ENHANCING XML-CACHE EFFICIENCY BY XML COMPRESSION

Stefan Böttcher, Lars Fernhomberg and Rita Hartel
Univeristy of Paderborn, Computer Science, Fürstenallee 11, 33102 Paderborn, Germany

Keywords: XML databases, Caching, Compression.

Abstract: Whenever a client with limited resources frequently has to retrieve large parts of a huge XML document that is stored on a remote web server, data exchange from the server to the client in combination with restricted bandwidth may become a serious bottleneck. We present an approach that combines the advantages of caching with the advantages of query-capable and updatable XML compression. We provide a performance evaluation that demonstrates that the combination of the two techniques – caching and compression – yields a benefit in terms of less data volume to be transferred as well as in terms of a reduced answer time for the user. The performance evaluation demonstrates that combining both technique yields an even stronger benefit than each technique alone.

1 INTRODUCTION

1.1 Motivation

In the last years, more and more standards for data exchange have evolved that are based on the flexible hierarchical data format XML. Examples for such data formats are SEPA (Single European Payment Area) for the financial sector or OTA (OpenTravel Alliance) for the flight sector.

Whereas previously, the clients that participated in internet-based applications got stronger resources, now a reverse development can be observed. More and more small mobile devices (like e.g. PDAs and smartphones) that come with restricted internal resources and restricted connection bandwidth participate in web applications.

Therefore, we consider scenarios, in which client applications need to access huge XML documents that are provided on remote web servers, and where the data transfer volume or the data transfer time from the server to the client is a bottleneck.

To overcome this bottleneck, there exist two different technologies: caching previously downloaded data on the one hand and compressing transferred data on the other hand.

A seamless integration of both techniques on the client side is a challenging task as compression combines similar structured data whereas caching isolates data in the cache.

Our approach combines both techniques as follows: we compress the data on the web server and process and transfer them in the compressed format to the client. On the client side, the data is then cached in compressed format, such that it can be used for answering not only identical but also similar queries without access to the web server.

1.2 Contributions

This paper proposes an approach for integrating caching and compression in order to reduce data transfer volume and time and that combines the following properties:

- It compresses the structure and the constant data of the XML document on server side.
- It transfers the data in a compressed way to the client and thereby saves transfer volume and transfer time.
- It transfers to the client only that part of the data that is needed to answer the user query and that is not yet stored within the compressed cached client data.
- The compression technique used for compressing the data can be exchanged. Our approach contains a generic caching framework for compressed data that generalizes the tasks that are common to all compression techniques to be used, such that only a simple interface has to be implemented by each

Böttcher S., Fernhomberg L. and Hartel R..
ENHANCING XML-CACHE EFFICIENCY BY XML COMPRESSION .
DOI: 10.5220/0003543004300800087
In *Proceedings of the 7th International Conference on Web Information Systems and Technologies* (WEBIST-2011), pages 80-87
ISBN: 978-989-8425-51-5

concrete compression technique before it can be used.

- We provide a technique for integrating compressed fragments on the client side into a single compressed fragment without total decompression of the cached fragments.

We have implemented our system with two different compression techniques and comparatively evaluated our system with querying uncompressed data that is not kept in the cache, querying uncompressed, cached data, and querying compressed data that is not kept in the cache. Our results show that the combination of caching and compression outperforms all the other techniques in terms of transfer volume and in terms of total query evaluation time considering data rates of up to that of UMTS.

1.3 Problem Description

The problem investigated in this paper is how to improve caching of answers to XPath queries in client-server web information systems in the following aspects: first, the overall data exchange volume between the web information source and the client shall be reduced, and second, the time needed by the client to get the results of a query from a remote server shall be reduced.

We follow the principle of compensation queries (Mandhani & Suciu, 2005), i.e., we calculate compensation queries from a given new query and old queries, the results of which are already contained in the cache. A compensation query of a query Q is a query Q' that, applied to the view V of the database that is represented by the cache, calculates the same query results as Q applied to the data on the server. However, beyond the approach of (Mandhani & Suciu, 2005), we consider a much broader subset of XPath including the search or navigation along all the XML forward axes. Thereby, our approach to caching supports a significantly wider field of web applications.

