The Research on XML Filtering Model using Lazy DFA

Zhao Heji and Xia Weijian
School of Computer Science and Technology
Shandong University
Jinan, China
E-mail: hejizh@sdu.edu.cn

Zhao Jiasheng
School of Mechanical,Electrical & Information Engineering
Shandong University At Weihai
Weihai,China
E-mail:983298132@qq.com

Abstract—**Parsing the XML document is generally accepted to be a major bottleneck in XML processing,especially when meeting the invalid elements,it will reduce the parsing's efficiency.In order to skip the invalid elements of the XML document,we propose a new XML filtering method based on hash table and stream index.The hash table stores the location information of elements, in order to judge the relationships of elements; the stream index locates the elements accurately, in order to skip the invalid elements quickly.We apply them to lazy DFA and get experimental results. The experimental results show that our method has a good filtering performance,which can improve the efficiency obviously.**

Index Terms—XML;Hash Table;Stream index;Filtering

I. Introduction

Since XML was born in 1998,it has been the standard of data interchange format on internet.A large number of associated applications,such as message notification system and personal personalized information system,need to filter the information.How to improve XML filtering efficiency,has been one of the hot issues in recent years.

In XML filtering,researchers mainly improve the filtering efficiency by reducing its time cost and space cost.For this purpose, in the past few years, a series techniques of XML filtering have been proposed.

XFilter establishes a independent finite state automaton for each path expression,uses the inverted index to the filtration,in the filtration,the automatons run at the same time.YFilter combines all the finite state automatons together corresponding to the path expressions,which forms a single automaton.It shares the prefix of a large number of path expressions,which reduce the number of automaton's states obviously.The DFA corresponds to the path expression,making each transformation from a state to another state.Lazy DFA reduces the number of DFA's states greatly,which improve the XML filtering efficiency obviously.

Generally speaking,the main bottleneck of XML document process is the parsing.In the traditional filtering method,to lazy DFA,the parser need to parse every element of the document,if one element can not be matched,all of its offsprings could not be matched either.The offsprings that can not be matched are invalid.Parsing the invalid offsprings will waste a lot of both time and space.If we can detect the invalid elements and skip them in the filtering,jump to their following elements directly,it will undoubtedly accelerate the filtering rate,improve the efficiency.It is what we will discuss in this paper.

In this paper,we reduce the time cost and space cost in XML filtering by reducing the processing of invalid elements.Our method includes two aspects,the first aspect is to detect the invalid elements in the ancestry – posterity relationships of XML document,which can be realized through hash table,as the hash table stores the location information of the elements,by comparing the location information and the filtering requirements,we can judge which elements are invalid;the second aspect is to skip the invalid elements,which can be realized by stream index,by stream index,we can go to the successor of invalid elements.During the running process of lazy DFA,the hash table and stream index are not independent, but interrelated.

II. Related Work

A. XML Document Tree

An XML document tree corresponds to an XML document,the tree node corresponds to the element of XML document,the tree edge corresponds to the 'element-child element' relationship of XML document.

Given an XML document tree as shown in figure 1.The tree has a root named 'book',the book node has three children nodes - title,author and publisher.

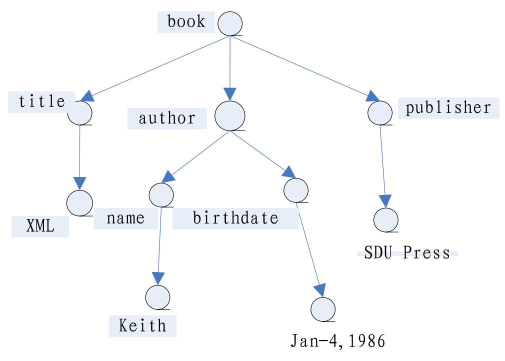

Figure 1 An XML document tree

The corresponding XML document is as follows:

```
<book>
        <title>XML</title>
          <author>
            <name>Keith</name>
            <birthdate>Jan-4,1986</birthdate>
          </author>
<publisher>SDU Press</publisher>
</book>
```

From figure 1 we can see,in the XML document tree,its every node corresponds to a node of the XML document.For example,the node 'book' is the root node of the tree,it's the start element in the document.Also,the tree maintains the level relationships of the document,and the level relationships are more intuitive in the tree.

