Research on Parallel Query of XML Stream Data Based on Pushdown Transducers

Hongliang Xie[1,*] Husheng Liao[1] and Hongyu Gao[1]

[1] Faculty of Information Technology, Beijing University of Technology, Beijing, China

[*]Corresponding author

***Abstract*—The social networking, network monitoring and financial applications have a need to query high rate streaming of XML data, but previous methods for executing XPath queries on streaming XML data have not kept pace with multicore CPUs. Data parallel query methods can improve processing efficiency by using multi-core CPU. Push down transducer as a special type of automaton, it not only can be used to handle XML data, and its internal stack structure can preserve the important information in the processing, so it can combine with the data parallel querying to improve the processing efficiency further. In order to improve the efficiency, transfers XPath query pattern into a series of ordered subquery which its result are valid and has easier logic.**

Keywords-XML stream; XPath; ordered tree pattern; pushdown transducer; data parallel

I. INTRODUCTION

XML (Extensible Markup Language), a semi-structured data representation language, is widely used in data transmission and storage. Network analysis, financial securities, sensor networks and other applications often via XML data stream for information transmission. The data in the streaming mode is continuous, not indexable, and the amount of data is large, causing the XML stream data to be stored impossible. Therefore, the stream data should be scanned once and obtained the result.

Begins from the the Y Filter algorithm to the XPath task parallel query method [1] [2] [3], then PSJ [6], p-DFA[7] approach. They can be used for XML data query processing, but there are flaws and deficiencies, such as the query efficiency is not high, you need to save a lot of intermediate results, limited ability to express, cannot handle XML stream data. The query method combining XML stream data processing and data parallelism [5] can be used to query XML stream data, but when there is a predicate in the query, it will lead to two aspects of processing in the link query phase: 1. link processing between block results; 2. link processing of sub queries processing results in block. Since the link query processing is executed serially, limiting the processing efficiency of the link query phase. Although there are many techniques for XML stream querying, the constraints of its existence limit the efficiency of XML stream data query.

In order to overcome the above shortcomings, this paper proposes a PPPTransducers based on pushdown transducer and data parallelism is used to perform XPath query on XML stream data under multi-core architecture.

Major contributions:

1. Design a new pushdown transducer to complete the tree mode query and have higher efficiency.
2. Transfer XPath query mode to an equivalent a group ordered tree patterns, making the processing logic of the automata more simple and the higher efficiency in the link processing phase.
3. Combine data parallelism with task parallelism, making full use of multi-core computing resources.

By the experimental results, PPPTransducers eliminates the processing between different sub-query results, and has higher efficiency for XML stream data query processing.

II. BACKGROUND

A. *XML Data Parallel Processing*

The data parallelism is that the continuous XML data is divided into uniform data blocks, then the data blocks is processed in parallel, finally, the results of all data blocks will be linked to get the final result. Relative to the serial processing, it has a higher efficiency.

In 2010, Zhang Y, Pan Y, Chiu proposed p-DFA automation [7]. Each block is processed using the same automaton. In order to remove the repeated state, site a set to store all automation state. Finally, get the results of the processing through linking block results.

In 2006, Alur R, P. Madhusudan proposed a dPDAs automaton [8], which uses a table to hold the input characters and the corresponding conversion rules, and complies the corresponding push or pop operation, according to the label.

Parallel computing can effectively improve the efficiency of processing in many areas. Many technologies of XML data processing use parallel strategy. However, although the existing methods can be used to deal with XML stream data, but some have low query processing efficiency when query with predicate, and some in the process will store a large number of state or require strict load balancing requirements.

B. *XML Data Processing Based On Automata*

Automation is a common and effective way to process XML. There are many theories based on automation of processing of XML data, which have their own characteristics and application fields.

In 2016 He Zhixue[4]proposed based on the list of extensible markup language QX List. When the algorithm is running, the tags in the XML stream data are filtered, and the tags contained in the query mode are cached. Finally, adopt hierarchical number match the buffered labels. Although the XML stream data is processed under limited memory, efficiency is not ideal.

In 2013, Ogden P, Thomas D, Pietzuch P[5]proposed scalable push-down transducer PPTransducer.XML stream data be divided to take data blocks, data blocks parallel processing, processing results execute link process to get match results. When XPath query contains predicates, it needs to rewrite the XPath, getting a set of subquery. In link process, 1. link processing between block results; 2. Link processing of sub queries processing results in block.