In the current approach, we do not consider, how to handle updates of server data of which a copy is stored in the client cache, but we expect that our current concept does not prevent outdated caches to be managed.

1.4 Paper Organization

This paper is organized as follows: In Section 2, we explain the key ideas of the general solution, including the compression techniques implemented in our caching framework and the mechanisms used for performing query evaluation on the compressed XML representations. Section 3 describes our performance evaluation and the evaluation results. Section 4 describes related work, and Section 5 contains a summary and the conclusions.

2 THE CONCEPT

2.1 The Basic Idea

Our framework for caching compressed XML data consists of a server and a client. The main idea of our caching approach is that the server only sends that data that is needed by the client to answer the query and that is not already contained in the client's cache. The data is transferred to the client and stored on the client in a compressed format. A second requirement is that the server is stateless, i.e., that it does not store the current state of the client's caches, as this would lead to a too large storage overhead on the server side.

In order to transfer only new data that has not yet been cached, the server has to find out whether or not a query requires to access data not already stored in the client's cache. Therefore, the server contains the compressed XML document and a query cache, where itself stores results to previously answered queries. When the server receives a query from any client, it assigns an ID to the query (either a new one, or it looks up the ID of the query if it was answered for another client before) and sends the ID of the query together with the answers to the client. As the server is stateless, i.e., it does not maintain any state information on its clients, the client sends in addition to the query to be answered a list of query IDs that describe the queries already stored in its cache. With the help of the query IDs and the global server query cache, the server then can reconstruct the content of the client's cache. Finally, the server computes the result R of the client's query and transfers the difference between the result R and the client's cache's content, i.e., only these nodes of the result that were not already contained in the cache, to the client. In addition, the server stores the client's query, the ID assigned to the query and the result R in its query cache.

When the client receives the answer to its query, it integrates the compressed XML fragments into its cache. It then can be sure, that applying its query to its cache will yield the same result as applying this query to the server document.

Figure 1 shows the application flow of the approach presented in this paper.

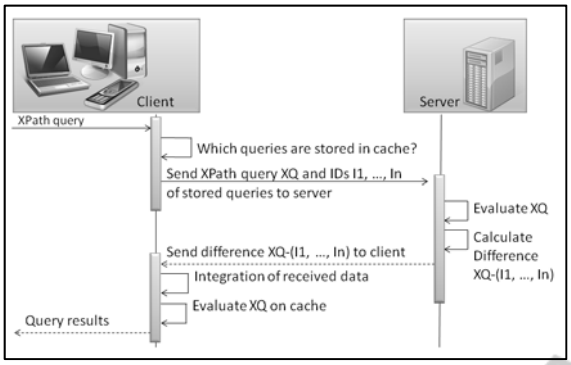

Figure 1: Application flow of our caching approach.

In order to integrate the transferred data, we use the updateable XML numbering scheme ORDPATH (O'Neil, O'Neil, Pal, Cseri, Schaller, & Westbury, 2004) that allows for two nodes specified by their ORDPATH ID to determine, in which XML axis relation they are to each other, e.g., whether node A is a following-sibling or a descendant of node B. Depending on the concrete requirements, other XML numbering schemes like DLN (Böhme & Rahm, 2004), or DDE (Xu, Ling, Wu, & Bao, 2009) could be chosen.

2.2 XML Compression

Our caching approach is designed in such a way that it can work with different XML compression techniques. The requirements to these techniques are that query evaluation and updates are possible on the compressed data directly, i.e., without a prior decompression.

Tasks that are common for all compressed representations of the XML document are isolated into a generic part. Therefore, the compression techniques being used for managing the client's cache do only have to implement a simple interface covering functions like basic navigation via first-child, next-sibling, and parent, as well as basic update functionalities in order to integrate the new results into the cache. Furthermore, the compression technique being used at the server-side additionally has to support computing the difference of two compressed node lists (with node unique node IDs).

Currently, we have implemented two different compression techniques within our caching framework: Succinct compression and DAG compression.

