Available online at www.ijpe-online.com

International Journal of Performability Engineering

Vol. 13, No. 5, September 2017, pp. 620-632
DOI: 10.23940/ijpe.17.05.p7.620632

An XML Streaming Data Processing Method based on Forest Transducer

Zhixue He[a,b,*], Husheng Liao[a]

[a]College of Computer Science, Beijing University of Technology, Beijing, 100124, China
[b]College of Computer, North China Institute of Aerospace Engineering, Langfang, 065000, China

Abstract

XML is the de facto standard for data representation and exchanging on web. The query processing technique of XML streaming data is a hotspot in current research. Focused on the characteristics of processing semi-structure XML streaming data such as the stream arriving continuously, requiring to be read sequentially and only once into memory, the querying must be processed on the fly, a method of processing XPath query based on forest transducer is proposed. Firstly, conversion rules of forest transducer are defined for XPath query. And then the transducer is driven by input streaming data nodes. Stack and abstract syntax tree are applied to implement match and state transformation in running procedure. The relationships between state functions and intermediate results are kept by the abstract syntax tree, and the query results are output in reducing process. Finally, the experimental results show that our approach is effective and efficient on this problem, and outperforms about 30 percent of the state-of-the-art algorithms especially for large processed data. At the same time, memory usage is nearly constant. This method resolves the balance between time and space complexity, and it is a useful reference for other methods.

Keywords: streaming data; XML data; XPath query; forest transducer

(Submitted on March 28, 2017; Revised on July 2, 2017; Accepted on August 10, 2017)

1. Introduction

In the big data era, with the development of wireless sensor network in internet of things and the popularity of mobile Internet, especially smart mobile terminals, massive data has been generated in the applications of environment monitoring, traffic flow management, financial transactions, public opinion analysis and social networks, etc. More than 80 percent of these data are semi-structure or unstructured. How to analyze and extract useful information to help decision-making is the main topic in big data research areas [25]. The data generated by the monitoring systems and sensor network are almost in the form of streaming. Streaming data is different from traditional structure data in how it arrives continuously, data process requires only once sequential scan, and return results on the fly. For example, in the system of fire detection, the query including temperature and concentration of smoke will be set in advance and kept in the processing engine. When the data gathered by sensors come into the engine in the form of streaming, the system will determine whether they meet specified set criteria in query. The specific location of fire occurs will be returned to the user if the conditions are satisfied. In the traditional relational database, the data will be persistent stored in external storage and the analysis will be done when the query is coming. Data are usually unchanged but query is on changing. Different from database processing pattern, in the streaming environment, query is unusually changed, but the data flow into the system continuously. To complete analysis task efficiently, the process will be done in main memory to reduce I/O operations with external memory. Because of the limited storage space available, cache data will be very limited, much less than the size of all data. How to efficiently and effectively process large scale streaming data in the limited main memory is the key problem in current research.

XML is a semi-structure markup language recommended by W3C. It has become the de facto standard for data representation and exchanging on web due to its flexibility of organizing data such as self-describing capability, flexible organization. XML is widely used in areas such as data representation, sharing, exchange in distributed computing, multiple

* Corresponding author.
E-mail address: zhixuehe@126.com.

Web systems. The purpose of querying data streams is to identify specific data patterns in the continuous flow of data, which match queries user presented. XPath [7] and XQuery [2] have emerged as the standards recommendations that are likely to receive broad support in most XML query languages. Since XPath forms an important core of XQuery, the problem of XPath query evaluation against streamed XML data is a fundamental problem in the context of XML.

For XML streaming data processing, automaton-based methods are more common due to their efficiency and clean design. But when the expression became complex, these methods face some problems such as state-space explosion, low efficiency and excessive usage of main memory, etc. Buffer management is an important problem in streaming processing for data arrive in uncontrolled velocity. If current data cannot be handled timely, the unprocessed data will be buffered and increase the pressure of main memory. At the same time, for the only once scan, the candidate results will be buffered, especially when multiple predicate exist in query. A simple example is querying XPath "/a[b]//c" in an XML fragment of the form "<a><c1>1</c1><c2>2 </c2>...<cn>n</cn>0", where c1 through cn have to be buffered until b arrives. The buffer is unavoidable, because the fact that elements in an XML stream may arrive in an order that does not match the order of the predicates that use them in the query, and due to recursive structure in the data, which leads to multiple matches for an input item. When the query contains "//" axis and multiple predicates, it is even more difficult to keep track of all the information needed for proper buffer management. Aiming at the problems above discussed, we propose a method to process XML streaming data base on forest transducer (FT), optimize the data structure and buffer management compared with paper [13]. In this method, the user's query will be converted to an instance of forest transducer. The input streaming data will drive structure match and state change of transducer. Abstract syntax tree (AST) is used to maintain the relationships between functions and the intermediate results in the transducer running procedure. Once the final results are got in the reduction of AST, they will be output immediately. This method meets the requirement of streaming data proceeding for one-pass scanning and returning results on the fly, and realizes the balance of efficiency and memory usage.

The rest of the paper is organized as follows. After the related works, we first describe some background information in Section 3, which including XML streaming data model, XPath query, and construction rules for FT. After that we present the framework and implementation of the algorithm for processing XML streaming data based on FT, in Section 4. An experimental study is provided in Section 5, and concludes our work in Section 6.

2. Related work

In context of XPath streaming evaluation, there are two problems are commonly studied [26,28]. One is stream filtering problem that for given a set of queries, the filter will determine which of them have a nonempty output on an incoming XML document stream. The other is the stream querying problem which finds all the matches of the output nodes of a given set of queries against the XML stream.

