International Journal of Advanced Smart Convergence Vol.10 No.2 64-72 (2021)
http://dx.doi.org/10.7236/IJASC.2021.10.2.64

IJASC 21-2-10

Query Processing based Branch Node Stream for XML Message Broker

Hye-Kyeong Ko

Assistant Professor, Department of Computer Engineering, Sungkyul University, Korea
ellefgt@sungkyul.ac.kr

Abstract

XML message brokers have a lot of importance because XML has become a practical standard for data exchange in many applications. Message brokers covered in this document store many users. This paper is a study of the processing of twig pattern queries in XML documents using branching node streams in XML message broker structures. This work is about query processing in XML documents, especially for query processing with XML twig patterns in the XML message broker structure and proposed a method to reduce query processing time when parsing documents with XML twig patterns by processing information. In this paper, the twig pattern query processing method of documents using the branching node stream removes the twigging value of the branch node that does not include the labeling value of the branch node stream when it receives a twig query from the client. In this paper, the leaf node discovery time can be reduced by reducing the navigation time of nodes in XML documents that are matched to leaf nodes in twig queries for client twig queries. Overall, the overall processing time to respond to queries is reduced, allowing for rapid question-answer processing.

Keywords: *Branching node, XML Message Broker, Steam, Query processing*

1. Introduction

The eXtensible Markup Language (XML) is a markup language adopted to compensate for the disadvantages of hypertext markup language. XML defines the entire document separately in document structure, document content and output format, which further accelerates the spread of XML documents in that it provides a variety of characteristics for document structure, such as recycling and output flexibility [1].

In emerging infrastructures, XML message brokers will serve as a central exchange point for messages sent between applications and/or users. Three of the main functions of these brokers are filtering, transforming, and routing. Filtering matches messages with predicates that indicate specifications of interest. Currently, as the use and interest of XML for data representation on the web increases, interest in efficient query processing in XML documents is also increasing [2].

Manuscript Received: April. 18, 2021 / Revised: April. 27, 2021 / Accepted: May. 2, 2021
Corresponding Author: ellefgt@sungkyul.ac.kr
Tel: +82-31-467-8113, Fax: +82-31-467-8113
Assistant Professor, Department of Computer Engineering, Sungkyul University, Korea

Therefore, many query processing methods for efficient query processing in XML documents have been proposed [3-4]. Existing efficient query processing studies have been dominated by simple pass query processing with a single path, but the twig query processing with two or more paths that require the most processing time for XML document processing is recognized as an important problem [5-7]. To efficiently handle these twig queries, this paper utilizes the labeling techniques such as a region encoding labeling, a Dewey labeling [8, 9], and propose the twig pattern matching algorithms to efficiently handle queries such as TJFast [10], TwigStack [11], and Twig^{2}Stack [12].

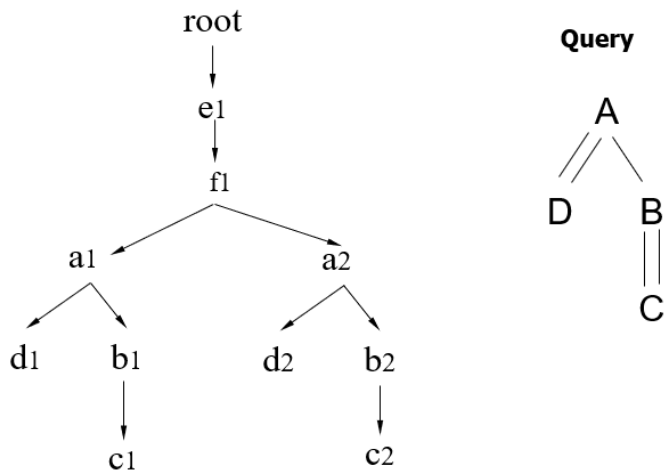


Figure 1. Example of XML document [10]

Figure 1 illustrates a typical example of an XML document. XML documents are constructed in hierarchical tree form, which enables search for document structures in Figure 1. Examples of XML query languages are XML Query Language (XQL) [13], the XML query language proposed as standard in W3C, and Query Language, which uses a separate specification, Xpath [14], indicating the elements of XML documents or the path of text in query representation. The figure to the right of Figure 1 illustrates the client's specific query to an XML document. A client's specific query specifies the relationship between nodes that construct documents using '/' or '//', a structural expression indicating the relationship between nodes.