2.2.1 Succinct XML Compression

Succinct XML compression is an XML compressor that separates the XML constants from the XML element names and the attribute names and from the nesting of start tags and end tags, i.e. the compressed document structure of an XML document consists of the following parts:

1. A bit stream representing the tree structure of the element nesting in the XML tree, without storing any label information. In the bit stream, each start-tag is represented by a '1'-bit and each end-tag is represented by a '0'-bit.
2. Inverted element lists, containing a mapping of element and attribute names to '1'-bit positions within the bit stream.
3. Constant lists containing the text values and attribute values

Succinct compression can handle unbounded input streams and huge files and it allows query evaluation and updates directly on the compressed data.

A variant of succinct compression has been presented in (Böttcher, Hartel, & Heinzemann, Compressing XML Data Streams with DAG+BSBC, 2008).

2.2.2 DAG-based XML Compression

A variant of DAG-based XML compression has been presented in (Buneman, Grohe, & Koch, 2003).

The constant data – i.e., text nodes and attribute values – are separated from the XML structure – i.e., element and attribute nodes – and compressed separately. Constant data is grouped according to their parent element name and each group of constant data is compressed via gzip.

DAG-based XML compression follows the concept of sharing repeated sub-trees in the XML structure. Whenever a sub-tree occurs more than once within the document tree structure, all but the first occurrence are removed from the document tree and are replaced by a pointer to the first occurrence.

The minimal DAG of an XML document tree structure can be calculated by a single pre-order pass through the document tree (e.g., by reading a linear SAX stream) with the help of a hash table, where all first occurrences of a sub-tree are stored.

As the DAG is stored in form of DAG nodes, where each node contains an address, the node label and pointers to its first-child and its next-sibling, the DAG compression allows for a very efficient basic navigation (similar to a DOM representation). In contrast, it does not reach as strong compression ratios as e.g. Succinct compression.

Besides query evaluation directly on the compressed data, DAG-based XML compression allows compressing unbounded and huge XML files as well as updating the compressed data directly.

2.3 Query Evaluation

The client and the server use an XPath evaluator based on a reduced instruction set that has to be provided by each compression technique. In order to simplify the presentation, we assume a simplified data model of an XML tree, where all nodes – no matter whether they are element, attribute or text nodes – are valid answers to the basic axes fc and ns:

- fc: Returns the first-child of the current context node ccn
- ns: Returns the next-sibling of the current context node ccn
- label: Returns the label of the current context node ccn if ccn is an element, the attribute name if ccn represents an attribute, the text value if ccn represents a text node, or an attribute value, if ccn represents an attribute value.
- parent: Returns the parent of the current context node ccn.
- node type: Returns the node type (i.e., either element, attribute or text node) of the current context node

We then use a technique like e.g. the one presented in (Böttcher & Steinmetz, Evaluating XPath Queries on XML Data Streams, 2007) to evaluate XPath path queries on the compressed XML representation providing the reduced instruction set. Based on such a generic XPath evaluation technique, our approach allows to evaluate XPath queries containing the axes self, child, descendant, descendant-or-self, following-sibling, following, attribute and text as well as predicate filters with logical expressions containing comparisons of paths to constants. If additionally the backward axes ancestor, ancestor-or-self, preceding-sibling and preceding-sibling are required, this can be provided by a preprocessing step described in (Olteanu, Meuss, Furche, & Bry, 2002).

3 EVALUATIONS

3.1 Performance Evaluation Environment

To evaluate the performance of the idea, a prototype has been developed and was tested using the XMark benchmark (Schmidt, Waas, Kersten, Carey, Manolescu, & Busse, 2002). The prototype uses Java 6 Update 18 and was optimized for compression efficiency.

Each benchmark was performed ten times to account for external effects like operating system influences. The system used for the benchmarks was equipped with an AMD Phenom 9950 (Quad Core, each core runs with 2.6 GHz) and 4 GB main memory and running Windows 7 (64 Bit). Despite having a multi core processor, the prototype is single-threaded and does not use the complete system capacity.