There are a lot of techniques for automating XML data processing, but there are still many shortcomings, such as inefficiency and lack of expressiveness.

III. Design Ideas

PPPTransducers first divides the XML stream data into blocks and use the automatons to process data blocks in parallel; then it link the results of the block processing with link query to find the matching of multiple data blocks result. In order to improve the efficiency of link query, we order XPath query, getting a set of orderly query tree model, each query tree model is in line with the query requirements, and the automation processing logic is simpler.

A. *Order X path*

Xpath query can be represented by a query tree pattern, and Figure I (1) is / a [b [c]] // e corresponding query tree mode. a node has b, e two sub-nodes, where b node contains a child node c, e node as a node descendant node; in matching, only need to meet each node, but does not require consider the order of matching nodes.

In order to simplify the link query, PPPTransducers use an orderly query tree pattern, that is, each node must follow the order of left to right. Obviously, XPath can be equivalent by a limited number of ordered tree pattern, for example: / a [b [c]] // e, can get 5 ordered query tree patterns. As shown in Figure II, the sum of the results of the five ordered query tree patterns is equivalent to the XPath result.

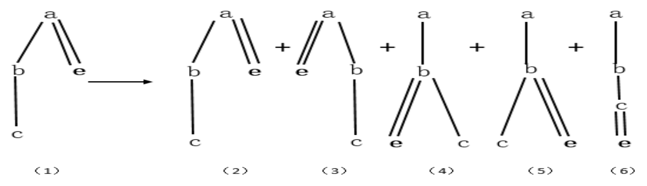


FIGURE I. THE RELATIONSHIP BETWEEN XPATH AND ORDERED QUERY TREE MODE

B. *The Construction Of The Automaton*

The automaton construction stage constructs a pushdown automaton for each ordered query tree pattern. For the five ordered query tree patterns in Figure I, will obtain five pushdown automata, through the automaton construction phase. Figure I (3) of the push down transducer is shown in Figure II.

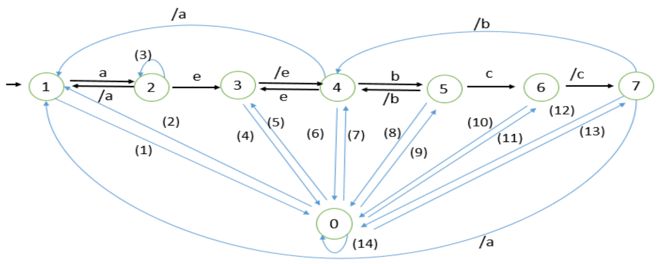


FIGURE II. Figure. 1 (3) TREE MODE CORRESPONDING AUTOMATA

The start and end states of the automaton are 1 state, and the transition and transfer of state 0 are specific, as described in Table 1. It is not difficult to see the correspondence relation between the structure of the automaton and the query tree pattern.

TABLE I. FIGURE.2. SIGN EXPLANATION

Number	Meaning	Number	Meaning
(1)	A non-a start tag	(2)	the end of the label in (1)
(3)	complete label	(4)	any start tag
(5)	end of the label in (4)	(6)	start tag other than b, e
(7)	end of the label in (6)	(8)	A non-c start tag
(9)	end of the label in (8)	(10)	any start tag
(11)	end of the label in (10)	(12)	any start tag
(13)	end of the label in (12)	(14)	complete label

C. *Data Partitioning*

The data partitioning phase splits the incoming XML stream data to provide data for the block processing phase. When the data is divided, the size of the data block is constrained, and when the data block reaches this constraint, a block of data is generated. At the same time, the integrity of the data structure is taken into account. If the size of the data block is not reached before reached complete block, also produces a block of data. As shown in Figure III, three data blocks have been divided.

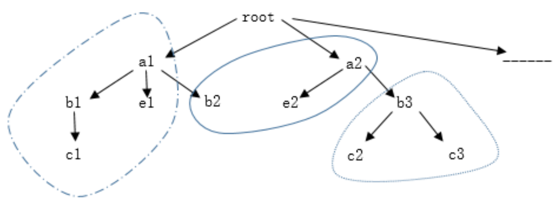


FIGURE III. XML DOCUMENT TREE

D. *Block Processing*

The block processing phase processes the data blocks generated by the data partitioning phase. All automation will process the received data block.