The client's query shows that node A and node *D*, regardless of which node the middle node is, has offspring node *D* except for child node A, and requests queries for nodes with child node *B* and offspring node *C*. It is called a branching node because it has a branching node as shown on the right side of Figure 1, and node *C* is leaf node because they are superior to the end. Typical region-encoding based labeling techniques require that the labeling of child nodes exists between labeled parent nodes and their child nodes [8]. This can replace the sequence of document structure exploration with labeled values when exploring the structure of an XML document, making it easy to see the relationship between parent and child nodes. However, these region-encoding-based labeling techniques have the problem of having to explore all the nodes described in the twig query when a client requests a query.

This paper is an XML document about XML twig pattern query processing for XML documents stored in a database. By processing the information provided during parsing of this paper provides a method using a branching node stream that can efficiently reduce query processing time. The study addresses to query the twig pattern of XML documents using the branching node stream. The proposed scheme provides a twig pattern

query processing system for XML documents using a branching node stream using a processing method. The structure of this paper is as follows. Section 2 describes the existing XML query processing model as a related study, and Section 3 describes in detail the twig pattern query processing method. Section 4 describes the experimental results. Finally, the conclusions are described in Section 5.

2. Related Works

Data representation on the web has led to increased use of XML and increased interest in efficient query processing. Therefore, many query processing methods for efficient query processing of XML documents have been proposed [1, 3, 8]. TJ Fast techniques [10] can label document structures using the extended Dewey labeling and use twig pattern symmetry algorithms to respond to queries from clients. The extended Dewey labeling technique [8] allows you to know all the node information in the upper layer of the leaf node in an XML document by knowing the labeling information of the leaf node. The TJ Fast technique must explore all nodes of the XML document corresponding to the leaf node of the query in order to extract and send the resulting XML document with the client's query's node structure to the client [10]. The TJFast technique must explore all nodes of the XML document corresponding to the leaf node of the query to extract and send the resulting XML document with the client's query node structure to the client. This is because the TJFast technique is labeled by the extended Dewey labeling technique, so knowing the labeling value of the nodes in the XML document corresponding to the leaf node in the query, one can understand the parent structure of the XML document.

Since the TJFast labeling techniques know the labeling information of nodes in XML documents corresponding to leaf nodes in queries, they need to understand the mode of leaf nodes in XML documents and convert all element values comprising labeling values to node names [10]. Next, the TwgStack [11] uses the stack structure for matching twig patterns. There are several problems with this method as follows. The first is that if all nodes in an XML document are equal to query nodes, the size of the stack becomes equal to that of an XML document. Second, the TwigStack algorithm shows poor performance in the relationship between parent and child nodes [11]. For example, given a query with *A[/C]/B*, the TwigStack algorithm shows optimal processing in ancestor node-child node relationships such as *A/B* relationships, but in addition, query processing is very slow in parent node-child nodes such as *A/C* [11].

Another twig pattern matching algorithm based on region encoding labeling is the Twig^{2}Stack [12]. The technique is an efficient way to improve the speed problem of processing the relationship between the TwigStack's parent and child nodes, which slows down. Twig^{2}Stack navigation technique uses a modified stack structure and is called a hierarchical stack [12]. Using this hierarchical stack, the Twig^{2}Stack the relationship of the client's Twig query. By using this method, we can improve the processing of the relationship between the ancestor node and the offspring node, which was inefficient in the stack, more efficiently and faster processing However, the Twig^{2}Stack also suffers from the worst-case scenario where all nodes of the document must be stored in the stack. Furthermore, we have relationships between nodes and generate unnecessary processing of query processing if the relationships between nodes are complex [12]. As such, the TwigStack [11] and Twig^{2}Stack [12] still have problems with local encoding labeling they used the local encoding labeling method. Regionally encoded labeling has no information other than information that can determine the relationship between the parent and the child nodes.

3. Branch Node Stream for Twig Pattern Query Processing

The most fundamental research in the field of XML query processing is how to eliminate unnecessary intermediate computational values and how to minimize the response time of queries. The unnecessary intermediate operational values described here are not included in the results but are values that must be processed during processing. The response time of a query is the time when the client gives it to the client to process and send the desired final value. This work extracts the branching nodes and twig query leaf nodes of the above specific twig queries upon receipt of a specific twig query from a client in how to process queries in an extended Dewey labeled XML document.