The benchmarks were performed on different XML documents, generated with increasing XMark scale factors, to determine the effect of different document sizes on the prototype. Table 1 shows the XMark scale factors being used and the resulting XML document size.

The generated documents were queried with a set of 22 different XPath queries that produce a mix of cache hits and cache misses. Queries 1-10 are closely related to the “XPath-A” benchmark of XPathMark (Franceschet, 2005) and are modified to overcome limitations in the used XPath evaluator. Queries 11-22 are selected to give an impression of (partial) cache-hits and cache-misses.

Table 1: Used XMark scale factors.

XMark scale factor	XML document size
0.000	~ 35 kB
0.001	~ 154 kB
0.002	~ 275 kB
0.004	~ 597 kB
0.008	~ 1.178 kB
0.016	~ 2.476 kB
0.032	~ 4.899 kB
0.064	~ 9.514 kB

3.2 Performance Results

Figure 3 shows the behavior of the prototype for each measured combination (Uncompressed, DAG or Succinct storage with or without cache) with growing document size. When the scale factor doubles, the transmitted data for each graph also doubles, meaning that the total amount of transmitted data scales linearly with the input data.

With an active cache, less data has to be transmitted in all cases. The additional use of compression techniques reduces the transferred data even further. Comparing the total data that is transferred (queries and query results including the compressed structure and the compressed constants of the XML representation), the best case scenario (Succinct compression with active cache) uses only about 16% of the data volume being used by worst case scenario (no compression and no cache).

The following performance analysis is based on benchmark results of the XML document that was

generated using a scale factor of 0.064, unless otherwise noted. The results are consistent through all tested XML documents.

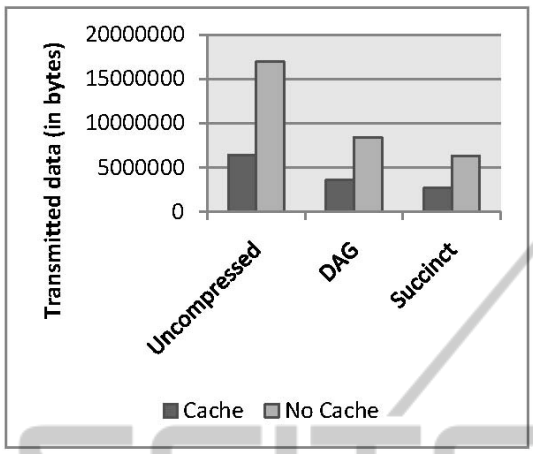

Figure 2: Total transmitted data.

Figure 2 shows an aggregation of the total transmitted data of all 22 queries. The measured value consists of the data transmitted to the server (query information) and the result of the server (XML fragment and numbering information). For each compression technique, we measured twice with an activated and with a deactivated cache.

Figure 4 shows the simulation of a real world scenario, considering different data rates on the channel between client and server, where the overall query evaluation time is calculated consisting of

- transferring data from client to server,
- evaluating query on server side,
- transferring data from server to client,
- integrating data into client's cache, and
- evaluating the query on the client's cache,

The left-most group of results in Figure 4:,denoted as framework duration, is given as a lower bound for comparison purposes and includes only the time needed to evaluate the query on the server, to build the query difference on the server, to integrate the data into the client's cache and to evaluate the query on the client's cache, i.e. without the time needed to transfer data from server to client and from client to server.

Figure 4 shows that using the combination of caching and succinct compression consumes less time than using caching or compression alone up to a channel speed of about 128 kbit/s. Independent of the available data rate, the combination of caching and succinct compression transfers less data than using caching or compression alone. Using the combination of caching and DAG compression delivers speed improvements of up to 384 kbit/s, in comparison to using caching or using compression alone. Furthermore, in terms of total time, using caching only for uncompressed data outperforms the immediate query evaluation on the server up to a data rate of about 2 Mbit/s.

A breakdown of the total duration into each subtask (computing XPath result on client and server, identification of missing nodes on the client, integration of missing nodes in the client cache and data transfer from client to server and from server to client) shows that the XPath evaluation on server side as well as on the client side consumes the greatest part of the time. Depending on the compression algorithm, the client-side data integration can also lead to noticeable runtimes, because it may require renumbering or rearranging of existing data structures. The runtimes of all other operations are comparatively small and do not contribute much to the total runtime.