B. XPath

XPath is a kind of query language,which is uesd to navigate the elements and their attributes of XML document. XPath defines the elements and operators of path expression, including axis,elements and one or more attributes of elements.The axis is used for judging the relationship of each element and its properties,such as the father-son relationship of symbol '/',the ancestry-posterity relationship of the symbol '//', the choice relationship among the paths of the symbol '|'.

XPath uses path expression to select element of the document.The element is selected by path or by step.

The following table 1 lists several most commonly used path expressions.

Table I
The Most Commonly Used Path Expressions And Their Descriptions

Expressions	Descriptions
nodename	Select all children of the node
/	Select from the root
//	Select the match node from current node, regard less of their depth
.	Select the current node
..	Select the father node of current node
@	Select the attributes

In the following table 2,we give out some examples of path expressions and results of the expressions.

Table II
Some Examples Of Path Expressions

Path expressions	Results
book	Select all children nodes of book
/book	Select root node book,if the path start from '/',then the path represents absolute path
book/title	Select all children nodes 'title' of book
//book	Select all children of book,regardless the depth of them
book//title	Select all offsprings nodes 'title' of book,regardless the depth
//@lang	Select all attributes that named 'lang'

According to XPath grammar, any XPath expression can be converted into a regular expression.Based on automaton theory, then there must be a Finite State Machine[6] to match it.

Given XPath expression: Q=/book//press, its equivalent NFA is shown in figure 2:

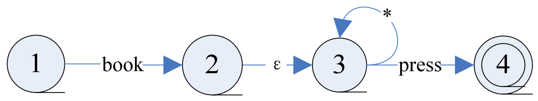

Figure 2 The NFA of expression Q

C. SAX Parser

In XML filtering process, we usually use SAX to parse XML document.

SAX(Simple API for XML) is based on event-driven(different from DOM,which is base on document-driven),its working mechanism likes the tag library.When the tag begins,ends or the errors happen,SAX will call corresponding interfaces to deal with the problems.As SAX is based on event-driven,SAX parse the documents sequentially and hierarchically.It focuses on the continuous processing of the current incidents, as it dose not need to read the document as a whole,it dose not read the whole document into the memory in one time.To SAX,the document's being read process is also the processing process,that is,SAX parses the document while it reading the document,which is different from DOM.

SAX has several interfaces,as shown in figure 3:

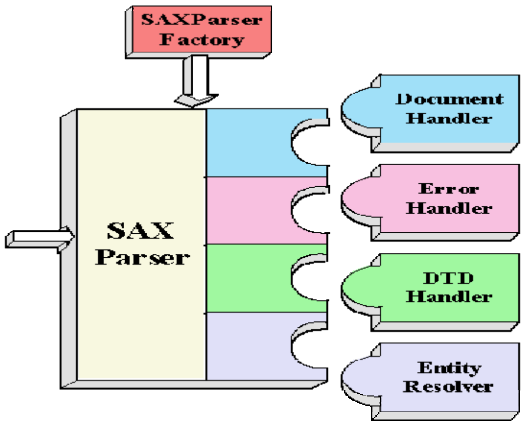

Figure 3 The interfaces of SAX

The most commonly used interface of SAX is ContentHandler. It contains the following five methods: startDocument (), endDocument (), startElement (String uri, String localName, String qName, Attributes atts), endElement (String uri, String localName, String qName) and characters (char[ ] ch, int start, int length).

This interface encapsulates several methods for events processing.When SAX begins to parse XML documents, it will meet several special events, such as document begins and ends, element begins and ends, the character data of the elements.When meeting these events, SAX will call the methods of the ContentHandler interface to response to these events.

D. *The Lazy DFA*

According to XPath grammar and automaton theory,any XPath expression can be converted into a corresponding NFA.The property of NFA is uncertain,as at the same converting conditions,the NFA may reach different states.In the states converting,the NFA reaches a collection of states,but not a single state,which makes the execute efficiency degrades greatly with the increases of XPath expressions.This is more obviously when XML query requisitions are huge.

