스트림된 XML 조각들의 효율적인 질의 처리

고 혜 경*

Effective Query Processing on Streamed XML Fragments

Hye-Kyeong Ko*

■ Abstract ■

Query processing on streamed XML fragments is one of key issues in XML databases. In this paper, XFSeed (XML Fragment Processor with Seed label) is proposed to provide effective query processing by removing many redundant path evaluations and minimizing the number of fragments processed. The conducted experimental results reveal that the proposed scheme efficiently handles query processing and reduces memory usage.

Keyword : Query Processing, Streamed Data, XML Fragment

1. Introduction

XML [12] is emerging as a de facto standard for information representation and data exchange on the web. As semi-structural data, XML can be represented as a tree-structured model with data content and structural relationships. To evaluate XML queries, XPath [13] and XQuery [14] were widely studied in database management systems. XML data, which is inherently hierarchical and semi-structured, imposes substantial overhead on run-time aspects, such as memory requirements and processing efficiency [2]. This paper focuses on effective query processing on continuous XML fragment streams. A wireless information dissemination system consists of battery powered portable devices such as PDA's and mobile phones with limited data processing abilities [2]. Most queries on a large XML document are selective, this, queries may benefit from XML fragmentation, for processing in components with less memory and processing power [2, 4]. Many applications, such as network intrusion detection, sensor network monitoring business transactions and earth climate monitoring, require analysis of streaming data [7]. For example, a real-time sensor-based system is continuously generating data from sensors and data is disseminated in fragments during this process [2, 4]. There are several reasons for fragmenting data. With increasing usage of mobile devices and request for information while mobile, a server disseminates data in low-bandwidth and error-prone environments [2]. In addition, synchronization for small fragments is easy, because transmitting an update to data only requires sending fragments relevant to the update, instead of sending the whole document.

Unfortunately, processing fragments instead of the whole XML document is fraught with challenges. Firstly, fragment may arrive in any order, which is challenging as regards memory requirements. Secondly, query processing requires location information about fragments, in order to reconstruct an XML document relative to fragment [2, 4, 7].

Recently, Hole-Filter model was proposed in order to correlate XML fragment [4]. Much research focuses on answering queries about streamed XML data, which handles fragmented XML data based on Hole-Filler model, such as XFrag [2], XFPro [7] and XFPR [6]. [Figure 1] represents an example of Hole-Filler model. Hole-Filler model, however, has two main limitations : (1) The memory overhead for the hole/filler IDs with the related XML tag in the XML fragment could be very large. (2) Given two XML fragment, it cannot directly identify the Ancestor-Descendant (A-D) relationships between fragments. Due to these limitations, the required memory for query processing on streamed XML fragment is increased. The principal problem with Hole-Filler is the limited effectiveness of reconstructing components of the original XML document with limited memory, and evaluating

[Figure 1] Example of Hole-Filler Model

queries relative to an XML fragment. In order to solve this problem, the location information of fragments should be known. To achieve this, the labels of fragments can be utilized such that the relationship of fragments can be obtained via these labels.

In order to facilitate the determination of relationships among nodes, nodes in the XML tree are typically labeled such that structural relationships between any two nodes can be quickly established.

In this paper, we present a new framework, called XFSeed (XML Fragment Processor with Seed label) based on the seed labeling scheme [23], which provides effective XPath query processing on streamed XML fragments in terms of processing time and memory usage.

Our current work extends [23] in two ways. First, XFSeed (XML fragment processor with seed label) is proposed to process XPath expressions on the client side. Second, we thoroughly evaluated the effectiveness of the proposed scheme.

The paper is organized as follows. We first discuss preliminaries in Section 2. Section 3 describes the framework for processing XML fragments. Section 4 show some experimental results. Section 5 presents the related work, and Section 6 provides our conclusion.

