

Under current state of development of general DSMS, many parts of them can be used in XML stream processing system, e.g. window processing, load shedding, etc. However, many open problems remain to be solved. Most of them are related to XML nature of the data streams — efficient querying, indexing, etc. Some of them are discussed further in Section *Querying XML data streams*.

XML data streams can be viewed in two ways. First, XML data stream is formed by one infinite XML document. Notice that the document can be streamed in document order [31] only. Due to memory limitations, we can not add (efficiently) new elements or subtrees to those branches of streamed XML document that were already constructed and abandoned — such stream could not be efficiently processed in any application as it would require storing whole stream. Although stream formed by one infinite document can be produced, this is not the case of real world applications which were described in detail in previous section. The second approach to XML data streams, which is found in real-world applications and is addressed in most of current papers, is a stream formed by a sequence of items, i.e. sequence of XML documents. Each data item in the stream is a valid standalone XML document, which is (usually) independent of other data items in the stream, e.g. hourly updated weather information or stock tickers. These data items are usually rather small XML documents and therefore they can be processed in memory without using any special algorithms for data streams. However, from a long-term point of view, this sequence of documents can be considered as a stream, e.g. a-year-history of an-hourly-updated-stock-ticker. As we can see from the previous example, such data streams cannot be processed using standard tools for XML querying as it would cause enormous memory usage. Unlike data streams in past, in XML streams structure of data items may vary significantly during time. Hence, more sophisticated algorithms are necessary to process XML streams efficiently.

Using algorithms which were developed for efficient processing of XML data streams can be advantageous on non-streamed XML documents as well. The streaming algorithms were designed to access data in document order and only one pass through the data is required. Lower memory requirements (compared to algorithms designed for non-streamed XML document processing) are also a benefit. Hence, algorithms for XML stream processing can be used on non-streamed XMLs in situations when memory usage and number of data accesses to XML documents are crucial.

Querying XML data streams

Queries on data streams can be divided into two basic groups — data mining and filtering.

Data mining comprises evaluating of very complex queries over a long time period. The query complexity may be caused among others by *blocking operators* [36]. Operator is blocking if its output can not be generated without the knowledge of the whole input stream, such as SQL operators EXISTS, NOT IN, EXCEPT. From algebraic point of view, blocking operators are non-monotonic expressions. If the operator is monotonic, e.g. SQL operators COUNT and SUM, then it is non-blocking. Notice that in traditional DBMS, blocking operators do not cause severe problems as the data are processed offline. However, in DSMS, blocking operators are irresolvable in traditional way. We have to employ window processing or approximate results of such operators. Approximation is also necessary when evaluating statistical queries, e.g. TOP-K, QUANTILE or COUNT DISTINCT, in case that all the data in the stream are needed to obtain value of such an operator. Approximate results are usually based on the estimate of statistical data distribution. In some cases approximation is even preferred to exact computations since it requires less amount of data to be stored as well as less time for query evaluation.

On the other hand, filtering the XML stream means to send to output only those (parts of) data items from the stream that match the filtering condition, e.g. XPath expression. In case that data transformation is needed before/after filtering, XQuery [6] is more suitable than XPath. In the following paragraphs, we present current results on XPath and XQuery processing on XML data streams.

XPath is a language for addressing parts of an XML document. XPath query evaluation is usually done in one of two ways — transforming XPath expression into a Deterministic or Nondeterministic Finite Automaton (DFA) or (NFA) (e.g. X-scan [24], XSQ [35], YFilter [16], AFilter [12], SPEX [34; 11]) or using parse-trees (e.g. TurboXPath [26], [13], [14], [9]). Streamed environment complicates the efficient evaluation. Consequently, current XPath query engines are limited to a subset of XPath queries (e.g. usage of certain predicates or functions is prohibited). According to many authors, the parse-trees should be more efficient than DFA or NFA transformation. However, it is hard to confront these systems directly as the implementations provide slightly different capabilities in XPath expression evaluation. In [21], authors present a method for the full support of XPath 1.0 in streamed environment using query rewriting. Another aproach to XPath evaluation is proposed in [17] where the streaming and traditional techniques (inndexing a nd join and query unnnesting) are combined.