The automaton uses a list to store the processing results of the data blocks. Each record in the list represents a processing path. The record structure is a five-tuple (qs, zs, qf, zf, o):

- qs indicates where the current data block is processed from the state of the automaton;
- zs used to store the end tag did not match in current data block;
- qf indicates the state of the automaton when the data block is processed;
- zf used to store the start tag did not match in current data block;
- o store results satisfying ordered query tree pattern in data block.

When the data block arrives, the automaton follows the following steps:

1. For the first block, the start state of the automaton is used as the start state;
2. For other block data, since the start state cannot be determined, each state in the automaton is taken as a start state, generating a processing path and placed in the list;
3. According to the current state and the label in the data block, takes state transfer; if the transfer cannot be made, the record is deleted from the collection. Since the automaton is a non-deterministic automaton, the same input tag may cause multiple state transitions, resulting in multiple paths.
4. Repeat 3 until the data in the block is processed;
5. All the end tags that no match in the state transition path, fill their corresponding states to the start stack in turn;
6. All the start tags that no match in the state transition path, fill their corresponding states to the complete stack in turn.

For example, the three data blocks in Figure. III are processed using the automaton of Figure. II, and the results are shown in Figure. IV. When processing the second block, use all the states of automation as the start state and process in accordance with the order of arrival of label. During matching and state transition, paths with 1, 3, 5, 6 as the start state cannot continue to match and are removed. Retention is recorded at 0,2,4,7 as the starting state. The processing path with the start state of 0 is differentiated out of 6 transfer paths. Since the second block has an unmatched end tag </ a1>, so the start stack of the six processing paths is filled with 0, 3 4, 5, 6, 7, as shown in Figure IV (b). After encountering an unmatched start tag <a2>, the stack is filled. The processing of the other states is similar. The final result is shown in Figure IV (b). For the seventh record (2, 1) -> (2, 1, ε) in Figure. IV (b), it can be seen that the record is processed from the 2 state, and the 1 state in the start stack represent </ a1> no match; the 1 state in the finish stack indicates that <a2> is no match.

E. Link Query

The link query phase treats the results of the data partitioning phase as input, looking for results that match queries across multiple data blocks. Figure. IV as an example, the results of Figure. IV (a) and Figure. IV (b) are first linked, getting(1,ε)→(3,[1],ε)and(1,ε)→(4,[1],ε), then, links with each of the items in Figure IV (c), as shown in the Figure. IV.

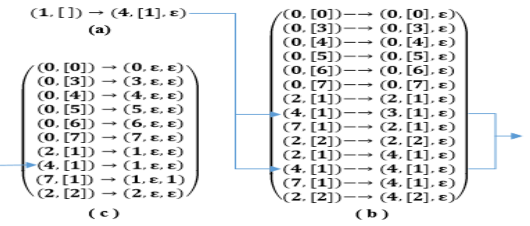


FIGURE IV. RUN TIME COMPARISON

IV. SYSTEM DESIGN

The PPPTransducers is composed of five steps (shown in Figure 5): XPath order, automaton construction, data partitioning, block processing, link processing. Among them, XPath order and automaton construction completed in the compilation phase, block processing, data block processing, link query parallel in execution. Block processing will start multiple block processing threads in parallel execution, each block processing thread within a number of automations to processing of data.

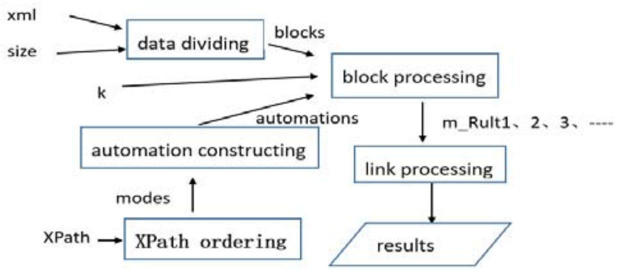


FIGURE V. PPPTRANSDUCERS PROCESSING FLOW CHART

XPath order: Occurrence step and the each predicate of XPath is ordered, the output is the order of the query tree model. Automata Constructor: Constructs a pushdown automaton for each query pattern. Data partitioning stage: According to a threshold size, the incoming XML stream data is divided into uniform size and the data block stored in the Blocks list. Block processing stage: The data block is processed by the automatons. Link query phase: link the results of the block processing to get the matching results. This chapter will focus on the implementation of the block processing phase and link query phase.