2. Preliminaries

2.1 XML Fragmentation Model

Given an XML tree, and XML fragment is a subtree of root. An XML document is pruned into many fragments. In order to summarize the structure of XML fragments, a tag structure is

used to support structural and fragment information. A tag structure is itself a valid XML fragment in accordance with an XML schema or DTD, where a tag corresponds to an XML element and is qualified by a unique id, a name (the element tag name), and a type [8]. In our approach, we adopt Hole-Filler model [8] to describe XML fragments, which hold both the data contents and structural relationships. We assume that a single document D is a node labeled acrylic tree with the set V of infinite nodes and the set E of finite edges. XML stream begins with finite XML documents and runs on as and when new elements are added into the document or updates occur upon the existing elements.

```
<stream : structure>
  <tag name = "book" id = "1" Filler = "true">
    <tag name = "title" id = "2"/>
    <tag name = "author" id = "3" Filler = "true">
      <tag name = "author" id = "4"/>
    </tag>
    <tag name = "year" id = "5"/>
    <tag name = "section" id = "6" Filler = "true">
      <tag name = "head" id = "7"/>
      <tag name = "subsection" id = "8"
        Filler = "true">
        <tag name = "head" id = "9"/>
        <tag name = "subsection" id = "10"/>
      </tag>
      <tag name = "figure" id = "11" Filler = "true">
        <tag name = "title" id = "12"/>
      </tag>
    </tag>
  </tag>
</stream : structure>
```

[Figure 2] Tag Structure of Hole-Filler Model

Some fundamental notions used in this paper present the following :

An XML document D is a tree $T_d = (V_d, E_d, \Sigma_d, root_d, Did)$, where V_d is an infinite set of nodes, including element nodes. attribute nodes

and text nodes; E_d is a finite set of directed edges, indicating parent-child relationship between element nodes or containment relationship between element nodes and attribute nodes; each has a type and is identified by Did , Σ_d is the set of node types; $root_d (\in V_d)$ is the root element of D .

A filler F is a subtree of XML document. $T_f = (V_f, E_f, \Sigma_f, root_f, fid, head_f, H_d)$, where V_f is the subset of V_d , E_f is the subset of E_d , and Σ_f is the subset of Σ_d ; each filler is identified by fid , which is included in $head_f$, H_d is a finite set of holes; $root_f (\in V_f)$ is the root element of the subtree.

A hole H is an empty node n ($n \in H_d$) assigned with unique hid , into which a filler with the same fid value could be positioned to complete the tree. Tag structure is a fragment of XML document with the highest priority $TS = (V_t, E_t, root_t, ID_t, Did)$, where V_t is an infinite set of tag nodes in XML document; E_t is a finite set of edges; ID_t is a set of number identifying the tag nodes in XML document; Did is the XML document identifier.

Given an XML document tree, we can fragment it by recursively inserting a hole at every point where a subtree is pruned, i. e. a filler is generated, and associated with an ID (the fid of the filler fragment). Note that the filler can in turn have holes in it, which will be filled by other fillers. We can reconstruct the original XML document by substituting holes with the corresponding fillers at the destination as it was in the source. Tag structure is a structure summary information for an XML document. It can be generated according to XML Schema or DTD, and also can be obtained when fragmenting an XML document without DTD.

2.2 XPath

An XPath query is in the form of $N_1N_2 \dots N_n/O$, which consists of a location path, $N_1N_2 \dots N_n$, and an output expression O [21]. An element matches the location path if the path from the document root to that element matches the sequence of labels in the location path, and satisfies all predicates (specified syntactically using square brackets). For each matching element, the results of applying the output function to the element, the results of applying the output function to the element are added to the query result. The output expression can specify an attribute of the element.

<p>Fragment 1:</p> <pre><stream: filler id="null" tsid="1"> <book name="corel"> <title> XML </title> <stream: hole id="1.2.1.2" tsid="3"/> <year> 2007 </year> <stream: hole id="1.4.1.4" tsid="6"/> </book> </stream: filler></pre>	<p>Fragment 2:</p> <pre><stream: filler id="1.2.1.2" tsid="3"> <authors> <author> J. Lee </author> </authors> </stream: filler></pre>
<p>Fragment 3:</p> <pre><stream: filler id="1.4.1.4" tsid="6"> <section> <head> XML DTD </head> <stream: hole id="2.4.4.7" tsid="8"/> </section> </stream: filler></pre>	<p>Fragment 4:</p> <pre><stream: filler id="2.4.4.7" tsid="8"> <subsection> <head> ... </head> <title> ... </title> </subsection> </stream: filler></pre>