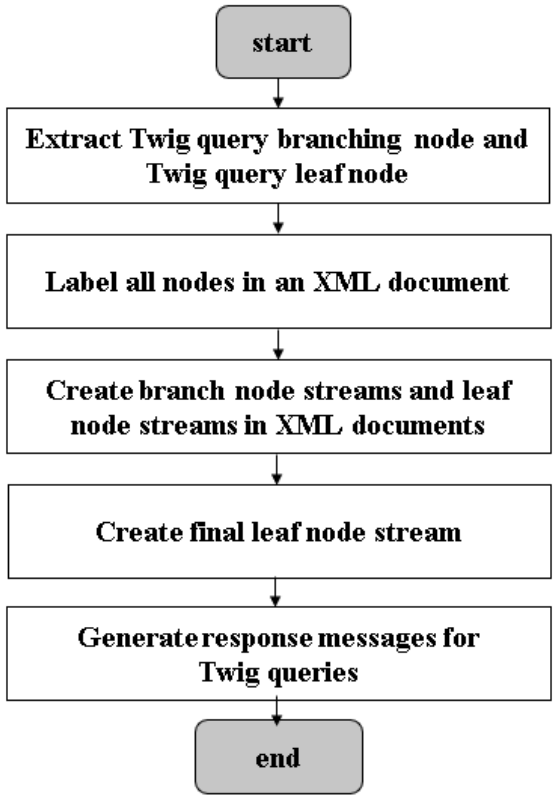


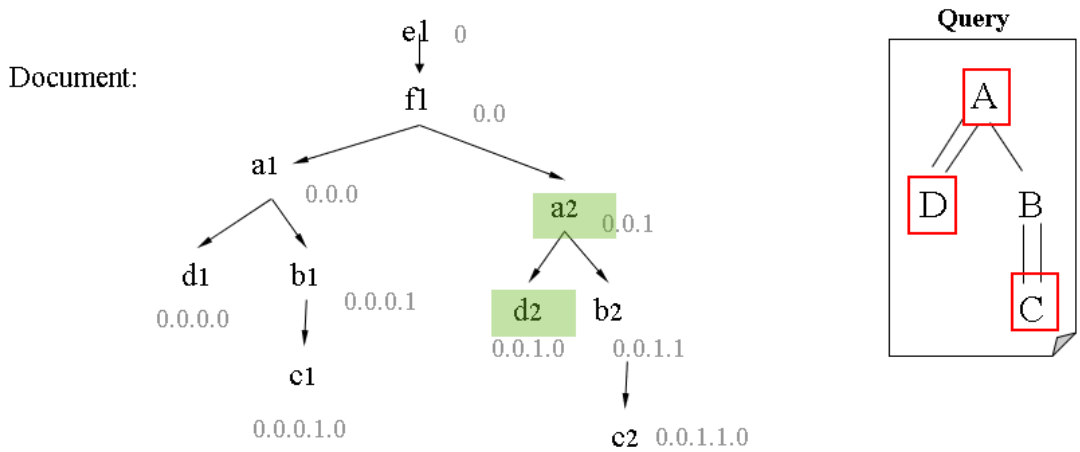
Figure 2. Flowchart of the query processing of twig query

Figure 2 is a flowchart of the query processing of Twig queries proposed in the paper. when a specific twig query is received for an XML document from a client in how to handle queries in an extended Dewey labeled XML document, we extract nodes consisting of twig query branching nodes and twig query leaf nodes in Figure 2. After parsing the XML document, the proposed scheme labels the XML document and generate a branching node stream of XML documents corresponding to the Twig query branching node and twig query leaf node of the twig query. The results in a final leaf node stream by creating a branching node stream consisting of the branching node of the same XML document as the branching node of the client's twig query and removing the labeled node of the XML document corresponding to the leaf node of the twig query. The results in a final leaf node stream by creating a branching node stream consisting of the branching node of the same XML document as the branching node of the client's twig query and removing the labeled node of the XML document corresponding to the leaf node of the twig query.

3.1 Twig query processor

The query processing device of the twig query in the XML document proposed in this paper can include the data handler, the twig query node extraction, the labeling generator, matching node stream generator, final leaf node stream generator, and the response message generator.

- **Data handler.** The data processor receives a specific the twig query from the client for the XML document and provides the client with a response message to the client's the twig query.
- **Twig query node extraction.** It receives a specific the twig query from the client for the XML document and provides the client with a response message to the client's the twig query. It serves to preferentially explore the twig query branching nodes and the twig query leaf nodes for twig queries in client. This allows us to distinguish between branching nodes and leaf nodes in the twig queries. It extracts query branching nodes and twig query leaf nodes for twig queries and sends extracted the twig query branching nodes and the twig query leaf nodes to matching node stream constructors, allowing labeling values of the branching nodes in XML documents corresponding to twig queries to be extracted from matching node stream generator.