A. Master Thread Algorithm

Master Thread is the master thread of PPPTransducers, XPath ordered, automaton construction, data partitioning, block processing, and link query threads are started in this thread. Xpath order is executed first, and a set of query tree patterns will be stored in the modes collection; then all query tree patterns in the modes collection will be translated into a set of automata that will be stored in the autos collection; Then, start a data partitioning thread, k block processing thread and a link query thread, these threads are executed in parallel. The data partitioning thread products data block, and each block processing thread apply the data block from the blocks. After the data block is processed, the result will be stored in the midlist (Synchronous hash table, the block number is key, the data block processing results for the value), and then apply for the next block for processing. The link thread links the block processing results in the midList in block order.

```
Master Thread (xml, xpath, k,size)
blocks;
midlist;
rults;
modes ← Order Paths(xpath);
autos ← AutoMation Building(modes);
                    blocks← Data Divided(xml, size)
foreach I int k
    BPQuery (block, autos, rults, midList);
end of foreach
LQAL (midlist, rults);
```

B. Block Processing

In block processing phase, multiple block processing threads are executed in parallel, each thread contains a set of pushdown automaton.

BPQuery (block, autos, rults, midList) algorithm is data block processing, block processing thread get a block of data and use automations process it.

When the start label arrives, each record in the mapping is processed. If the label matches, change the state and complete the stack operation. If the label is the target label, the label is temporarily saved. If the current state contains the AD axis, add a record to the mapping.

```
BPQuery(block, autos, rults, midL ist)
block←blocks. get();
foreach au in  autos
        foreach label in block
                   mapping←au. get Mapping();
                  foreach rec in top mapping
                      if qs==start   then
                           mark (rec);
                           if start tag==label
                                zf.push (qf);
                               qf← qf. next;
                                if label==target   then
                 o.add←label;
             if rec include AD then
             rec1←rec;
             rec1.zf.push (qf);
             mapping. add (rec1);
       end of foreach
       popout (label, mapping, rults);
end of foreach
m_Rult.add (mapping);
end of foreach
midlist. add (m_ Rult);
```

When end label arrive, call the popout function, processing each record in the mapping: 1. if finish stack is not empty, then finish stack take stack operation, complete state take state transfer; 2. If finish stack is empty, then you need to enumerate each complete state that the end label can reach. If the record cannot be matched, delete the record from mappping.

Link query phase, some query results may span two or more data blocks, and the link query phase is responsible for finding cross-block matching results. The link query thread starts from the first entry of the midlist. First, links the first two items, and result link to the next item of the midList until all items was processed.

The LQAL (MidList, rults) algorithm is used to link the processing results of the block processing thread to find the results across multiple blocks. LQAL (MidList, rults) treat block processing results (MidList) as input. First from the MidList to take two elements to link, the link result will link third element, repeat the process. Mapping link use Join algorithm to deal with.

```
LQAL (midlist, rults)
m_ Rult0←midList(0);
m_Rult1;
for (n=1; ;)
  m_Rult1←m_Rult0;
                        m_Rult2←midList(n++);
  for (i=0; i<m_Rult1.size; i++)
     mapping1 ←m_Rult1.get (i);
     mapping2 ←m_Rult2.get (i);
     for (j=0; j<mapping1.size; j++)
       for (k=0; k<mapping2.size; k++)
         rults←Join(mapping1.get(j), mapping2.get(k),
m_Rult0);
         end of for
     end of for
```

V. Experiment

In order to verify the efficiency of the PPPTransducers algorithm, PPPTransducers and QXList [4] were implemented in the Java language, and statistics runtime and memory usage of the two process methods. The experimental environment is Windows Server 2008 operating system, AMD Opteron (tm) Processor 6344 2.59GHz Core48 CPU, 32GB memory, JProfiler test software. The test cases and data sets as shown in Table 2, 3.

TABLE II. Table 2 TEST CASE

XPath	Number
/site/regions/africa/item/description/parlist/listitem	A1
//regions//location	A2
/site/regions/africa//listitem	A3
/site/regions/africa[item/description/parlist/listitem]	A4
/site/regions/africa[//listitem]/item	A5
/site/regions/africa[item[name][payment]]/description	A6
//africa//parlist/text	A7

TABLE III. Table 3 XML DATA SET INFORMATION

	XMark1	XMark2
size (MB)	1165	2333
nodes (million)	33.46	46.84

PPPTransducers data partition thread continues to divide the XML stream data, Table 4 statistics the XMark1 and XMark2 two data sets information which includes the number of blocks, test cases A7 sub query mount.