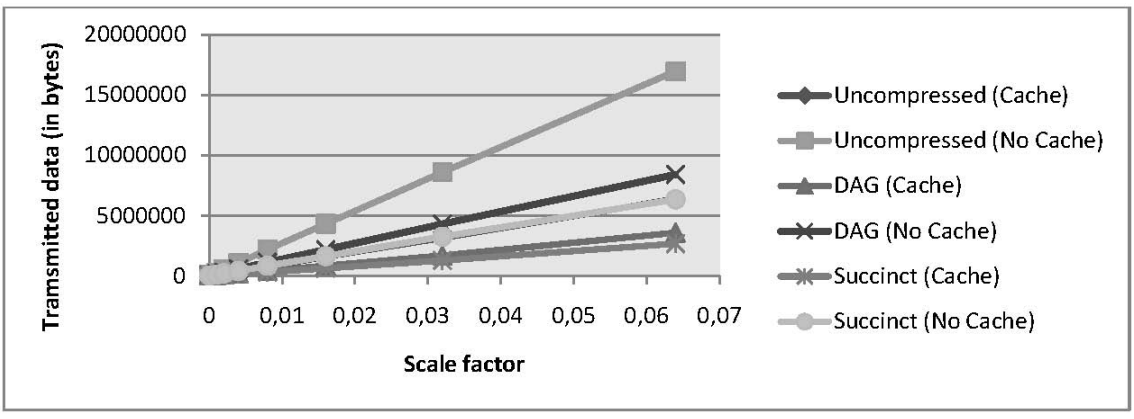

Figure 3: Total aggregated sent data for growing documents.

3.3 Evaluation Summary

Our evaluation has compared transferred data volume and evaluation time being used in different combinations of caching and different compression techniques in comparison to caching and compression alone and in comparison to the immediate query evaluation on server site. Our evaluation has shown that using the combination of caching and compression not only reduces the data volume that is transferred from client to server and from server to client, but especially for data rates up to the data rate of UMTS reduces the total time that is needed to answer the query.

Clients with a limited bandwidth connection to the server will benefit from using the compressed XML cache as it is presented in this paper, whereas clients that are connected to the server via a high-speed connection might deactivate the cache and might request the query results from the server directly.

Our evaluation was using an unlimited cache size on the client side, thereby ignoring the following further advantage of the combination of compression and caching.

As the compressed data is significantly smaller than the non-compressed data, a cache of a fixed size can hold more XML data fragments in compressed format than in non-compressed format.

Therefore, for limited cache size, we expect that the number of cache misses is significantly smaller for the combination of caching and XML compression than for caching only. In other words, if the cache size is limited, this leads to even stronger benefits of the combination of caching and XML compression over caching only.

4 RELATED WORK

Although both, web data caching and XML compression, contribute to a reduction of the data transfer from server to client, the fields of web data caching and XML compression have mostly been investigated independently of each other.

There has been a lot of work in XML compression, some of which does not support query processing on compressed data, e.g. (Liefke & Suciu, 2000), and most of which support querying compressed data, but not querying cached compressed data, e.g. (Busatto, Lohrey, & Maneth, 2005), (Cheng & Ng, 2004), (Ng, Lam, Wood, & Levene, 2006), (Skibinski & Swacha, 2007), (Zhang, Kacholia, & Özsu, 2004).

Contributions to the field of caching range from concepts of querying and maintaining incomplete data, e.g. (Abiteboul, Segoufin, & Vianu, 2001), over caching strategies for mobile web clients, e.g.