2.1. Stream-Filtering Algorithms

Several works have been focused on the problem of filtering a stream a XML document. For filtering, the input is a stream of XML document that is to be matched, and a query is matched if the result of evaluating the query on the document is non-empty.

XFilter is the first filtering system that addressed the processing of streaming XML data [1]. It adopts the finite state machine to represent path expression. If an accepting state is reached during parsing, the query matches to this document. YFilter, based on XFilter, uses sharing prefix method to combines multiple finite state machines in to a single nondeterministic finite automaton (NFA) [8]. All common prefixes of the paths are represented only once in the NFA. XTrie is built on the top of YFilter [5]. It proposed a trie-based index structure, which decomposes the XPath expression to substrings that only contain parent-child axis. The common substrings among expressions can be shared. It can process complex and multiple path expressions and supports both ordered and unordered matching. Another method is XPush. It can process a large number of XPath queries for creating an alternating finite automaton (AFA) for each expression [12]. Then all the AFAs be transformed into a single deterministic pushdown automaton (DPDA). Besides, there are other methods for filtering streaming data, such as index-based technique Index-Filter [4], Bloom-Filter method [10], and FiSt [16] that transforms twig patterns expressed in XPath and XML documents into sequence using prüfer's method, etc.

2.2. Stream-Querying Algorithms

The querying problem has attracted a lot of research attention. These algorithms can be classified into three categories according to the main data structure used: the automaton-based approach, the array-based approach, and the stack-based approach.

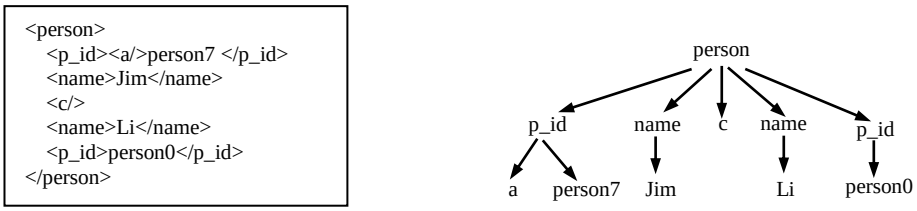
The elements, attributes and texts in XML streaming are usually abstracted to events. Automaton is driven by events for states changing, so it is a natural selection for streaming process in many systems. XSQ adopt pushdown transducer with buffers as the basic building block in its system design [20]. It supports the fragment of XPath query that includes child and descendant axes, aggregation and multiple predicates. But XSQ does not support the AND operator, the same node-labels in a query and requires that each axis node has at most one predicate node child. It needs to store all pattern matches for a query during its execution. So it suffers from exponential states blow-up and its worst case complexity can be exponential in the size of the query. SPEX maps the input XPath expressions to a network of transducers [18]. For each XPath sub-expression, it constructs an independent transducer. The transducer network is linear in the size of the query. A transducer accepts the input streams and outputs annotated streams with marks on selected nodes. So each coming stream event is processed by the entire network by default. The size of communication messages will reach the size of all input document stream. And it cannot process predicates efficiently. Paper [19] proposed an algorithm which uses double-layered non-deterministic finite automata(NFA) to process XPath queries involving the axis child, descendant, following siblings, and followings as well as predicates. This method divided the query into backbone part and predicate part. The first layer of NFA evaluates the backbone part and the second layer NFA handles the predicate part. It introduces state sharing and state pruning optimization techniques to avoid the exponential growth in the state size. Although this approach supports following and following sibling axes, it does not support logic predicates that include ‘and’, ‘or’, ‘not’. GCX [24] system proposes a buffer management scheme which combines static and dynamic analysis for efficiency in memory usage and streaming XQuery processing. This method combines the technique of projection for pre-filtering data which is irrelevant to query evaluation and the technique of dynamic garbage collection for automatic memory management. Other methods, for example, XEBT based on tree automaton [9] and the method of paper [27] based on pushdown automaton to deal with XPath query having complicate AND/OR sub-query.

For the methods that are based on stack, TwigM is a lazy streaming processing method, which extends the multi-stack framework of the TwigStack algorithm. It uses a compact data structure to encode pattern matches rather than storing them explicitly [6]. But in the processing, it has to examine a large number of stack entries for each incoming event and has to store multiple physical copies of candidate results at a time in different stack entries. This will cause the space problem for recursive data. LQ and EQ, based on the lazy strategy and on the eager strategy respectively, are proposed by paper [11]. Two algorithms implement efficient evaluation univariate XPath in O(|D|.|Q|) time where |D| is the document size and |Q| is the query size. They not only have time-efficient performance but also have better buffering space performance. They have also solved problems of the recursion in XML document and same node-labels in XPath expression. StreamTX is presented to supports tree pattern query with multiple output nodes [14]. In this method, the authors adapt the holistic twig joins for tuple-extraction queries on streaming with block-and trigger mechanism and pruning techniques. It needs to compute all the pattern matches of the query, which may contribute to the same solutions. So at the final stage it will eliminate duplicated solutions. TurboXPath is a representative of array-based methods [15]. It can evaluate XPath query with downward and upward axis, restricted form of FLOWR expression in XQuery. In TurboXPath, the input query is translated into a set of parse trees. The result of a query execution is a sequence of tuples of XML fragments matching the output nodes. This method is efficient for queries on non-recursive XML data. However, it exhibits exponential behavior for queries on recursive XML documents. Extended TruboXPath, a streaming algorithm X_{aos} is proposed by paper [1]. It can process an extension of tree pattern queries with reverse axes that are parent and ancestor. In this method, it converts all reverse axes into their symmetrical forward axes. The query will be built as a parse tree or o directed acyclic graph. It is not suitable for applications that require incremental outputs and the input is an unbounded stream, because, it output the answers at the process end.