The DFA is the deterministic form of NFA,they are equivalent.The DFA depends on XPath expression,when XPath expression contains a lot of symbols "//" or wildcard rings "*",the states number of DFA will increase exponentially,which lead to the degrade of efficiency.In order to improve the filtering efficiency of the XML document,we introduce lazy DFA.

Lazy DFA includes two phases,the first phase is the warm-up phase,the second phase is the stable phase.It is constructed during the running process of the NFA.It computes and creates its new state according to the existing states of DFA.It only with the structure of the XML document,but has nothing to do with XPath expression.Therefore,it avoids the exponential growth of lazy DFA with the number increase of XPath expression.

Each state of lazy DFA corresponds to a NFA table,which is to record the existing NFA state.And each state of lazy DFA maintains a states transition hash table,in order to record the trigger conditions of DFA's states transition.The lazy DFA needs to maintain a stack,stores all the intermediate states from initial state to current state.During the running process of the lazy DFA,the stack pushes or pops DFA state all the time.

E. *The Related Issues*

In the process of parsing XML document, if an element can’t match any inquires,then all of its sons or offsprings can't get any matching results too. Especially in the ancestry - posterity relationship '//', it contains a ‘ ε ’ conversion and a wildcard ring ‘*’, which nested in a large depth, when an element does not satisfy the filtering conditions, its children will cause a lot of waste in time and space.

For example, use the XML document shown in figure 1 as the input, use the NFA shown in figure 2 as the filter, the running process of stack to the filtering is shown in table 3.

Table III
The Running Process Of Stack For The NFA

					3		3					
		3		3	3	3	3	3		3		
	2, 3	2, 3	2, 3	2, 3	2, 3	2, 3	2, 3	2, 3	2, 3	2, 3	2, 3	
1	1	1	1	1	1	1	1	1	1	1	1	1

-
<book><title></title><author><name></name><birthdate></birthdate></author><publisher></publisher></book>

From the running process of the stack we can see, based on the XML document and the filtering expression, any offspring of the node 'book' can't match the expression Q = / book/ / press, but in the filtration, we still need to put them into stack to compare with the filtering expression. It will cause a lot of waste both in time and space, this kind of waste is more obvious when XML document is large or there are a lot of ancestry - posterity relationships in the filtering expression . If we can judge certain elements in the document are invalid in advance, that is,they can't match the filtering expression, then we put them out of the running process,this will undoubtedly improve the filtering efficiency,therefore,we introduce stream index and hash table. The hash table is used to judge any two elements of the document that whether they exist the ancestry - posterity relationship,the stream index is used to locate the position of elements accurately,in order to skip the invalid elements quickly. In the previous example ,by inquiring hash table in advance, we can judge that all offspring nodes of node ‘book’ can’t match node ‘press’, then we can skip the node ‘book’ and its offsprings while filtering through stream index,so in this way we can get a better efficiency in time and space.

III. The introduction of Stream Index and its generating algorithm

A. *The Introduction of Stream Index*

Parsing and tokenizing the XML document is generally accepted to be a major bottleneck in XML processing. An obvious solution is to represent an XML document in binary, as a string of binary tokens. In an XML message system,the messages are now binary representations, rather than real XML tags, or they are converted into binary when they enter the system.Some commercial implementations adopt this approach in order to improve performance.The disadvantage is that all servers in network must understand that binary format. This defeats the purpose of XML standard,which is supposed to address precisely but lack interoperability,that is associated with a binary format.

Stream IndeX (SIX) [1,4,7]is an additional structure of XML document. It consists of a binary group (start, finish), both 'start' and 'finish' are integer, respectively marks the elements’ start position and end position.

Apply stream index to XML filtering,we use stream index to locate the positions of elements,by the start value and finish value we can get the elements' start positions and finish positions accurately.We locate the elements' position by start value, skip the invalid elements by finish value,in this way we skip the invalid elements and their offsprings,improve the efficiency of filtration.