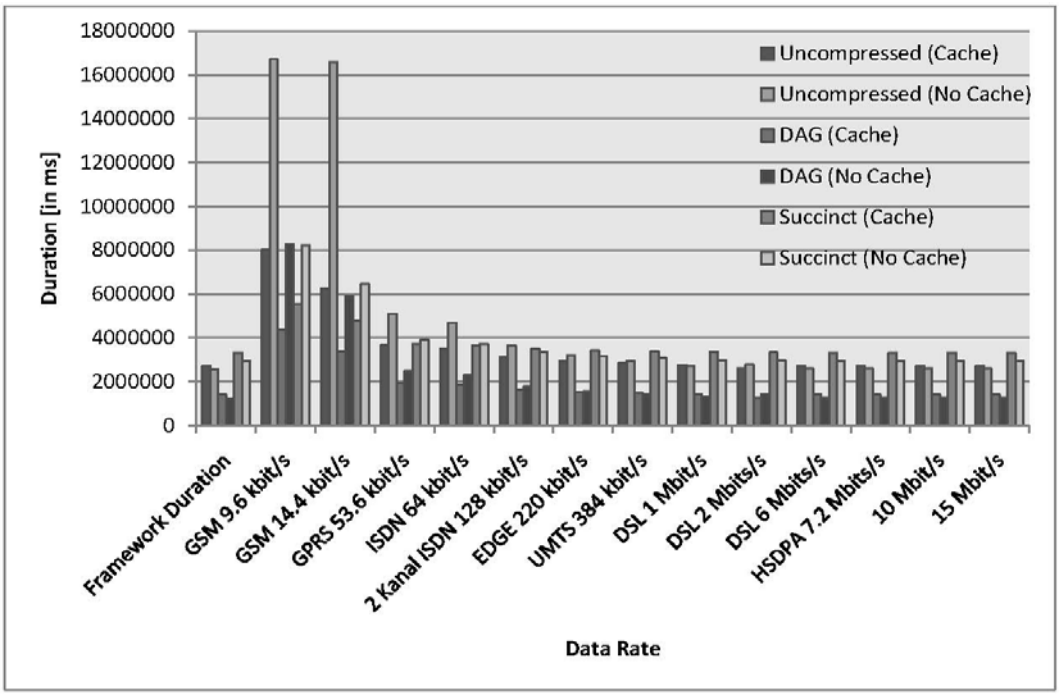

Figure 4: Aggregated total duration (framework duration and simulated transmission duration).

(Böttcher & Türling, 2004), caching strategies for distributed caching (Obermeier & Böttcher, 2008), to caching strategies based on frequently accessed tree patterns, e.g. (Yang, Lee, & Hsu, 2003). In comparison, our approach allows for XPath queries using filters and comparisons with constants even on compressed cached XML.

Different approaches have been suggested for checking whether an XML cache can be used for answering an XPath query. On the one hand, there are contributions, e.g. (Balmin, Özcan, Beyer, Cochrane, & Pirahesh, 2004), (Mandhani & Suciu, 2005), (Xu & Özsoyoglu, 2005), that propose to compute a compensation query. These approaches can also be used on compressed XML data, but they are NP-hard already for very small sub-classes of XPath. On the other hand, containment tests and intersection tests for tree pattern queries have been proposed, and could in principle be used for deciding whether a given XPath query can be executed on the cached data locally. However, such intersection tests and containment tests are NP-hard for rather small subsets of XPath expressions (Benedikt, Wenfei, & Geerts, 2005), (Hidders, 2003). In comparison, our approach uses a fast difference computation that can be done within a single scan through the compressed XML file.

In contrast to (Böttcher & Hartel, CSC: Supporting Queries on Compressed Cached XML, 2009), where the whole compressed structure was loaded into the cache and only the text values needed to answer the query was requested from the server, our approach stores only that part of the structure in its cache that was delivered from the server in order to answer previous queries.

In comparison to all other approaches, our technique is to the best of our knowledge the only strategy that combines the following advantages: it caches compressed documents – whereas the compression technique can be exchanged – and thereby reduces data transfer volume and data transfer time in comparison to caching non-compressed XML data and in comparison to transferring non-compressed or compressed XML data without caching the results.

5 SUMMARY AND CONCLUSIONS

Whenever the data exchange with XML-based information sources is a bottleneck, it is important to reduce the amount of exchanged XML data. Our approach combines two reduction techniques for exchanged data, i.e. caching and XML compression. Additionally, we have provided a performance evaluation that shows that a significant reduction in data transfer volume and transfer time can be achieved by our approach in comparison to caching only or to using only compression.