3. XML data model and forest transducer

3.1. XML streaming data model

XML data are commonly modeled as a rooted, labeled and ordered tree, as DOM (Document Object Model). The XML data in Figure 1(a) can be model as DOM tree in Figure 1(b). Node in the tree corresponds to an element, an attribute or a value, and the edge represents (direct) element-subelement or element-value relationships. This data model requires the entire document to be read into memory to construct a DOM tree, but in big data era, many files become very large, the size scale can even reach terabytes (TB). It exceeds the storage capacity of memory and at the same time, there are a lot of data which are not relevant to user’s query, so it is not feasible to load all data in processing. Now, many applications have generated much data in the form of streaming. Streaming form XML data are parsed in order, typically as a sequence of events following preorder traversal of DOM tree. Callback function will be executed at the start event and end event of a data node. For example, in SAX [3], every element will generate start event at an open tag and end event at a close tag. If we define SE(e) is the start element event of e, and EE(e) is the end element event of e, then, the corresponding sequence of SAX events of XML document in Figure 1(a)is SE(person), SE(p_id), SE(a), EE(a) characters(), and EE(p_id).



(a) XML data (b) DOM tree

Figure 1. XML data and its DOM tree

3.2. XPath query

XPath is a standards query languages for XML, since it has been used in many XML applications and in some other languages for querying and transforming XML data, such as XQuery and XSLT. In this paper, we focus on a practical fragment of XPath defined as follows:

Path := Step | Step Path
Step := Axis NodeTest | Axis NodeTest '['Predicate']'
Axis := '/' | '//'
NodeTest := tag
Predicate := [Step] | [Step] Predicate

where Path is the query expression and compose single step or stem following anther query expression; A query step can be an axis added node test with or without a predicate. The axis can be either "/" or "//", which constraints the two matched nodes is either a P-C (Parent-Child) relationship or an A-D (Ancestor-Descendant) relationship. The node test target can be a node tag. The predicate is a query step or multiple step composition. For the XML data in Figure 1, if we want to find the name of a person and his p_id is equal "person0", the query can be expressed as:

Q1: /person [/p_id [text() = "person0"]]/ name/text().

3.3. Forest Transducer

Forest transducer processes XML data as a set of trees [21]. A XML tree can be expressed as node(x1)x2, where "node" is current visiting data node, "x1" is the set including children nodes of current data, and "x2" is the set including sibling nodes of current data.

Definition 1: XML forest XML document can be expressed as t1, t2, ..., tn, $n \geq 0$, where t_i ($0 \leq i \leq n$) is a subtree of XML data and formed with a root node and a sequence of other subtrees. The concrete definition as follows:

forest := ε | tree forest
tree := tag(forest), tag $\in \sum$
where $\sum$ is the set of tags.

Definition 2: Forest transducer is a 4-tuples as FM := (S, F, T, f0), where

S: a set of states and functions;
F: a set of transition functions, as forest×f → {f_i | f_i is the state function after matched};
T: a set of streaming data nodes;
$f_0 \in F$: the start state function.

In forest transducer or FT for short, state is defined as function, which will receive the streaming data in the form of forest. For XPath query defined in the Sec. 2.2, the rules for converting query from XPath to instance of FT is defined in Figure 2. The rules consist two parts: first part T1 is for data selecting path query, and second T2 is for Boolean query. For different forms of query composition, the rules define how to switch states and which functions should be executed according to the current visiting node. In the rule, "%t" is a special marker that represents any string that is not the same as the current node. Assuming that the rule's target matched node is "test", if current visiting node's tag is "test", the first rule will be selected for completing state switch and succeed functions execution, and the rule is successfully matched. But if current visiting node's tag is not "test", the second rule, the is the rule with "%t", is selected, and the match is failed. If at the end of streaming. there are no input data, "nil" means null as input.

The rule T1 is defined for converting XPath data-selection query or its sub-query to function. As for rule T1-2, query is in the form of "//test[preds]". Firstly, function q will judge the current visiting node whether match the target axis node "test" or not. If matched, q' judges the children data of the current nod will make the predicate "preds" hold. The result "test(x1)" will be returned if "preds" is satisfied. As the relationship in query is A-D, match operations will be done in children stream x1 and sibling stream x2. Rule T2 process the situation for the query has multiple parallel predicates. With rule T2-2, for example, the Boolean query is "[/test preds]". The function q firstly judges the current visiting node is matched with target node "test". If the matched result is true, q' will match the predicate preds in the child stream x1 of the current node. If q' get false result, it will continue to judge the query in the sibling stream x2 of current node. For parallel predicate, such as T2-4 "[child::test preds] preds' ", the query result is decided by logic and operation result of multiple parts. Other rules are similar with above but for the space limit, we will not explain them.