$Branch\_Stream_A$	Result	$Leaf\_Stream_D$	Result	$Leaf\_Stream_c$	Result
0.0.0	a1	0.0.0.0	a1.d1	0.0.1.0	a2.d2
0.0.1	a2	0.0.0.1.0	a1.c1	0.0.1.1.0	a2.c2

Figure 3. Matching node stream

- **Labeling generator.** It parses XML documents and acts to ensure that all existing nodes have labeling values. In this paper, XML documents have a hierarchical document structure, so each node corresponding to an XML document is labeled according to the extended Dewey labeling technique.
- **Matching node stream generator.** It generates a branching node stream and leaf node stream of XML documents corresponding to the branching node and the twig query leaf node of the twig query. In this paper, values containing all the labeling values of the branching nodes of the XML document corresponding to the twig query are referred to as the branching node stream, and values containing all the labeling values of the leaf nodes of the XML document corresponding to the query are referred to as

leaf node streams. It creates a branching node stream of the same XML document as the branching node of the client's twig query, then corresponds to the above the twig query leaf node, and a leaf node of the XML document generates a leaf node stream with labeled values.

- **Final leaf node stream generator.** It generates the final leaf node by extracting the labeling values of the leaf nodes that represent the leaf node stream, which do not contain the labeling values of the branching nodes that make up the branching node stream. In other words, it is very important to explore leaf nodes in XML documents corresponding to leaf nodes in Twig queries because the extended Dewey labeling can know the labeling values of leaf nodes and understand the structural information of the parent nodes. However, all leaf nodes can be explored up to leaf nodes unrelated to the branching nodes in the query, so in this paper, we explore leaf nodes in the XML document corresponding to the leaf nodes in the query, as well as branching nodes in the query.
- **Response message generator.** It generates a response message to the twig query using a branching node stream and a final leaf node stream. In other words, we generate a response message to the twig query using the labeling values of the nodes included in the branch node stream and the labeling values included in the final leaf node stream.

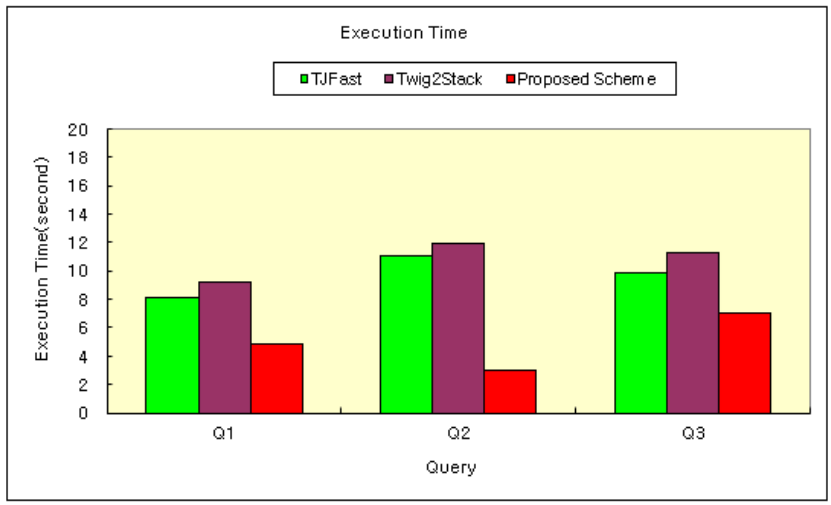
Figure 3 illustrates the matching node stream generated by the matching node stream constructor. Figure 3 shows generating XML node streams and leaf node streams that match client's twig queries for client queries. For client the twig queries, the nodes in the XML document corresponding to branching node *A* are *a1* and *a2*, respectively labeled (0.0.0) and (0.0.1). For the twig query, the XML documents corresponding to leaf nodes *D* is *d1* and *d2* with labeling values (0.0.0.0) and (0.0.1.0).

On the other hand, the nodes in the XML document corresponding to leaf node *C* for the twig query are *c1* and *c2*, and their labeling values are (0.0.0.1.0) and (0.0.1.1.0). Based on this, the branching node stream $Branch\_Stream_A$ of the XML document generated by the matching node stream generator for the twig query branching node *A* is (0.0.0), (0.0.1) and the leaf node stream $Leaf\_Stream_D$ of the XML document generating by the matching node stream generator for the twig query leaf node *D* is (0.0.0.0), (0.0.1.0). In addition, the leaf node stream $Leaf\_Stream_C$ of XML documents generated by the matching node stream generator for the twig query list node *C* is (0.0.0.1.0) and (0.0.1.1.0).