Altogether, our approach provides the following advantages:

- It provides a technique to combine caching and compression in such a way that any unnecessary decompression of data is avoided. The document is stored in a compressed format on the server, the results are transferred in a compressed format from server to client and all results are stored and integrated in a compressed format in the client's cache.
- The combination of compression and caching yields a reduced data transfer volume in contrast to each technique alone.
- Furthermore, the combination of compression and caching yields a reduced data transfer time for data rates up to that of UMTS in contrast to each technique alone.

We assume it to be an interesting task to combine our caching framework with other compression techniques to find a compressions technique that is most suitable in terms of strong compression capabilities and fast query evaluation such that the benefit in total evaluation time will be highest.

Finally, we assume that our approach will show even stronger benefits if it is combined with real world assumptions like a limited cache size.

REFERENCES

Abiteboul, S., Segoufin, L., & Vianu, V. (2001). Representing and Querying XML with Incomplete Information. *Proceedings of the Twentieth ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems, May 21-23, 2001, Santa Barbara, California, USA.*

Balmin, A., Özcan, F., Beyer, K. S., Cochrane, R., & Pirahesh, H. (2004). A Framework for Using Materialized XPath Views in XML Query Processing. *(e)Proceedings of the Thirtieth International Conference on Very Large Data Bases* (pp. 60-71). Toronto, Canada: Morgan Kaufmann.

Benedikt, M., Wenfei, F., & Geerts, F. (2005). XPath satisfiability in the presence of DTDs. *Proceedings of the Twenty-fourth ACM SIGACT-SIGMOD-SIGART* (pp. 25-36). Baltimore, Maryland, USA: ACM.

Böhme, T., & Rahm, E. (2004). Supporting Efficient Streaming and Insertion of XML Data in RDBMS. In Z. Bellahsene, & P. McBrien (Ed.), *DIWeb2004, Third*

International Workshop on Data Integration over the Web, (pp. 70-81). Riga, Latvia.

Böttcher, S., & Hartel, R. (2009). CSC: Supporting Queries on Compressed Cached XML. In A. Bouguettaya, & X. Lin (Ed.), *Database Technologies 2009, Twentieth Australasian Database Conference (ADC 2009)* (pp. 153-160). Wellington, New Zealand: CRPIT.

Böttcher, S., & Steinmetz, R. (2007). Evaluating XPath Queries on XML Data Streams. *Data Management. Data, Data Everywhere, 24th British National Conference on Databases, BNCOD 24* (pp. 101-113). Glasgow, UK: Springer.

Böttcher, S., & Türling, A. (2004). Caching XML Data on Mobile Web Clients. *Proceedings of the International Conference on Internet Computing, IC '04* (pp. 150-156). Las Vegas, Nevada, USA: CSREA Press.

Böttcher, S., Hartel, R., & Heinzemann, C. (2008). Compressing XML Data Streams with DAG+BSBC. In J. Cordeiro, S. Hammoudi, & J. Filipe (Ed.), *Web Information Systems and Technologies, 4th International Conference, WEBIST 2008, Revised Selected Papers, Lecture Notes in Business Information Processing* (pp. 65-79). Funchal, Madeira, Portugal: Springer.

Buneman, P., Grohe, M., & Koch, C. (2003). Path Queries on Compressed XML. *Proceedings of 29th International Conference on Very Large Data Bases* (pp. 141-152). Berlin, Germany: Morgan Kaufmann.

Busatto, G., Lohrey, M., & Maneth, S. (2005). Efficient Memory Representation of XML Documents. *Database Programming Languages, 10th International Symposium, DBPL 2005* (pp. 199-216). Trondheim, Norway: Springer.

Cheng, J., & Ng, W. (2004). XQzip: Querying Compressed XML Using Structural Indexing. *Advances in Database Technology - EDBT 2004, 9th International Conference on Extending Database Technology* (pp. 219-236). Heraklion, Crete, Greece: Springer.