Rule for path query translation T1： Path→Name				
T1〖child::test preds〗		=	q	**（T1-1）**
	where q(test(x1)x2)	=	(q'(x1) ? test(x1) : nil) q(x2)	
	q(%t(x1)x2)	=	q(x2)	
T1〖desc_or_self:test preds〗		=	q	**（T1-2）**
	where q(test(x1)x2)	=	(q'(x1) ? test(x1) : nil) q(x1) q(x2)	
	q(%t(x1)x2)	=	q(x1) q(x2)	
T1〖child::test preds path〗		=	q	**（T1-3）**
	where q(test(x1)x2)	=	(q'(x1) ? q"(x1) : nil) q(x2)	
	q(%t(x1)x2)	=	q(x2)	
T1〖desc_or_self::test preds path〗		=	q	**（T1-4）**
	where q(test(x1)x2)	=	(q'(x1)? q'' (x1):nil) q(x1) q(x2)	
	q(%t(x1)x2)	=	q(x1) q(x2)	
for all	q(nil)	=	nil	
	q'	=	T2〖preds〗	
	q''	=	T1〖path〗	

(a) rules for data-selecting path query

Rule for predicate query T2: Preds→Bool				
T2〖nil〗		=	True	**（T2-1）**
T2〖[child::test preds]〗		=	q	**（T2-2）**
	where q(test(x1)x2)	=	q'(x1) OR q (x2)	
	q(%t(x1)x2)	=	q (x2)	
	q(nil)	=	False	
	q'	=	T2〖preds〗	
T2〖[desc_or_self::test preds]〗		=	q	**（T2-3）**
	where q(test(x1)x2)	=	q'(x1) OR q (x1) OR q(x2)	
	q(%t(x1)x2)	=	q (x1) OR q(x2)	
	q(nil)	=	False	
	q'	=	T2〖preds〗	
T2〖[child::test preds] preds' 〗		=	T2〖[child::test preds]〗AND T2〖preds' 〗	**（T2-4）**
T2〖[desc_or_self::test preds] preds' 〗		=	T2〖[desc_or_self::test preds]〗 AND T2〖preds' 〗	**（T2-5）**

(b) rules for predicates path query

Figure 2. Forest transduce conversion rules

4. Streaming data processing method

In this section, we first give the framework of XSPFT method that is processing XML streaming data by forest transducer, and then present the detailed implementation for processing XPath query. At the last, we analyze the time and space complexity of this method.

4.1. Framework of algorithm

Firstly, forest transducer is constructed for user's query according by the construction rule. Then the transducer instance is running driven by input streaming data. In the processing, abstract syntax tree is used to maintain the relationships between states and functions of FT and the intermediate results. The output nodes will get when reducing AST's subtree. The concrete steps as follows:

Step 1: According to the rules defined in Sec2.3, we construct the forest transducer instance of XPath query

Step 2: Initialize the running environment of FT, which includes the stack, AST and global variables

Step 3: The functions are executed and the states are switched in the running of FT, which are driven by the input streaming XML node data

Step 4: The intermediate results are represented by abstract syntax tree, and the tree node of the executed function is replaced by this tree. The query results output as soon as query structure and conditions are satisfied in the procedure of AST reduction

4.2. The implementation of XSPFT

The framework of the algorithm of XSPFT is as follows:

Algorithm: execFT
Input: XML Stream S, XPath query Q
Output: the answer ans
01 convert Q to FT_Q
02 funcID = 0;
03 S = new Stack( );
04 T = new AST( );
05 level = 0;
06 while(not end of S)
07 n = incoming element
08 case SE(n):
09 n.level = level++;
10 while(S.top.targLevel≥n.level)
11 funcs = S.pop( );
res = {};
12 for each function f in funcs
13 res = res ∪ **match**(f, n);
14 fSet = group res by target level
15 push fSet into stack ordered by level from small to large
16 case EE(n)
17 level--;

Firstly, the XPath query Q will be converted into forest transducer instance according to the rules defined in Sec. 3.3(Line 01). Then, initialize the global variables in the running process, which include functionID, S and T. The variable functionID is the identification of each function is the FT, which can help quickly visit the AST node and differentiate functions with the same names. The stack S is used for buffering functions which will be executed in the future and the functions that will be reduce at the event of the close tag. The item S is a set of triples, every triple is represented as <funcID, func, targetLevel>, where func is the name of function, and targetLevel is level value of the target node that the func want to matched. S is initialized by the set that includes only the function q0. The order of set pushed into S is: the set includes functions which target input is the sibling subtrees of current visiting node, the set of reduced functions, the set include functions which include the child subtrees of current visiting node. Abstract syntax tree T is used to maintain the relationship between functions and the intermediate results. The root node is function q0, means that the query result of Q is the processing result of q0. After the action of match, a AST subtree is constructed to represent the matched result, and the current executed function node in the T is replaced by this subtree. The variable level records the depth of the node in the streaming data, which can be used for judgment of P-C or A-D relationship between two nodes. When the data node comesinto this system, the algorithm will pop the top element of S according to the node's level, and then execute the functions in the popped set to implement states switch (Line 06-15). In the procedure of reduction of T, the query results can be output if the subtree can be reduced from an expression into a value.

The P-C or A-D relationship between nodes can be calculated by the level value, the property is described as below:

Poperty1: Two elements x and y satisfy A-D relationship if and only if y is included by x, that is y is in the scope of x's start and end of label; if the y's level value is lager one than x, then they are P-C relationship.

For example, in Figure 1 (b), the level value is 1 for the "person" node, and the nodes between its begin tag and end

tag are all its descendants, for instance p_id, a and so on, they satisfy A-D relationship. For p_id node, its level is 2, and in the scope of person, so the two nodes satisfy P-C relationship.

In the process of FT running, the state switch and FT's function execution algorithm is described as below:

Procedure match
Input: the MFT function f, data node n
Output: match result set res
01 if f.targetLevel > n.level then
02 res = execute rule f(nil) in FT_Q;
03 else
04 res = execute rule f(n) in FT_Q;
05 for each function u in res
06 funcID++;
07 add funcID to u;
08 generate subtree st of res;
09 replace f node in T by st;
10 ans = ans ∪ **reduce(T)**;
11 return res;

In the algorithm of **match**, if the level value of current function is larger than the level of current visiting node, then execute the FT rule by null input. Otherwise, call the rule and the input data is the current node.