In Figure 3, we see a node name table computed by the final leaf node stream $Leaf\_Stream_D$ generated for $Branch\_Stream_A$ and the twig query leaf node *D*, and the final ret node stream $Leaf\_Stream_C$ generated for the twig query leaf node *C*, based on the node stream. In this paper, the labeling values of multiple leaf nodes that make up the final leaf node stream, we create an optimized leaf node stream by removing overlapping nested labeling elements and convert the optimized leaf node stream containing residual labeling elements to node names on XML documents. In Figure 3, we show that the nested labeling element, where both leaf node *C* and leaf node *D* is nested, has two zeros in the front, thus eliminating it to produce an optimized leaf node stream.

4. Experimental Results

We implemented three XML twig join algorithms:TJFast [10], Twig^{2}Stack [11], proposed scheme in JDK 1.7 [15] using the file system as a storage engine. All experiments were run on a Intel® Core™ i5 CPU with 8MB of main memory and running windows 10. We used the DBLP [16] dataset in our experiments and used structures that can represent parent-child and ancestor-sibling relationships. Figure 4 is a graph comparing the performance of existing twig query processing techniques with the twig query processing methods proposed in this paper.



Q1 : //dblp/inproceedings[title]/author

Q2 : //dblp/article[author][.//title]/year

Q3 : //inproceedings[./title]//booktitle

Figure 4. Execution Time (DBLP)

Figure 4 shows a graph comparing the performance of TJFast, Twig^{2}Stack techniques and the techniques proposed in this paper among existing query processing methods. In Figure 4, *Q1* compares the overall response processing time for the twig query if it is *//dblp/inprocessings[title]/author*, and *Q2* compares the overall response processing time for the twig query if it is *//dblp/article[author][.//title]/year*. The *Q3* compares the overall response processing time when the twig query is *//inprocessedings[./title]//booktitle*.

As a result, the proposed scheme shows that the proposed method shows a performance improvement of *23*% to *50*% over TJFast, Twig^{2}Stack among the query processing methods of conventional twig queries for both twig queries *Q1*, *Q2*, and *Q3*.

- **Theorem 1.** The I/O optimality of the proposed technique implies that there is an alternative to the upper-lower or upper-lower relationship between the branch node and the sub-node of the branch node. The I/O complexity of the proposed technique in the worst case scenario is similar to the worst case of the I/O complexity of TJFast.

- **Theorem 2.** The worst case of spatial complexity is $O(d*b*m)$. The d, b, and m are used to represent the length of the longest label in the input list, the number of leaf nodes in the input list of leaf nodes in query Q, and the number of branch nodes in the input list of branch nodes in query Q.

As shown in the experimental results, the proposed method in this paper can reduce the I/O time, a leaf node navigation time, by reducing the navigation time of nodes on XML documents that are matched to leaf nodes in a twig query for clients' twig queries. In addition, when converting labeled values on leaf nodes to original node names for nodes explored for twig queries, the labeled values that correspond to twig queries are not converted to original node names, reducing the intermediate processing time. Overall, the overall processing time is reduced allowing for rapid question answering processing.

5. Conclusion

In terms of how to process queries for the twig query in the extended Dewey labeled XML documents, the proposed scheme receives specific the twig query for XML documents from clients. It performed steps to extract the twig query branching nodes and twig query leaf nodes from specific the twig query to generate response messages to provide a way to handle queries. The proposed scheme the branching node stream with the leaf node stream, which causes the code to generate the final leaf node stream. Furthermore, it determines whether the labeling value of each leaf node that makes up the leaf node stream contains the labeling value of either of the branching node streams. If the labeling value of a leaf node does not contain the labeling value of any one branching node, it should contain code that removes the labeling value of the leaf node from the leaf node stream.

The experimental results show that, for client's the twig queries, we can reduce the I/O time, which is the leaf node navigation time, by reducing the navigation time of nodes on XML documents that are matched to leaf nodes in the twig query. Furthermore, the discovered nodes for the twig query convert the labeled values of the leaf nodes to the original node names. The proposed scheme does not convert the labeled values nested with the branched nodes corresponding to the twig query to the original node name, which reduces the intermediate processing time and reduces the overall processing time to respond to the query.