Franceschet, M. (2005). XPathMark: An XPath Benchmark for the XMark Generated Data. In S. Bressan, S. Ceri, E. Hunt, Z. G. Ives, Z. Bellahsene, M. Rys, et al. (Ed.), *Database and XML Technologies, Third International XML Database Symposium, XSym 2005*, (pp. 129-143). Trondheim, Norway.

Hidders, J. (2003). Satisfiability of XPath Expressions. *Database Programming Languages, 9th International Workshop, DBPL 2003* (pp. 21-36). Potsdam, Germany: Springer.

Liefke, H., & Suciu, D. (2000). XMILL: An Efficient Compressor for XML Data. *Proceedings of the 2000 ACM SIGMOD International Conference on Management of Data* (pp. 153-164). Dallas, Texas, USA: ACM.

Mandhani, B., & Suciu, D. (2005). Query Caching and View Selection for XML Databases. In K. Böhm, C. S. Jensen, L. M. Haas, M. L. Kersten, P.-A. Larson, & B. C. Ooi (Ed.), *Proceedings of the 31st International Conference on Very Large Data Bases* (pp. 469-480). Trondheim, Norway: ACM.

Ng, W., Lam, W. Y., Wood, P. T., & Levene, M. (2006). XCQ: A queriable XML compression system. *Knowl. Inf. Syst.* , 421-452.

Obermeier, S., & Böttcher, S. (2008). XML fragment caching for large-scale mobile commerce applications. *Proceedings of the 10th International Conference on Electronic Commerce, ICEC 2008* (p. 26). Innsbruck, Austria: ACM.

Olteanu, D., Meuss, H., Furche, T., & Bry, F. (2002). XPath: Looking Forward. In A. B. Chaudhri, R. Unland, C. Djeraba, & W. Lindner (Ed.), *XML-Based Data Management and Multimedia Engineering - EDBT 2002 Workshops, EDBT 2002 Workshops XMLDM, MDDE, and YRWS* (pp. 109-127). Prague, Czech Republic: Springer.

O'Neil, P. E., O'Neil, E. J., Pal, S., Cseri, I., Schaller, G., & Westbury, N. (2004). ORDPATHs: Insert-Friendly XML Node Labels. In G. Weikum, A. C. König, & S. Deßloch (Ed.), *Proceedings of the ACM SIGMOD International Conference on Management of Data* (pp. 903-908). Paris, France: ACM.

Schmidt, A., Waas, F., Kersten, M. L., Carey, M. J., Manolescu, I., & Busse, R. (2002). XMark: A Benchmark for XML Data Management. *VLDB 2002, Proceedings of 28th International Conference on Very Large Data Bases*, (pp. 974-985). Hong Kong, China.

Skibinski, P., & Swacha, J. (2007). Combining Efficient XML Compression with Query Processing. *Advances in Databases and Information Systems, 11th East European Conference, ADBIS 2007* (pp. 330-342). Varna, Bulgaria: Springer.

Xu, L., Ling, T. W., Wu, H., & Bao, Z. (2009). DDE: from dewey to a fully dynamic XML labeling scheme. In U. Cetintemel, S. B. Zdonik, D. Kossmann, & N. Tatbul (Ed.), *Proceedings of the ACM SIGMOD International Conference on Management of Data, SIGMOD 2009* (pp. 719-730). Providence, Rhode Island, USA: ACM.

Xu, W., & Özsoyoglu, Z. M. (2005). Rewriting XPath Queries Using Materialized Views. *Proceedings of the 31st International Conference on Very Large Data Bases* (pp. 121-132). Trondheim, Norway: ACM.

Yang, L. H., Lee, M.-L., & Hsu, W. (2003). Efficient Mining of XML Query Patterns for Caching. *Proceedings of 29th International Conference on Very Large Data Bases* (pp. 69-80). Berlin, Germany: Morgan Kaufmann.

Zhang, N., Kacholia, V., & Özsu, M. T. (2004). A Succinct Physical Storage Scheme for Efficient Evaluation of Path Queries in XML. *Proceedings of the 20th International Conference on Data Engineering, ICDE 2004* (pp. 54-65). Boston, MA, USA: IEEE Computer Society.