In filtering, if an element 's' meets the state transition conditions, it drive the NFA to state transition, according to the start value of stream index on element 's', jump to element whose start value equals s.start+1; if the element 's' dose not meet the state transition conditions,then according to s's finish value of its stream index,jump to the element whose finish value equals s.finish+1.Through this method,we skip the invalid elements directly.

B. The Generating Algorithm of Stream Index

Stream index can be generated by SAX ,that we use SAX to parse XML document in a traverse. In the process of creating stream index,it needs to maintain a global variable 'count' and a stack,the 'count' is a counter,and the stack determines analytic sequence,the generating process is as follows:

1) Call startDocument() method to parse the document;

2) Analyse the element's startElement() event, add 1 to the value of count, order the element's start value of stream index equal count, put the start value into the stack;

3) Analyse the element's endElement() event, pop the top element out of stack, order the element's finish value of stream index equal count;

4) Analyse the endDocument() event, which means the end of XML document, all the elements' stream index have been created.

By the algorithm above,we can generate stream index for the XML document.

IV. The Introduction of Hash Table and Its Generating Algorithm

A. The Introduction of Hash Table and Its Filtering Principle

In the stream index section above,we locate the elements' positions accurately,but we have not given out the level relationships among them.In XML document filtering,if XPath expression contains symbol "//",the node that followed the symbol can match any element of the document that corresponding to the node after symbol "//" ,no matter whatever depth it is.If there are too many "//" symbols in XPath or there are too many nested relationships in the document,it will affect the filtering efficiency greatly.Therefore,we need to introduce a mechanism to judge the level relationships of any two elements in the document.Therefore,we introduce hash table,we give an example as follows.

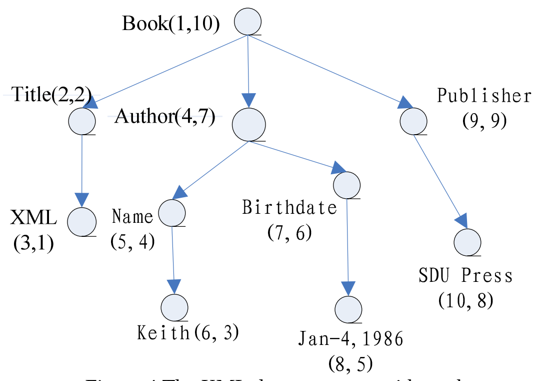

Figure 4 The XML document tree with marks

Traverse the XML document tree of figure 1 in depth first traverse,set two separate integer variable attached to XML document tree-start and finish,the initial value both are 0,used to mark the start position and finish position of the elements.The rules are as follows:

Traverse the XML document in depth first traverse,when push the element into stack,add 1 to start;when pop the element out of stack,add 1 to finish.

After that,we can get the result as shown in figure 4.From the figure we can see,by comparing the start value and finish value of any elements,we can get the level relationships between any two nodes,namely,it's the ancestry - posterity relationship.The rules are as follows:For any two nodes a and b in an XML document, a is b's ancestral node, if and only if a.start<b.start and a.finish>b.finish; similarly, b is the offspring of node a, if and only if b.start>a.start and b.finish<a.finish.

For example,the node 'name' is the grandson of node 'book',therefore,they satisfy the following relationships:name.start(5)>book.start(1)&name.finish(4)<book.finish(10).At the same time,the node 'author' and the node 'publisher' don't have the ancestry-posterity relationship,so their start value and finish value do not meet the rules above.

As we can see,there are many common characteristics between the location marks attached to the nodes and the stream index,they both mark the element's start position and finish position,the difference is:the start value and finish value share the same counter in stream index while they are independent in location marks.In order to traverse the XML document to generate the stream index and location marks in one time by SAX,we need to combine them together,therefore,we use the stream index as the location marks.So,in order to judge the level relationships of the elements,we modify the level-relationships rules referred above as follows:for any two nodes a and b in an XML document, a is b's ancestral node, if and only if a.start<b.start and a.finish ⩾ b.finish;similarly,b is the offspring of node a, if and only if b.start>a.start and b.finish⩽a.finish.