Poperty2: In XML streaming data, the level value is changed from m to k, and $k \leq m$, then there are no data in m, m-1, m-2, ..., K+1 level.

The match result of FT rule includes three data types: result nodes for data select queries, true or false value for predicate queries, and the functions that will be processed. In this method, we add the ID attribute to every function to get the tree node of the function in AST and differentiate the functions with the same name. For recording the relationship of functions we use the matched result and the intermediate result nodes. the matched result will be represented as an abstract syntax subtree to replace the node of current function in AST. AST will be reduced when getting the partial results, and this is consistent with the character of streaming. As for the FT rule T-1, we can see that query results are composed by many functions matched and the results will be reduced at the second level in AST, so as soon as the result is got in 2nd level, it will be output.

The operators of AST reduction include:

(1) "+" represents the composition of results of two functions.
(2) "?" represents the condition judgment, if the first child of the subtree can be reduced into "True", then return the second child as matched result, else return "nil". Note that in the rules of FT, all the ternary operation's third children are 'nil', so they will not be included in the AST.
(3) "OR" represents logic OR operator, only one child can be reduced into "True", the subtree can be reduced into "True".
(4) "AND" represents logic AND operator, when all children can be reduced into "True", the subtree can be reduced into "True".

The maintenance and reduction of AST have great influence for efficiency of this method and we adopt the "earliest reduction" strategy, that is after getting the match result for a function, the reduction operation will be executed, instead of waiting for all the match results. The reduction operation methods are defined as follows:

(1) When the subtree's root is "+", if the child reduced into "nil" then it can be removed as the null value will not contribute to the final result.

(2) When the subtree's root is "OR", if the child reduced into "False" then it can be deleted, because in the OR expression, "False" value's remove have no influence for other operations.

(3) When the subtree's root is "OR", if one of its children is reduced into "True", then other functions in this tree can

be marked with “N” represent that these functions will not be compute in future, and the tree is reduced into “True”.

(4) When the subtree’s root is “AND”, if the child reduced into “True” then it can be deleted, because in the AND expression, “True” value’s remove have no influence for other operations.

(5) When the subtree’s root is “AND”, if one of its children is reduced into “False”, then other functions in this tree can be marked with “N” represent that these functions will not be compute in future, and the tree is reduced into “False”.

These rules are summarized as:

(1) + : x+nil = x
(2) OR: x OR false = x
(3) OR: x OR true = true
(4) AND: x AND true = x
(5) AND: x AND false = false

Example 2: In Figure 3, the left tree can be reduced into the right if the subtree rooted with “OR” has one child reduced into true. In this “OR” subtree, the function q1 will not be matched, because its result will not change the final “True” value.

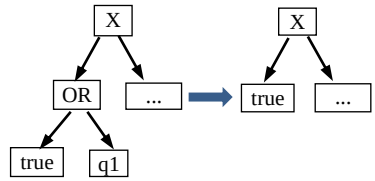


Figure 3. Reduction of abstract syntax tree

4.3. A case

In this section, we will show the detail of streaming processing method base on forest transducer.

Example 3: For the streaming data which gave in Figure 1 and the XPath query
Q1: /person [/p_id[text()=”person0”]]/name/text(),
the concrete process step as follows:

(1) The forest transducer instance is firstly constructed according the rules defined in Sec. 2.3.
q0(person(x1)x2) = (q1(x1)?q3(x1):nil) q0(x2)
q0(%t(x1)x2) = q0(x2)
q0(nil) = nil
q1(p_id(x1)x2) = q2(x1) OR q1(x2)
q1(%t(x1)x2) = q1(x2)
q1(nil) = false
q2(person0(x1)x2) = true
q2(%t(x1)x2) = q2(x2)
q2(nil) = false
q3(name(x1)x2) = q4(x1) q3(x2)
q3(%t(x1)x2) = q3(x2)
q3(nil) = nil
q4(text(x1)x2) = text() q4(x2)
q4(%t(x1)x2) = q4(x2)
q4(nil) = nil

(2) Initialize the global variables in running, for functionID = 0, S:{<0, q0, 1>} and T is added a rooted node as q0;

(3) Read into the streaming data, pop the top element in S, and match this data node with functions' target node, the functions formed as a set is at the top of stack S. Then update T and S with the match results.

① For the data in Figure 1, "person" node is firstly read, its level value is 1. The set {<0, q0, 1>} popped from S, and the function q0 will be executed according to the rules in the FT instance. The matched result is (q1(x1)?q3(x1):nil) q0(x2), and functions in this results q1, q3, q0 will be added an attribute functionID. The result will be expressed in the form of AST (q1[1]?q3[2]:nil)+q0[3] to replace the tree node q0[0] in T, where "+" is the connection operator for composing the left and right part result of "+". The function q1's target data is the children of the current visiting node, so its target level value is 2. Similarly, function q3's target level is 2. The function q0's target data is the sibling of current visiting node, so its target level is 1. The grouped set are {<1, q1, 2,>, <2, q3, 2>} and { <3, q0, 1>} according to the target level value. Then the sets will be pushed into S by ascending order, S is updated as {<1, q1, 2,>, <2, q3, 2>} { <3, q0, 1>}.