[Figure 3] Example of XML Fragments

3. XFSeed Query Processor

3.1 Fragment Representation

In Hole-Filler model [4], fillers are associated with holes by matching $fids$ with $hids$. Given two XML fragments, it cannot directly identify Ancestor-Descendant (A-D) relationships between fragments. Due to these limitations, the memory requirements for query processing on streamed XML fragments are increased. Thus,

Hole-Filler is the limited effectiveness in reconstructing components of the original XML document relative to XML fragment. In order to solve this problem, the location information of an element is represented as seed labeling (C1.C2.C3.C4) [23]. For example, subsection and head are in the same filler, and the level of this filler equals the level of subsection in the original document in [Figure 3]. C1 is the nesting depth of the fragment's root element in the original XML document, which aids in the identification of the parent-child relationship between fragments. C2 represents the seed number, which aids in the easy identification of Ancestor-Descendant relationship, because it minimizes the number of fragments processed. Thus, seed labeling provides effective processing of the descendant axis (//) and axis (/) of an XPath expression.

In this paper, both fid and hid are represented using seed labeling. When an XML document modeled as a tree is fragmented, it can also be represented as a tree. [Figure 3] represents four fragments of an XML document in [Figure 2], after coding fid and hid with seed labeling.

Here, a root's filler is null. Other filler IDs can be generated on the server side. The fillers are connected with holes by matching filler IDs with hole's IDs. Fragment 4's fid corresponds to Fragment 3's hid. Fragment 4 is a subtree of Fragment 3, when an XML document is reconstructed. Given the label, the parent-child relationship between nodes is obtained by matching labels. For example, suppose that f1 with label (f1C1f1C2.f1C3.f1C4) and f2 with label (f2C1.f2C2.f2C3.f2C4) are fragments. f2 is a descendant of f1 iff f1C1 > f2C1 and f1C2 = f2C2. f2 is a child of f1 iff f1C1 = f2C1-1 and f1C4 = f2C3. Thus, Fragment

4 with label (2.4.4.7) is a child of Fragment 1 with label (1.4.1.4) in [Figure 3].

3.2 XFSeed Query Handling

According to Hole-Filler model, infinite XML streams can be represented as a sequence of XML fragments, which are the basic processing units of the query. However, input queries evaluate elements in an XML document, not XML fragments. Our goal is effective query processing by removing many redundant path evaluations and minimizing the number of fragments processed.

[Figure 4] Example of XML Fragment with Seed Labeling [23]

3.2.1 Acceleration of Determination of Fragment

Fragments in the original document may arrive in any order and query expressions contain predicates at any level in the XML tree. Thus, the location information of fragments should be known. We maintain fragment relationship in the association hash table at each operator, to record parent-child relationships in fragments processed by the operator. Three values are possible in an

intermediate operator that does not produce a result. In addition, each entry is assigned a value of true, false, undecided ($\perp$), or a result fragment. We need to locate the corresponding fragments, which are represented by element nodes with filler in query expressions.

[Figure 4] shows an example of an XML fragment for Hole-Filler model with seed labeling. For example, Q1 : */book/section/subsection/head* that head is not filler and section element is filler. We need to process the particular fragment section in an output operator, b matching its C2 component in the label without inquiring parent fragments. Q2 : */book/section//head* is a simple query with “//”. we can identify structural relationship with a section filler (1.4.1.4) and a filler matching head element (3.4.2.1), because seed labeling can quickly identify an ancestor-descendant relationship without checking intermediate fillers. Using the two components (C1 and C2) of seed labeling, we can minimize comparison of fragments to identify the ancestor-descendant relationship. If $a.C1 > d.C1$ and $a.C2 = d.C2$, fragment a is the ancestor of fragment d. In this case, intermediate nodes in a query tree are removed.