XQuery is a query language designed to query collections of XML data. The semantics of XQuery is similar to SQL. Several systems for XQuery expressions evaluation over XML data streams have been proposed. In XSM [29], transducer networks are used; aggregation functions cannot be used and more than one traversal through the document may be required. In [32], authors present another method which uses query representation called *stream data flow graph* and a query transformation. It supports aggregate functions and guarantees single pass through the data when processing the query. The BEA streaming XQuery processor [18] was developed for high performance message processing, i.e. transforming XML data streams. Identification of the data neccessary for the query evaluation allows us to lower the memory requirements on the data caching as proposed in [10].

XQuery and XPath languages were not designed to be evaluated over streams. Therefore, it is not easy to implement them in a stream-fashion. Another approach to query XML streams is to propose brand-new querying language designed specifically for XML streams, e.g. XML Stream Attribute Grammars (XSAGs) [27]. XSAGs expressions can be evaluated in linear time in a streaming fashion. The memory requirements are bounded only by the depth of the streamed XML document. The query is evaluated in a single pass through the XML stream.

Sliding window processing

Most of the problems connected with blocking operators or bursts of data items on the input can be avoided using *sliding window* processing. During the query evaluation using sliding window, we do not access the whole data stream but only the most recent part of it. There are two types of sliding windows — a *count-based* sliding window stores the n most recent items; and a *time-based* sliding window stores only the items that have arrived in the last t time units. Parameters n and t are set up and fixed before the first data item arrives. The oldest data items expire from the window as new items arrive. In order to evaluate queries on the data items in a window efficiently, it is desirable to avoid redundant computations. Therefore, the window is associated a *synopsis*, which contains results that can be reused when re-evaluating query as window slides forward, i.e. new data arrive. Depending on the data items input rate and window size, synopsis can be exact or approximated. Very thorough study on a sliding window processing over data streams is provided in [22].

Data Integration

In order to create a business application based on SOA (introduced in Section *XML streams applications survey*), we reuse existing services. For example, we may want to create single interface to two bibliographic database systems which are available as services. However, individual services may provide data in different XML structures. One of the main issues in such data integration scenarios is the detection of (approximate) duplicates across the two sources [15], e.g. detection of the same books in our bibliographic example. The duplicate elimination can be a difficult task for autonomously managed sources[1] as their XML structures may be different. Thus, there is a need for similarity evaluation of the XML streams. Authors in [19] propose a method for XML data streams correlation in small space using

[1]Autonomously managed sources are becoming more important as the total number of services is increasing and human-driven integration is becoming unsustainable.

approximate structure and content matching which is based on tree-edit distance [8]. It can be used in conjunction with random sketching techniques to build small synopsis of large XML data which can then be used in approximated tree-edit distance computations. A survey of methods of (non-streamed) XML data similarity evaluation is provided in [33]. However, similarity evaluation on XML data streams is addressed in few papers.

Conclusion

The aim of this paper was to provide an introduction into XML data streams and present current results in XML data streams processing. We have shown that the XML streams differ from traditionally understood data streams, in particular in the data item complexity. The problems related to efficient querying have been pointed out on XPath and XQuery languages and blocking operators. We have also presented the real-world motivation to processing XML streams.

Open Issues

A brief summary of the open issues in XML data stream processing is provided in the following list:

Similarity evaluation is necessary for automated data integration, especially in SOA. On the structural level, similarity evaluation represents structural mapping; on the data level, similarity evaluation represents e.g. duplicate elimination.

Approximate aggregates Since certain aggregate functions (e.g. QUANTILE, TOP-K) cannot be evaluated accurately on a data stream, there is a need to approximate them. There has been a great body of work on this theme concerning general data streams (e.g. sketches and samples). Few of them have, however, been proposed for XML data stream processing.

Query decomposition In some cases, queries over an XML data stream can be evaluated in a distributed manner as it is inherent to underlying systems. Therefore, a query is decomposed so that part of it is evaluated on the underlying systems, whereas the rest of the query (e.g. an aggregation part) is evaluated on a higher level, i.e. stream management system. For example, in sensor networks, such an approach may improve query evaluation time and reduce amount of transmitted data.

Varying data item arrival rate and bursts of data do not present an issue when processing only one data stream. However, in case that join of two or more data streams is carried out, the data caching is necessary to handle afore-mentioned perturbations in data item arrival. Existing approaches, i.e. window processing or results of queueing theory, may be applied.

References

[1] Extensible Markup Language (XML) 1.0 (Fourth Edition). `http://www.w3.org/TR/xml`. Dated 6 August 2006, last update 29 September 2006; accessed 25 May 2008.