Next,we discuss the storage mechanism of location marks,according to the relationships of element mark and location marks,we use hash table to store them.

On hash table's mapping structure,the key value(index value,in hash table named h_key) of hash table uses the start value of stream index, that's because the start value is unique and increases gradually, the search work can be finished in linear time; the hash table address stores the element's tag and finish value of stream index(in hash table named h_addr).

Through the method above, after marking the elements by location marks,whatever degree of the elements' depth and circulation are in the document, with an input XML document containing n elements, whether the elements meet the filterring conditions can be judged in linear time (O (n)) .According to the filtering expression,we start to parse the input XML document,when meeting the filtering symbol '/' of the filtering expression, put the input elements into the stack directly to compare with the filtering expression;when meeting the filtering symbol '//' in the filtering expression, do not put the input elements into the stack, but to look up the hash table,according to the rules referred above,compare the two elements' 'h_key' values and 'h_addr' values ,which corresponding to the two elements that before and after the filtering symbol '//' in the filtering expression.In this way,we can judge whether the two elements meet the ancestry - posterity relationships. If meet, then put the input element into the stack to continue the parsing; if not meet, that means the input element is invalid, then skip it and its offsprings through the stream index.

For example, given the XPath expression Q = a //b, in the filtering, we don't need to put the element 'b' of the document into stack, but through the h_key value to look up the hash table to find a's and b's 'h_key' value and 'h_addr' value.By comparing their 'h_key' value and 'h_addr' value,we can judge whether a and b exist the ancestry - posterity relations.If exist, put b into stack to continue the parsing;if not, then skip element b by stream index,we avoid to put b and b's offsprings of the document into stack,which is in order to compare them with the filtering expression .

B. The Generating Algorithm of Hash Table

Since hash table and stream index both need to be created by the SAX,and their values are the same,so they can be generated in the same time while using SAX to scan the XML document in a traverse.

The generating algorithm of hash table and stream is as follows:

1) Call startDocument () method to start the parsing of XML document, and set a global variable count;

2) When analysing the startElement () event, add 1 to the count value, create the element's stream index and hash table, order this element's start value of stream index equal count, the hash table's h_key value equal start, add this element's tag to the hash table's address,then put the start value into stack;

3) When analysing the endElement () event, pop the top element out of stack,order this element's finish value equal count,then according to the corresponding h_key value, add the finish value to the address of hash table,in this way,we set the element's h_addr value equals the finish value.

4) When analysing the endDocument() event,it means we reach the end of the XML document,all the stream index and hash table for the whole XML document have been created.

For example, given the following XML document:

<a><b><c><d><e>text</e></d></c><f>text</f></b></a>

Call the algorithm above,we can create the hash table for each element of the document, the result is shown in table 4.

Table IV
THE HASH TABLE CORRESPONDING TO THE DOCUMENT

1	a	6
2	b	6
3	c	5
4	d	5
5	e	5
6	f	6

V. THE APPLICATION OF HASH TABLE AND STREAM INDEX ON LAZY DFA

A. Apply the Hash Table and SIX to Lazy DFA

We have referred above,according to XPath grammar and automaton theory,any XPath expression can be converted a corresponding NFA,but the NFA is uncertain,at the same converting conditions,it may reach different states.It requires us to generate the determined form of NFA,that is the DFA.In this process,in order to save space,improve the efficiency of filtration,we not only need to omit some invalid states of NFA that could not be reached,we also need to avoid the DFA's states number grows exponentially, which with the increase of XPath's amount.So,we introduce lazy DFA.The lazy DFA's working mechanism we have discussed in the 'Related Work' section,in this part,we only discuss the lazy DFA's working mechanism that after introducing the hash table and stream index.

Next,we mainly talk about how to judge which elements in the XML document are invalid elements and how to skip them in the filtering.