3.2.2 XFSeed Processing

When a fragment is processed by an operator which can identify the predecessor, the operator has excluded its parent fragment. Therefore, each operator preserves both a successor operator list and a pointer to the predecessor operator. Algorithm 1 describes fragment relationships for reconstruction of an XML document relative to fragments. When a fragment is processed by XFSeed, it initially identifies seed labeling of the fragment. If the ancestor fragment has arrived,

the value of label is copied from the status of its ancestor's value, otherwise the value is assigned as “ $\perp$ ”. Then, it must trigger descendant fragments and pass the status value to its descendants, as fragment may be waiting for operators to decide their ancestor relationship.

```

if(isFragStart( ) == true) then
  tsid = getTsid( );
  fid = getFid( );
  if(isQueryFrag( ) == true) then
    a = findAncestorOperator (tsid);
    for each seed_label sl of a do
      if (sl.C1x > sl.C1y && sl.C2x == sl.C2y) &&
        sl.fid.value! =  $\perp$ )
        then CurrentValue = sl.value;
      end if
    end for
  else
    p = findParentOperator (tsid);
    for each seed_label sl of d do
      if (sl.C1x == sl.C1y+1 && sl.C4x == sl.C3y) &&
        sl.fid.value! =  $\perp$ )
      then
        end if
      end for
    end if
  endif
endif

```

[Algorithm 1] Searching Fragment Relationships

Consider Q3 : *//section/subsection[figure]//title* as a branching query. In order to distinguish figure in the main path expression, existing approaches cannot reduce the rate of query evaluation, because blocking will occur when waiting for the arrival all correlated fragments to complete a branching query. However, XFSeed can remove intermediate fragments according to components (C2) of the label and obtain results as soon as possible using the proposed labeling scheme. Based on the seed number (C2), XFSeed provides efficient query processing by removing many redundant path evaluations and minimizing the number of fragments processed. For example, the *subsection* fragment with label (2.4.4.2) has previous arrived, and its information is re-

corded in the association table. When the *figure* fragment with label (3.4.3.1) arrives, it is compared with the *subsection* fragment by their labels. Since $subsection.C2 = 4$, $figure.C2 = 4$ and $subsection.C4 = 2$ and $figure.C3 = 3$. Therefore, *subsection* is not a parent fragment of *figure*, thus, the *figure* fragment is removed.

4. Performance Evaluation

4.1 Environment

In our experiment, it is assumed that an XML document has previously been fragmented. We focus on the query processing time and memory usage for streamed XML fragments on the client side. We implemented XFrags [2], XFPro [7], XFPR [6] and XFSeed in Java on Windows XP Professional. We present the results of performance evaluation on different queries and document sizes. All experiments are conducted on a PC with 2.6GHz CPU and 2GB memory. The experiments are conducted on database generated by the xmlgen program [11].

<Table 1> Query Type

	XPath Expression
Q1	doc("book.xml")/book/section/subsection/title
Q2	doc("book.xml")/book/section[difficulty >= "default"]/title
Q3	doc("book.xml")/book//section/subsection/title
Q4	doc("book.xml")/book/section[figure/title]/section/title

4.2 Experimental Results

4.2.1 Query Performance

XFPR adopted region-based coding, which reduced intermediate fillers. However, it cannot easily identify the parent-child ("/") relationship

and predicates. When processing predicates such as Q2, some intermediate fillers for processing query occur, thus, memory usage is increased.

[Figure 5] Query Processing Time(Q1~Q4)

[Figure 6] Memory Usage(Q1~Q4)

[Figure 5] showed that XFrag is costly, as the document size increased. In XFPro, the query processing pipeline is improved for XFrag, because it merely considers subroot nodes. How-

ever, many intermediate fillers occur for queries, including “*” or “//.”

XFSeed shows the best performance for four queries, because it removed redundant operations. Moreover, the proposed labeling scheme minimized the number of fragments processed using a seed component (C2). The query processing time required by XFSeed is less than that of XFPR. Q2 needed to track filler-hole links from *section* fillers to the corresponding *title* fragment. Therefore, the memory usage is increased. Comparing XFSeed with XFPR, in terms of processing time, the improvement ratio is 30%(Q1), 49%(Q2), 23%(Q3), 40%(Q4) on XFPR for a 20MB XML document.