Use A5 on the XMark2 data set to compare PPPTransducers and QXList, as shown in Figure VI(a). PPPTransducers started four block processing threads (QXList does not support parallel processing), PPPTransducers link query thread are executed in parallel with data partitioning thread and block processing threads, so PPPTransducers are processing faster.

TABLE IV. Table 4 PPP TRANSDUCERS PROCESSING INFORMATION

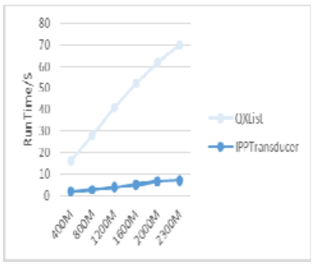
	XMark1	XMark2
block (number)	16405	32641
sub-query (number)	3	3
memory (G)	1.96	2.74
Response time (ms)	161	159

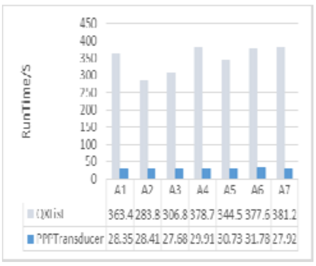
The A1-A7 test case is compared on the XMark1 data set, as shown in Figure VI(b). PPPTransducers processs data in parallel by starting multiple block processing threads. The processing time is much faster than the QXList. The A4-A6 test case contains the predicates. The PPPTransducers will convert the query, resulting the query processing time increase.

PPPTransducers block processing will continue to generate block data, so it will take up part of the memory. Data block processing threads will soon process these data blocks, but the result is not the end result, but need link query processing to release the memory occupied. As shown in Figure VII, the memory used by PPPTransducers increases as the amount of processed data increases, and the QXList due to use the filter operation, so the use of memory will not be affected by the amount of data processing.

VI. CONCLUSION

Parallel query algorithm of XML stream data based on data parallel and push-down transducer(PPPTransducers), use multi-core processor to achieve XML stream data query. The algorithm uses the order tree mode as matching method and apply parallel processing strategy effectively improves the query processing efficiency, reduces the processing cost of the link query, making the algorithm has a very good processing efficiency.





（a） （b）

FIGURE VI. RUN TIME COMPARISON

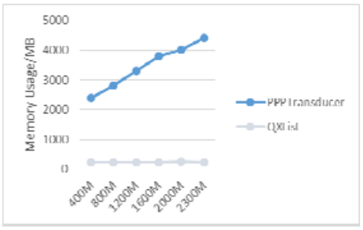


FIGURE VII. MEMORY USAGE CONTRAST

VII. REFERENCES

[1] Bordawekar R, Lim L, Kementsietsidis A, et al. Statistics-based parallelization of XPath queries in shared memory systems[C]. Lausanne, Switzerland: EDBT , 2010:159-170.

[2] Moussalli R, Salloum M, Najjar W, et al. Accelerating XML Query Matching through Custom Stack Generation on FPGAs[C]. Pisa, Italy: Proceedings , 2010:141-155.

[3] Zhang Y, Pan Y, Chiu K. A Parallel XPath Engine Based on Concurrent NFA Execution[C]. Washington, DC: IEEE, 2010:314-321.

[4] HE Zhixue,LIAO Husheng. Query processing method of XML streaming data using list[J]. Journal of Computer Applications, 2016, 36(3): 665-669.

[5] Ogden P, Thomas D, Pietzuch P. Scalable XML query processing using parallel pushdown transducers[J]. Proceedings of the Vldb Endowment, 2013,6(14):1738-1749.

[6] Liu L, Feng J, Li G, et al. Parallel Structural Join Algorithm on Shared-Memory Multi-Core Systems[C]. Washington, DC: IEEE, 2008:70-77.

[7] Zhang Y, Pan Y, Chiu K. Speculative p-DFAs for parallel XML parsing[C].High Performance Computing (HiPC), 2009 International Conference on. IEEE, 2010:388-397.

[8] Alur R, Madhusudan P. Adding Nesting Structure to Words[M].Berlin Heidelberg: Springer, 2006:1-13.