Apply the hash table and stream index to lazy DFA, traverse the input document for a pretreatment,establish the stream index and hash table for the elements.The hash table stores the location information about the elements,in order to judge any two elements whether exists the ancestry - posterity relations,the stream index is used to locate the elements' positions.In the generative process of lazy DFA, according to NFA that transformed from the filtering expression, when meet the ancestry - posterity relations, go to look up the hash table first, in order to look for the two elements in the input document which corresponding to the two elements that before and after the symbol '//'.By comparing their 'h_key' value and 'h_addr' value,we can judge whether they have the ancestry - posterity relationship,that is to judge the element that corresponding to the element after the symbol '//' whether is valid.If this element is invalid,then skip it by stream index,so in the generative process of lazy DFA,it can reduce the useless states obviously.

B. Examples

For example, given the XML document :<a> <g> <d> text </d> </g><b name='java'> <c name = 'C++'> <f> <h>content</h><i><j> text </j></i><e> </e> </f> </c> </b> </a>.Parse the given document with SAX, and call the algorithm that given in 3.2, we can get the following results: SIX(a) =(1,10),SIX(g) = (2,3),SIX (d) = (3,3),SIX(b)=(4,10), SIX(c)=(5,10),SIX(f)=(6,10),SIX(h)=(7,7),SIX(i)=(8,9),SIX(j)=(9,9),SIX(e)=(10,10),and the hash table that corresponding to the document is shown in table 5.

Given the Xpath expression:Q1==/a/b/*/f//e,the corresponding NFA is shown in figure 5:

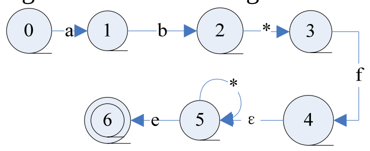

Figure 5 The NFA that corresponding to expression Q1

Table V
THE HASH TABLE THAT CORRESPONDING TO THE XML DOCUMENT

1	a	10
2	g	3
3	d	3
4	b	10
5	c	10
6	f	10
7	h	7
8	i	9
9	j	9
10	e	10

The process of using the rules above to construct the corresponding lazy DFA is as follows:

1) The DFA's initial state is state 0, get the first input element of the XML document,the filtering conditions of the filtering expression is '/a', match successfully, then drive the DFA to create a new state,marked by number 1, set the state 1 to be the current active state of DFA;

2) Get the next input element g, the filtering conditions of filtering expression is '/b', match fails, namely that g is an invalid element, according to the rules above, then jump to the element whose stream index's start value equal g.finish+1,so get the next input element 'b';

3) Now the filtering conditions of filtering expression is '/b', match successfully, so from state 1 of DFA to create a new state,marked it as state 2,then set state 2 as DFA's current active state;

4) Now get the next input element 'c',the filtering conditions of filtering expression is '/*',this is a wildcard,it can match any element,so from the state 2 of DFA to create a new state,marked it as state 3,set the state 3 as the DFA's current active state;

5) Now get the next input element 'f',the filtering conditions of filtering expression is '/f',match successfully,then drive DFA to create a new state from state 3,marked as state 4,then set state 4 as DFA's current active state;

6) Now the filtering conditions of the expression is '//e',this is a filtering condition that matches the ancestry - posterity relations.According to the rules that referred before,we turn to look up the hash table,to look for the two elements in the input XML document, we get f.h_key=6,f.h_addr=10,e.h_key=10,e.h_addr=10,meet the following relationship expression e.h_key>f.h_key and e.h_addr≤f.h_addr,namely that 'e' is the offspring of 'f',so match the filtering condition successfully.Then from state 4 of the DFA to create a new state,marked it as state 5,the element 'e' is its drive condition.We have reached the end of the document,the parsing finish.

Form the example above we can see,in the processing procedure of filtering condition '//e',which means the ancestry - posterity relationships,through looking up the hash table,we get the results in a linear time O(n),therefore, we avoid to put all of element 'f's offsprings of the document into the stack,which is to compare with '//e' and skip the invalid elements directly,so we reduce the time cost and improve the filtring efficiency.