4.2.2 Memory Usage

The memory usage was measured using the Eclipse Profiler plugin [16] for Eclipse IDE, and results are shown in [Figure 6], using generated *book* XML document. The *book* XML document was fragmented into fillers and holes, and the resulting fuller fragments were processed sequentially. *Section*, *subsection* and *title* are filler fragments, according to the fragment information in the tag structure, Q1 processed *subsection* and *title* on a fragment using a seed label. In XFrag, each fragment needs to be passed via the pipeline and evaluated step-by-step. During query processing, the time was measure on the client side. In the experiments, XFSeed outperforms other approaches.

It is observed that XFPro generally outperforms XFrag for query processing time. XFPro reduces CPU time, which avoids subsumption operations for query processing time. However, it did not outperform XFSeed, because, XFPro cannot remove intermediate fillers. In [Figure 6],

XFSeed outperformed the other method for memory usage for four queries. Comparing XFSeed with XFPR, in terms of memory usage, the improvement ratio is 30% (Q1), 15% (Q2), 23% (Q3), 10% (Q4) for a 20MB XML document.

5. Related Work

Several recent efforts have focused on addressing frameworks for continuous processing of data streams [3, 20, 21]. However, there is no work node in stream query processing of fragmented XML data. The Tribeca [22] data stream processing system provides language constructs to perform aggregation operations, but it is restricted to relational data. Several efforts were made to address stream processing of XML data using XPath expressions [1, 5, 10]. The Hole-Filler model was first proposed in [8]. However, it is used in the context of pull-based content navigation of mediated views of XML data from disparate data sources. In Xstream [12], the advantages of semantics-based fragmentation of XML data for efficient transmission in a wireless medium are highlighted. Recently, much research work addresses query processing of streamed XML fragments based on the Hole-Filler model, such as XFrag [2], XFPro [7] and XFPR [6]. In XFrag [2], XML fragments are processed when they arrive, and only fragments that may affect query results are maintained in the association table. However, XFrag pipeline is still memory intensive in maintaining the links in association tables, and time intensive in scheduling operations for each fragment [6, 7]. Moreover, redundant operations are caused by dependence between adjacent operations in XFrag. XFPro [7] presents a framework and a set of

techniques for processing XPath queries on streamed XML fragments, and techniques for enabling conversion from an XPath expression to an optimized query plan. In XFPro, the query processing pipeline is improved for XFrag because it merely considers subroot nodes. However, many intermediate fillers occur for queries, including “*” or “//”. Region-based coding was adopted for the streamed XML fragment model in XFPR [6]. Utilizing region-based coding, XFPR quickly identifies the ancestor-descendant relationship. However, XFPR can not efficiently identify the parent-child relationship and predicates, thus memory usage is increased.

6. Conclusion

This paper presented the framework for processing a streamed XML fragment using Hole-Filler model. Firstly, a robust labeling scheme is proposed, which can determine structural relationships between fragments very quickly. Secondly, based on the proposed labeling scheme, XFSeed is proposed, which supports efficient query processing by removing many redundant operations and minimizing the number of fragments processed. XFSeed can efficiently process structural relationships, such as “./”, “/”, and branching nodes. The experimental results showed that the proposed framework can reduce the query processing cost and memory usage cost for queries on XML fragments.

References

- [1] Barton, C., P. Charles, D. Goyal, M. Raghavachari, M. fontoura, and V. Josifovski, *Streaming XPath Processing with Forward*