② The "p_id" node is read into this algorithm, and its level value is 2. The element popped from S is set {<1, q1, 2,>, <2, q3, 2>} include function q1 and q3. According to the rule of FT instance, current visiting node is matched with q1's target node, the matched result is q2[4] OR q1[5], which will replace the q1[1] node in T. For function q3, the "p_id" node is not its target, then the result is q3[6] will update the node q3[2] in T. All the functions in results will be grouped by their target level, and the grouped sets will pushed into S. The newest stack S is {<4, q2, 3>} {<6, q3, 2>, <5, q1, 2>} {<3, q0, 1>}.

③ The rest data node is processed in the similar way. Note that when visiting the first "name" node, S's state is {<q2, 3, 08>} {<q3, 2, 06>, <q1, 2, 05>} {<q0, 1, 03>}. The function q2's target node level is 3 in the top element of S, but "name" node's level value is 2, then q2 is matched for input data as nil.

The patterns of T at different times for at the beginning, "person" node, "p_id" node and at the end of computing are shown is Figure 4.

(4) In the end, the AST T is shown as T3 in Figure 4, reduce T and get the result as JimLi.

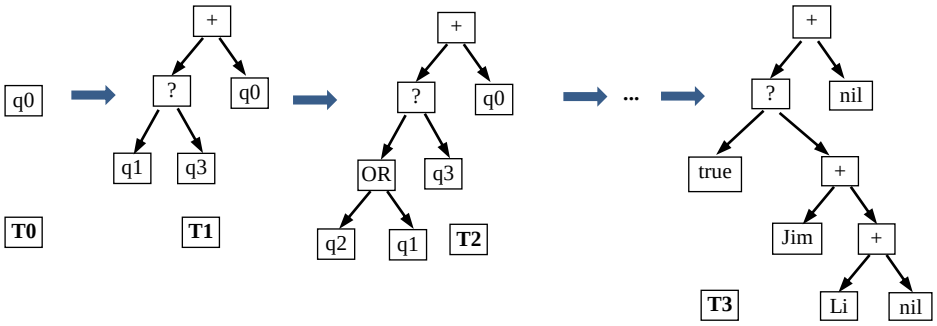


Figure 4. Reduction process of AST

4.4. Analysis

In the running process of method based on FT, each data node will be analyzed when it flows into the system. If the data's size is n, and XPath query Q include |Q| query steps, the time complexity of our method is O(n|Q|). In the implements of this method, stack and abstract syntax tree are the main data structure. Stack is used for keeping functions that will be matched for follow-up streaming data node, and its largest depth is the embedded depth of XML data organized. If the query result scale is O(m), abstract syntax tree's size is O(m|Q|). The size is related to the output result and independent of the streaming data.

5. Experimental results and analysis

To verify the effectiveness and efficiency of this method, we adopt XML standard data sets to test and analyze two metrics: (1) the efficiency of this method for streaming data; (2) the memory usage in the running time. For the current situation of methods and engines for XML streaming, we choose paper [13] methods named as FTX and GCX as the compared objects. In the experiment of GCX, results have indicated that GCX has obvious advantages in processing efficiency and memory usage than FluXQuery, Galax, MonetDB, Saxon and Qizx/open. FTX is the newest method for XQuery streaming by forest transducer.

Running time is the time spent from the beginning parsing the input XML data document to the end element event of last, and memory usage is the maximum usage of main memory that each method consumes during query processing. Each query is executed ten times, and the average value is calculated and compared.

5.1. Experimental settings

We have implemented this method in Java and ran all the experiments on 3.6 GHz Intel i7-4790 CPU with 8GB memory at Window 7 operating system. In our experiments, we use both real and synthetic datasets that differ in size and characteristics. We test the performance of our method on two real datasets, DBLP [17] and Treebank [24] , and one synthetic dataset, XMark [22]. Through SAX analyze these datasets to simulate the data coming in streaming format. For efficiency analysis, the average value of ten running times in different data scale is selected for comparing. For main memory usage, only XMark data as the input, because FTX and GCX experiment has been done for this dataset. This selection can make a targeted comparison.

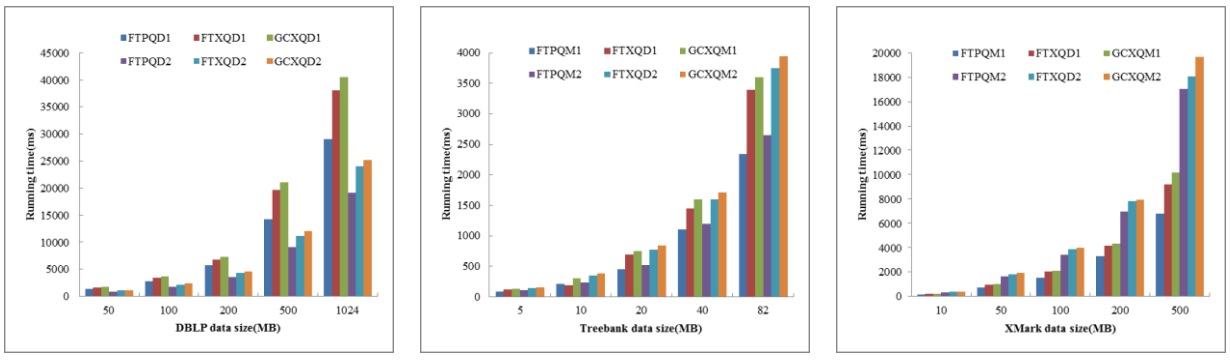
5.2. Running time performance

The queries for three datasets are listed in Table 1. These queries are representative where QD1 and QD2 mainly test P-C relationship processing in DBLP, which is shallow but wide; QT1 and QT2 mainly test multiple A-D relationship in query in TreeBank, which is narrow and deeply recursive; XMark is a well-known benchmark dataset, it has the features of data-centric and document-centric for XML. Document generator of XMark can be parameterized to get different size data. In this experiment, we generate five data different in scale to test QM1 and QM2 for comparison with TTX and GCX.