VI. THE EXPERIMENT

The processing of predicates in the XPath expresion influences the XML filtering greatly.The predicates may have many test conditions,these conditions could be connected according to the needs.At present,many query processing systems do not support the predicates enough or have low efficiency.So,in our experiments,we have both tested the expressions that with predicates and the expressions without predicates,in order to make our experimental results more comprehensive.

In order to test the performance of lazy DFA that based on hash table and stream index,we have done a lot of experiments.In this part,we give out the experimental results and our analysis.

A. The Experimental Environments

The hardware environments of the experiment:the operation system is Windows XP,the CPU is Intel® Core ™ Duo E6550,2.33GHZ,the memory is DDRII 667HZ,2GB

The software environment of the experiment: the IDE is MyEclipse 8.0,the programming language is JAVA.

We realized to establish the hash table and stream for the elements of input XML document, tested the running time of the traditional lazy DFA and the lazy DFA based on hash table and stream index,tested the expressions that with predicates and the expressions without predicates,recorded their running time and the experimental results.

B. The Data Generation

Our experiment used XMark to generate the experimental data. We generated the experimental data by adjusting the parameters for the document of size 0M、40M、60M、80M and 100M, and the depth of each document was no more than 10. We chose the same XPath statements Q=/site//asia/item id="item50"/*/ date to test these documents.

C. *The Experimental Results*

In our experiments, we compared the running time of the lazy DFA that based on hash table and stream index and the running time of the original lazy DFA.

The experimental results show that the time costs of original lazy DFA's query processing grow with the increase of the XML documents,while the time costs of the lazy DFA that based on hash table and stream index grow much slower.For XML documents that has large depth,the time costs of lazy DFA based on hash table and stream index has obvious advantage to original lazy DFA.The query time costs also have the relationship with the length of XPath,if the length of XPath is longer,the query time is longer.The query time of XPath that with predicates is relatively longer,but it also has the relationship with the result set that selected by the query expressions.

The experimental results are as shown in figure 6 and figure 7. As we can see from the figures, with the growth of input XML document's size, both of the running time of two methods are increasing, but the method that we proposed has a much slower growth.Especially in the filtering of XPath expression without predicates, our method is more effective, the running time is far less than traditional lazy DFA's running time.

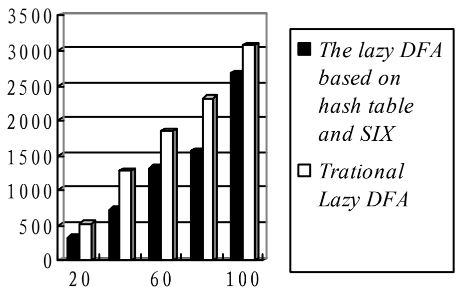

Figure 6 The experimental results of the document without predicates

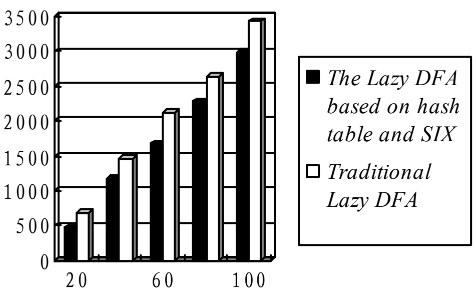

Figure 7 The experimental results of the document with predicates

D. *Discussion*

Our experiments clearly demonstrate that the DFA is effective in XML filtration,as SAX parses the document sequentially and hierarchically,not reads the whole document into the memory in one time,it can maintains a continuous throughput.

The experiments also show that in most cases the lazy DFA can avoid the states increase exponentially.There are many issues affect the efficiency of lazy DFA to process the XML data flow,including the stream index cost,the handing of predicates cost,the construction of lazy DFA cost and so on.

Our experiments indicate our methods on XML document processing with the optimized lazy DFA approach performs significantly better (in terms of both the time/space efficiency and scalability)than the traditional lazy DFA approach.

VII. Conclusion

How to filter the XML data flow in high efficiency,is one of the hot issues in recent years.Use the automaton on XPath modeling,improve the query efficiency and optimize the filtering model are important research directions.