[2] NewsML: News Markup Language. `http://www.newsml.org/`. Dated 2008; accessed 25 May 2008.

[3] NITF: News Industry Text Format. `http://www.nitf.org/`. Dated 2007; accessed 25 May 2008.

[4] SOA - Documents - Document details. `http://opengroup.org/projects/soa/doc.tpl?gdid=10632`. Dated 2 June 2006, last update 8 June 2006; accessed 25 May 2008.

[5] XML Path Language (XPath) Version 1.0. `http://www.w3.org/TR/xpath`. Dated 16 November 1999; accessed 25 May 2008.

[6] XQuery 1.0: An XML Query Language. `http://www.w3.org/TR/xquery/`. Dated 23 January 2007; accessed 25 May 2008.

[7] Mehmet Altinel and Michael J. Franklin. Efficient Filtering of XML Documents for Selective Dissemination of Information. In *VLDB '00: Proceedings of the 26th International Conference on Very Large Data Bases*, pages 53–64, San Francisco, CA, USA, 2000. Morgan Kaufmann Publishers Inc.

[8] Alberto Apostolico and Zvi Galil, editors. *Pattern matching algorithms.* Oxford University Press, Oxford, UK, 1997.

[9] Ziv Bar-Yossef, Marcus Fontoura, and Vanja Josifovski. On the memory requirements of XPath evaluation over XML streams. In *PODS '04: Proceedings of the twenty-third ACM SIGMOD-SIGACT-SIGART symposium on Principles of database systems*, pages 177–188, New York, NY, USA, 2004. ACM.

[10] Alexandru Berlea. On-the-fly tuple selection for XQuery. In *XIME-P '07: Proceedings of the 4th international workshop on XQuery implementation, experience and perspectives*, pages 1–6, New York, NY, USA, 2007. ACM.

[11] Francois Bry, Fatih Coskun, Serap Durmaz, Tim Furche, Dan Olteanu, and Markus Spannagel. The XML Stream Query Processor SPEX. In *ICDE '05: Proceedings of the 21st International Conference on Data Engineering*, pages 1120–1121, Washington, DC, USA, 2005. IEEE Computer Society.

[12] K. Selçuk Candan, Wang-Pin Hsiung, Songting Chen, Junichi Tatemura, and Divyakant Agrawal. AFilter: adaptable XML filtering with prefix-caching and suffix-clustering. In *VLDB '06: Proceedings of the 32nd international conference on Very large data bases*, pages 559–570. VLDB Endowment, 2006.

[13] Chee-Yong Chan, Pascal Felber, Minos Garofalakis, and Rajeev Rastogi. Efficient filtering of XML documents with XPath expressions. *The VLDB Journal*, 11(4):354–379, December 2002.

[14] Yi Chen, Susan B. Davidson, and Yifeng Zheng. An Efficient XPath Query Processor for XML Streams. In *ICDE '06: Proceedings of the 22nd International Conference on Data Engineering*, page 79, Washington, DC, USA, 2006. IEEE Computer Society.

[15] Tamraparni Dasu and Theodore Johnson. *Exploratory Data Mining and Data Cleaning.* John Wiley & Sons, Inc., New York, NY, USA, 2003.

[16] Yanlei Diao, Peter M. Fischer, Michael J. Franklin, and Raymond To. YFilter: Efficient and Scalable Filtering of XML Documents. In *ICDE*, pages 341–342, 2002.

[17] Mary Fernández, Philippe Michiels, Jérôme Siméon, and Michael Stark. XQuery Streaming á la Carte. *ICDE*, pages 256–265, 2007.

[18] Daniela Florescu, Chris Hillery, Donald Kossmann, Paul Lucas, Fabio Riccardi, Till Westmann, Michael J. Carey, Arvind Sundararajan, and Geetika Agrawal. The BEA/XQRL streaming XQuery processor. In *VLDB'2003: Proceedings of the 29th international conference on Very large data bases*, pages 997–1008. VLDB Endowment, 2003.

[19] Minos Garofalakis and Amit Kumar. XML stream processing using tree-edit distance embeddings. *ACM Trans. Database Syst.*, 30(1):279–332, 2005.