Table 1. Test data sets and queries

Data set	Query expression
DBLP	QD1:/dblp/article[author][year]/title
	QD2: /dblp/proceedings[title]/author
TreeBank	QT1://S[.//VP[.//JJ][.//VBD]]//NP[.//WP]//DT
	QT2://S[.//NP[.//DT][.//NN]]//PP[.//TO]//NN
XMark	QM1://item[name]//mail[text//emph]
	QM2://closed_auction/annotation/description//emph

Figure 5 shows the running time performance for three datasets. In this figure, for every dataset, there are two queries are processed which are added the prefix whit FTP (our method of **F**orest **T**ransducer for X**P**ath), FTX [14] and GCX to distinguish the results. DBLP has reached the size of 1GB, we chose the data size range from 50MB to 1024MB. For TreeBank, the free dataset from the Internet is 82MB, so we change the data size from 5MB to 82MB. XMark can be generated at arbitrarily size; we choose 10MB to 500MB for comparison with FTX and GCX. From the results of running time, we can see that FTP is higher than FTX and GCX in running efficiency percentage. This variation range exists because of the differences in organizing structure of data and the complexity of queries. For instance, TreeBank dataset is deeper than the other two and complex nested. The FTP method has an obvious improvement in efficiency for processing QT1 and QT2, which have multistep query axis including A-D relationship. This embodies the advantage of our method.



(a) DBLP dataset (b) TreeBank dataset (c) XMark dataset

Figure 5. Comparison for running time

5.3. Memory space performance

We select FTX and GCX methods as the compared objectives for FTX is an efficient implement for forest transducer and GCX proposed dynamic garbage collection technique in its implement. In streaming processing, the main memory usage is very small and will keep constant in FTX and GCX. In this experiment, we chose XMark dataset as input for GCX's experiment have been done in this dataset, and the queries are:

Q1://site/open_auctions/open_auction[/bidder[personref/personref_person/text()="personXX"]]

Q2://site/closed_autions/closed_aution[/annotation/description/parlist/listitem/parlist/listite/text/emph/keyword]/seller/serller_person.

These two queries are long and include many predicates for selecting process. So they are suitable for testing the memory usage. The data scale range from 100MB to 20GB. Table 2 shows the performance in terms of memory usage.

Table 2. XMark data set memory usage

Query	Data set（GB）	FTP Method(MB)	FTX(MB)	GCX(MB)
Q1	0.1	2	2.7	1.4
	1	3.6	4.5	2.0
	10	6	8.4	4
	20	10	15.8	5.4
Q2	0.1	2.2	3.0	1.7
	1	4.2	6.4	3.3
	10	6.9	11.3	4.8
	20	12	17.5	6.7

The experiment results show that the usage of main memory will increase with the data size becoming larger. There are two reasons for twofold: one is Java virtue machine's garbage collection mechanism is different with GCX, some useless data will be kept in memory for some time; the other is that the increasing of data size will enlarge the size of abstract syntax tree, and at the same, the reduction results can't transfer immediately which will take some memory. Note that, the usage of FTP and FTX is larger than GCX. This is because in GCX implement, it specially considers the problem of garbage collection dynamically and can keep the usage at the constant level. This is a point of optimizing our method in the future. But FTP is more efficient than FTX because utilizing of AST. From an overall perspective, FTP memory usage is much less than the capability of hardware provides even if the size reaches the GB or TB scale. FTP method can satisfy the requirement of streaming processing in efficiency and memory usage.

6. Conclusion

XML is the de facto standard for data representation and exchange between multiple application and data sources on the Web. There are more and more data in the monitor and analysis system form streaming. In this paper, we propose the method for XPath query processing based on forest transducer. This method considers the requirements of streaming data to be read sequentially and only once into memory, and the query must be processed on the fly. It balances the running efficiency and memory usage. In this method, user's query is translated into forest transducer instance, the incoming data node drive the state switch and function execution. Abstract syntax tree is used for maintaining the relationship between state functions and buffering intermediate results. With the reduction of AST, the partial query result will be output as soon as possible. The experimental results show that this method is more efficient and effective for streaming data and requires a smaller usage of memory. These features are of great importance for streaming data processing.

As for the future work, we will focus on two aspects, one is to enlarge the scope of XPath operators, and the other is how to optimize the operation of matching and reduction.

Acknowledgements

This work was supported by the National Science Foundation for Young Scholars of China (61202074), Municipal Natural Science Foundation of Beijing (4122011), the Higher Education Scientific Research Project of Hebei Province (QN2016248) and the Science and Technology Development Program of Hebei Province (15210126).