- and Backward Axes, In Proceedings of the 19th International Conference on Data Engineering, (2003), pp.455-466.
- [2] Bose, S. and L. Fegaras, *XFrag : A Query Processing Framework for Fragmented XML Data*, In Proceedings of International Workshop on the Web Databases, (2005), pp.97-102.
- [3] Carney, D. et al., *Monitoring Streams-A New Class of Data Management Applications*, In Proceedings of the 28th International Conference on Very Large Data Bases, (2002), pp.215-226.
- [4] Fegaras, L., D. Levine, S., and Bose, V. Chaluvadi, *Query Processing of Streamed XML Data*, In Proceedings of the 11th International Conference on Information and Knowledge Management, (2002), pp.126-133.
- [5] Gupta, A. and D. Suciu, *Stream Processing of XPath Queries with Predicates*, In Proceedings of the 2003 ACM SIGMOD International Conference on Management of Data, (2003), pp.419-430.
- [6] Hui, X., G. Wang, H. Huo, C. Xiao, and R. Zhou, *Region-Based Coding for Queries over Streamed XML Fragments*, In Proceedings of the 7th International Conference in Web Information Systems Engineering, (2006), pp.487-498.
- [7] Huo, H., G. Wang, X. Hui, R. Zhou, B. Ning, and C. Xiao, *Efficient Query Processing for Streamed XML Fragments*, In Proceedings of the 11th International Conference on Database Systems for Advanced Applications, (2006), pp.468-482.
- [8] Ludascher, B., Y. Papakonstantinou, and P. Velikhov, *Navigation-Driven Evaluation of Virtual Mediated Views*, In Proceedings of the 7th International Conference of Extending Data Base Technology, (2000), pp.150-165.
- [9] Olteanu, D., T. Furche, and F. Bry, *An Efficient Single-Pass Query Evaluator for XML Data Streams*, In Proceedings of the 2004 ACM Symposium on Applied Computing, (2004), pp.627-631.
- [10] Peng, F. and S. Chawathe, *XPath Queries on Streaming Data*, In Proceedings of the 2003 ACM SIGMOD International Conference on Management of Data, (2003), pp.431-442.
- [11] Altova XML Spy, http://www.altova.com/products/xmlspy/xml_editor.html.
- [12] Schmidt, A., F. Vaas, M. L. Kersten, M. J. Carey, I. Manolescu and R. Busse, *XMark : A Benchmark for XML Data Management*, In Proceedings of the 28th International Conference on Very Large Data Bases, (2002), pp.215-226.
- [13] Wang, E. et al., *Efficient Management of XML Contents over Wireless Environment by Xstream*, In Proceedings of the 2004 ACM Symposium on Applied Computing, (2004), pp.1122-1127.
- [14] W3C Recommendation : Extensible Markup Language (XML) 1.0 (Second Edition). <http://www.w3.org/TR/REC-xml>.
- [15] W3C Working Draft : XML Path Language (XPath), version 2.0(2001) Technical Report WD-xpath20-20011220, W3C, 2001, <http://www.w3.org/TR/WD-xpath2020011220>.
- [16] W3C Working Draft : XQuery 1.0 : An XML Query Language. (2001) Technical Report WD-xquery-20010607, World Wide Web Consortium.
- [17] <http://eclipsecolorer.sourceforge.net>.

- [18] Tatarinov, S., D., Viglas, K. Beyer, J. Shanmugasundaram, E. Shekita, and C. Zhang, *Storing and Querying Ordered XML Using a Relational Database System*, In Proceedings of the 2002 ACM SIGMOD International Conference on Management of Data, (2002), pp.204-215.
- [19] Zhang, C., J. Naughton, D. Dewitt, Q. Luo, and G. Lohman, "On Supporting Containment Queries in Relational Database Management Systems", *ACM SIGMOD Record*, Vol.30, No.2(2001), pp.425-436.
- [20] Gupta, A. and D. Suciu, *Stream Processing of XPath Queries wit Predicates*, In Proceedings of the 2003 ACM SIGMOD International Conference on Management of Data, (2003), pp.419-430.
- [21] Peng, F. and S. Chawathe, *XPath Queries on Streaming Data*, In Proceedings of the 2003 ACM SIGMOD International Conference on Management of Data, (2003), pp. 431-442.
- [22] Sullivan, M. and A. Heybey, *Tribeca : A system for managing large databases of network traffic*, In the USENIX Annual Technical Conference, (1998), pp.13-24.
- [23] Ko, H.-K., "Efficient Labeling Scheme for Query Processing over XML Fragment Stream in Wireless Computing", *The KIPS (Korea Information Processing Society) Transactions : Part D*, Vol.17, No.5(2010), pp.353-358.

◆ 저 자 소 개 ◆

고 혜 경 (hkko@sungkyul.ac.kr)

고려대학교 정보통신대학원 컴퓨터학과에서 전산학 박사 학위를 취득하였으며, 현재 성결대학교 교양교직학부 조교수에 재직 중이다. 주요 관심분야는 빅데이터 분석, XML 데이터베이스, XML 보안, 시맨틱 웹 서비스 등이다.