This paper describes the background, the challenges, existing solutions,main idea and key findings of the paper at the beginning,illustrates the relevant theoretical knowledge on XML,XPath,SAX parser and lazy DFA,points out that in the XML document filtering,how to judge and skip the invalid element is the emphasis of this pater.In addition,this paper introduces the stream index and hash table,illustrates their use mechanisms.Then,this paper gives out an example on how to use the stream index and hash table on lazy DFA.At last,this paper gives out the experimental results on XML filtering using traditional lazy DFA and lazy DFA that based on stream index and hash table,discusses the XML document with predicates and the XML document without predicates.So we can see that the lazy DFA that based on stream index and hash table can be well used in XPath expression with a lot of predicates,it can greatly reduce the states numbers during the construction process,therefore improve the efficiency.

This paper uses SAX parser to scan the input XML document,establishes the hash table and stream index for the elements of the input document.The hash table can judge whether the elements meet the filtering condition quickly,especially for the ancestry - posterity relationship.If the elements are invalid,then skip them through the stream index, which greatly improve the filtering efficiency.

Acknowledgment

This research is supported in part by the National Natural Science Foundation of China (NSFC) under grant 60970003.

References

[1] Zhang Lili, Zhao Heji, Xia Weijian. Lazy DFA Filter Based on Stream Index for XML Data Streams[C]. In Proc ICEIT, Chongqing, China, Sep 2010: 163-167

[2] Weiwei Sun, Yongrui Qin, Ping Yu, Zhuoyao Zhang, Zhenying He. HFilter: Hybrid Finite Automaton Based Stream Filtering for Deep and Recursive XML Data[C]. In Proc. of 19th International Conference on DEXA, Turin, Italy, Sep 2008: 566–580.

[3] Shaolei Feng and Giridhar Kumaran. XML Data Stream

Processing: Extensions to Yfilter. 2007: (11), 380-402.
[4] T. Green, A. Gupta, G. Miklau, et al. Processing XML streams with deterministic automata and stream index [J]. ACM Trans. on Database Systems (TODS), 2004:29(4) 752-788.
[5] D. Chen, R.Wong. Optimizing the lazy DFA approach for XML stream processing [C]. The Fifteenth Australasian Database Conference (ADC), Dunedin, New Zealand, 2004:131-140
[6] T. Green, G. Miklau, M. Onizuka, et al. “Processing XML Streams with Deterministic Automata, ” Siena, Italy: Submitted to The 9th International Conference on Database Theory, 2003:173-189
[7] T. Green, A. Gupta, G. Miklau, et al. “Processing XML streams with deterministic automata and stream index,” ACM Trans. on Database Systems (TODS), 2004, 29(4) 752-788
[8] Rajeev Motwani, Jennifer Widom, Arvind Arasu, et a1.Query Processing, Resource Management, and Approximation in a Data Stream Management System, CIDR Conf．Asilomar, 2003：245—256
[9] Y.Diao, P.Fischer, M.J.Franklin et al.YFilter:Efficient and scalable filtering of XML documents．In Proc. of ICDE 2002:341-342
[10] XML Path Language(XPath)2.0.W3C Working Draft 2003. www.w3c.org/TR/xpath20/49
[11] Peter Fankhansar．XQuery Formal Semantics State and Challenges．ACM SIGMOD Record, Volume 30, Issue 3, 2001
[12] Mehmet Altine, Michael J.Franklin.Efficient Filtering of XML Documents for Selective Dissemination of Information.In Proceedings ofthe 26th VLDB Conference, 2000
[13] Chan, C.Y, Felber, P, Garofalakis, M.N., Rastogi, R.Efficient filtering of XML documents with XPath expressions.In Proceedings of International Conference On Data Engineering (ICDE), 2002
[14] V.Josifovski, M.Fontoura, A.Barta.Enabling relational engines to query XML streams.IBM Intemal publication．
[15] Charles Barton, Philippe Charles, Marcus Fontoura, vanja Josifovski. An Algorithm for Streaming XPath Processing with Forward and Backward Axes．In Proceedings of International Conference on Data Enginerring(ICDE),2003．