References

1. M. Altinel and M. J. Franklin, "Efficient Filtering of XML Documents for Selective Dissemination of Information," in *Proceedings of the 26th International Conference on Very Large Data Bases*, pp.53-64, Cairo, Egypt, September 2000
2. C. Barton, 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*, pp.455–466, Bangalore, India, March 2003
3. S. Boag, D. Chamberlin, M.F. Fernández, D. Florescu, J. Robie, and J. Siméon, "XQuery 1.0: An XML Query Language (Second Edition): W3C Recommendation," (2015-09-07). http://www.w3.org/TR/xquery/, 2016
4. D. Brownell and D. Megginson, "SAX: Simple API for XML," (2004-04-27). http://www.saxproject.org/, 2016
5. N. Bruno, L. Gravano, N. Koudas, and D. Srivastava, "Navigation-vs. Index-based XML Multi-query Processing," in *Proceedings of the 19th International Conference on Data Engineering*, pp.139−150, Bangalore, India, March 2003
6. C. Chan, P. Felber, M. Garofalakis, and R. Rastogi, "Efficient Filtering of XML Document with XPath Expressions," *The VLDB Journal*, vol. 11, no. 4,pp. 354-379, December 2002
7. Y. Chen, S. B. Davidson, and Y. Zheng, "An Efficient XPath Query Processor for XML Streams," in *Proceedings of the 22nd International Conference on Data Engineering*, pp.79-79, Atlanta, GA, USA, April 2006
8. J. Clark and S. Derose, "XML Path Language (XPath) Version 1.0: W3C Recommendation," (2015-09-07). http://www.w3.org/TR/xpath/, 2016
9. Y. L. Diao, M. Altinel, M. J. Franklin, H. Zhang, and P. Fischer, "Path Sharing and Predicate Evaluation for High-performance XML Filtering," *ACM Transactions on Database Systems*, vol. 28, no. 4, pp.467-516, December 2003
10. J. Gao, D. Q. Yang, S. W. Tang, and T. J. Wang, "Tree Automata based Efficient XPath Evaluation over XML Data Stream". *Journal of Software*, vol. 16, no. 2, pp.223-232, February 2005
11. X. Q. Gong, W.N.Qian, Y. Yan, and A. Zhou, "Bloom Filter-based XML Packets Filtering for Millions of Path Queries," in *Proceedings of the 21st International Conference on Data Engineering*, pp.890-901, Tokyo, Japan, April 2005
12. G. Gou and R. Chirkova, "Efficient Algorithms for Evaluating XPath over Streams," in *Proceedings of the 2007 ACM SIGMOD International Conference on Management of Data*, pp.269-280, Beijing, China, June 2007
13. A. Gupta and D. Suciu, "Stream Processing of XPath Queries With Predicates," in *Proceedings of the 2003 ACM SIGMOD International Conference on Management of Data*, pp.419−430, San Diego, California, USA, June 2003
14. S. Hakuta , S. Maneth , K. Nakano , and H. Iwasaki. "XQuery Streaming by Forest Transducer," in *Proceedings of the IEEE 30th International Conference on Data Engineering*, pp. 952-963, Chicago, USA, March 2014.
15. W.S. Han, H.F. Jiang, H. Howard, and Q. Li, "StreamTX: Extracting Tuples from Streaming XML Data," in *Proceedings of the VLDB Endowment*, pp.289-300, Auckland, New Zealand, August 2008
16. V. Josifovski, M. Fontoura, and A. Barta, "Querying XML Streams," *The VLDB Journal*, vol. 14, no. 2, pp. 197-210, April 2005
17. J. Kwon, P. Rao, B. Moon, and S. Lee, "FiST: Scalable XML Document Filtering by Sequencing Twig Patterns," in *Proceedings of the 31st International Conference on Very Large Data Bases*, pp.217-228, Trondheim, Norway, August 2005
18. M. Ley. DBLP XML Records. http://dblp.uni-trier.de/xml/
19. D. Olteanu, "SPEX: Streamed and Progressive Evaluation of XPath," *IEEE Transactions on Knowledge and Data Engineering*, vol. 19, no. 7, pp.934-949, July 2007
20. M. Onizuka, "Processing XPath Queries with Forward and Downward Axes over XML Streams," In *Proceedings of the 13th International Conference on Extending Database Technology*, pp.27-38, Lausanne, Switzerland, March 2010
21. F. Peng and S. C. Sudarshan, "XSQ: A Streaming XPath Engine," *ACM Transactions on Database Systems*, vol. 30, no. 2, pp.577-623, June 2005
22. T. Perst and H. Seidl, "Macro Forest Transducers," *Information Processing Letters*, vol. 89, no. 3, pp.141-149, February 2004
23. A. Schmidt, F. Waas, M. 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*, pp.974-985, Hong Kong, China, August 2002
24. M. Schmidt, S. Scherzinger, and C. Koch, "Combined Static and Dynamic Analysis for Effective Buffer Minimization in Streaming XQuery Evaluation," in *Proceedings of the 23rd International Conference on Data Engineering*, pp.236-245, Istanbul, Turkey, April 2007
25. A. Taylor, M. Marcus, and B. Santorini, "The Penn Treebank: An Overview," *In Treebanks*, pp.5-22, Springer Netherlands, 2003
26. X. D. Wu, , X. Q. Zhu, G. Q. Wu, and W. Ding, "Data Mining with Big Data," *IEEE Transactions on Knowledge and Data Engineering*, vol. 26, no. 1, pp.97-107, January 2014
27. X. Y. Wu, and D. Theodoratos, "A Survey on XML Streaming Evaluation Techniques," *The VLDB Journal*, vol. 22, no. 2, pp.177-202, April 2013
28. W. D. Yang, Q. M. Wang, and B. L. Shi, "Complex Twig Pattern Query Processing over XML Streams," *Journal of Software*, vol. 18, no. 4, pp.893-904, April 2007
29. W. D. Yang, and B. L. Shi, "A Survey of XML Stream Management," *Journal of Computer Research and Development*, vol. 46, no. 10, pp.1721-1728, October 2009

Zhixue He is a Ph. D. candidate from the Beijing University of Technology. And he is a lecturer of the North China Institute of Aerospace Engineering. His research interests include database theory and application, XML query processing, parallel and distributed computing.

Husheng Liao is a professor and PhD supervisor in College of Computer Science, Beijing University of Technology. He is a senior member of China Computer Federation. His main research interests include database theory, software automated method, compiling technique.