[20] Minos N. Garofalakis, Ioana Manolescu, Marco Mesiti, George A. Mihaila, Ralf Schenkel, Bhavani M. Thuraisingham, and Vasilis Vassalos. What's Next in XML and Databases? In Wolfgang Lindner, Marco Mesiti, Can Türker, Yannis Tzitzikas, and Athena Vakali, editors, *EDBT Workshops*, volume 3268 of *Lecture Notes in Computer Science*, pages 318–324. Springer, 2004.

[21] Pierre Genevès and Kristoffer Rose. Compiling XPath for streaming access policy. In *DocEng '05: Proceedings of the 2005 ACM symposium on Document engineering*, pages 52–54, New York, NY, USA, 2005. ACM.

[22] Lukasz Golab. *Sliding Window Query Processing Over Data Streams.* PhD in Computer Science, University of Waterloo, Waterloo, Ontario, 2006.

[23] Lukasz Golab and M. Tamer Özsu. Issues in data stream management. *SIGMOD Rec.*, 32(2):5–14, 2003.

[24] Zachary G. Ives, Alon Y. Halevy, and Daniel S. Weld. An XML query engine for network-bound data. *The VLDB Journal*, 11(4):380–402, 2002.

[25] Shawn R. Jeffery, Minos Garofalakis, and Michael J. Franklin. Adaptive cleaning for RFID data streams. In *VLDB '06: Proceedings of the 32nd international conference on Very large data bases*, pages 163–174. VLDB Endowment, 2006.

[26] Vanja Josifovski, Marcus Fontoura, and Attila Barta. Querying XML streams. *The VLDB Journal*, 14(2):197–210, 2005.

[27] Christoph Koch and Stefanie Scherzinger. Attribute grammars for scalable query processing on XML streams. *The VLDB Journal*, 16(3):317–342, 2007.

[28] Nick Koudas and Divesh Srivastava. Data stream query processing: a tutorial. In *VLDB'2003: Proceedings of the 29th international conference on Very large data bases*, pages 1149–1149. VLDB Endowment, 2003.

[29] Bertram Ludäscher, Pratik Mukhopadhyay, and Yannis Papakonstantinou. A transducer-based XML query processor. In *VLDB'02: Proceedings of the 28th international conference on Very Large Data Bases*, pages 227–238. VLDB Endowment, 2002.

[30] Samuel Madden and Michael J. Franklin. Fjording the Stream: An Architecture for Queries Over Streaming Sensor Data. In *ICDE '02: Proceedings of the 18th International Conference on Data Engineering, 26 February – 1 March 2002, San Jose, CA*, pages 555–566. IEEE Computer Society, 2002.

[31] Moad Maghaydah and Mehmet A. Orgun. Labeling XML Nodes in RDBMS. In Heng Tao Shen, Jinbao Li, Minglu Li, Jun Ni, and Wei Wang, editors, *APWeb Workshops*, volume 3842 of *Lecture Notes in Computer Science*, pages 122–126. Springer, 2006.

[32] Philippe Michiels. XQuery Optimization. *Proceedings of the VLDB 2003 PhD Workshop, Co-located with the 29th International Conference on Very Large Data Bases. Berlin, September 12-13*, 2003.

[33] Irena Mlýnková and Jaroslav Pokorný. Exploitation of Similarity and Pattern Matching in XML Technologies. Technical Report 13, Charles University, Prague, Czech Republic, 2006.

[34] Dan Olteanu, Tobias Kiesling, and Francois Bry. An evaluation of regular path expressions with qualifiers against XML streams. In *Data Engineering, 2003. Proceedings. 19th International Conference on*, pages 702–704, 2003.

[35] Feng Peng and Sudarshan S. Chawathe. XSQ: A streaming XPath engine. *ACM Trans. Database Syst.*, 30(2):577–623, 2005.

[36] Jaroslav Pokorný and Václav Snášel. Zpracování proudů dat. In Peter Vojtáš and Tomáš Skopal, editors, *Datakon 2006*, pages 61–76. Masarykova Univerzita Brno, 2006.

[37] Alex C. Snoeren, Kenneth Conley, and David K. Gifford. Mesh-based content routing using XML. *SIGOPS Oper. Syst. Rev.*, 35(5):160–173, 2001.

[38] Yunyue Zhu and Dennis Shasha. Efficient elastic burst detection in data streams. In *KDD '03: Proceedings of the ninth ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 336–345, New York, NY, USA, 2003. ACM.