Acknowledgement

This work was supported by the National Research Foundation of Korea (NRF) grant funded by the Korea government (MSIT).(No.NRF-2021R1A2C1012827)

References

[1] Extensible Markup Language (XML), https:/.www.w3.org.

[2] I. Tatarinov, S. Viglas, K. Beyer, S. Jayavel, E. Shekita, and C. Zhang, “Storing and Querying Ordered XML Using a Relational Database System,” in Proc. ACM SIGOMD International Conference on Management of Data, pp. 204-215, 2002. DOI: https://doi.org/10.1145/564691.564715.\

[3] S. Im and H. Hwang, “Design and Development of Framework for Wireless Data Broadcast of XML-based CCR Document,” The Journal of the Institute of Internet, Broadcasting and Communication, vol. 15, no. 5, pp. 169-175, 2015.
DOI: https://doi.org/10.7236/JIIBC.2015.15.5.169

[4] H. Lee and H-W. Lee, “Forgery Detection Mechanism with Abnormal Structure Analysis on Office Open XML based MS-Word File,” The International Journal Advanced Smart Convergence, vol. 8, no. 4, pp. 47-57, 2019.
DOI: https://doi.org/10.7236/IJASC.2019.8.4.47

[5] M. Kim, J. Lee, H. Son and H. Kim, “HCov-IMDB: Database for the Analysis of Interactions between HCoV and Host Immune Proteins,” The International Journal Advanced Smart Convergence, vol. 8, no. 1, pp. 1-8, 2019.
DOI: https://doi.org/10.7236/IJASC.2019.8.1.1

[6] M. Min, “Experiments of Search Query Performance for SQL-Based Open Source Databases,” International Journal of Internet, Broadcasting and Communication, vol. 10, no. 2, 2018.
DOI: https://doi.org/10.1109/ICDE.2002.994704.

[7] H. Kwon, “A Query-based Data Allocation Scheme for Multiple Broadcast-Channel Environments,” The Journal of the Institute of Internet, Broadcasting and Communication, vol. 16, no. 6, 2016.
DOI: https://doi.org/10.7236/JIIBC.2016.16.6.165.

[8] J. Lu, T. W. Ling, C. Chan, and T. Chen, “From Region Encoding to Extended Dewey: On Efficient Processing of XML Twig Pattern Matching,” in Proc. 31st International Conference on Very Large Data Bases, pp. 193-204, 2005.
DOI: https://doi.org/10.5555/1083592.1083618.

[9] S. Al-Khalifa, H. V. Jagadish, N. Koudas, J. M. Patel. Y. Wu, N. Koudas, D. Srivastava, "Structural Joins: A Primitive for Efficient XML Query Pattern Matching," in Proc. 18th International Conference on Data Engineering, pp. 141-152, 2002.
DOI: https://doi.org/10.1109/ICDE.2002.994704.

[10] J. Lu, T. Chen, and T. W. Ling, "TJFast: Effective Processing of XML Twig Pattern Matching," in Proc. 14th International Conference on World Wide Web, pp. 11P18-1119, May 10-14, 2005.
DOI: https://doi.org/10.1145/1062745.1062897.

[11] N. Bruno, D. Srivastava, and N. Koudas, "Holistic Twig Joins: Optimal XML Pattern Matching," in Proc. International Conference on Management of Data and Symposium on Principles Database and Systems, pp. 310-321, June, 2002.
DOI: https://doi.org/10.1145/564691.564727

[12] J. Lu, T. Chen, and T. W. Ling, "Efficient Processing of XML Twig Patterns with Parent Child Edges: A Look-Ahead Approach," in Proc. 13th ACM International Conference on Information and Knowledge Management, pp. 533-542, 2004.
DOI: https://doi.org/10.1145/1031171.1031272.

[13] S.Boag, D. Chamberlin, M. F. Fernandez, D Florescu J. Robie, J. Simeon, "XQuery 1.0: An XML Query," W3C Working Draft 22, August, 2003.

[14] A. Berglund, S. Boag, D. Chamberlin, M. F. Fernandez, M. Kay, J. Robie, and J. Simeon, "XML Path Language (XPath) 2.0," W3C Working Draft 22, August, 2003.

[15] JDK 1.7, https://www.oracle.com/kr/java/technologies/javase/javase7-archive-downloads.html.

[16] DBLP: Computer Science Bibliography, https://dblp.org.