High performance complex event detection method based on macro forest transducer

Yitian Liu[1,a,*], **Husheng Liao**[1,b] , **Hang Su**[1,c], **Hongyu Gao**[1,d]

[1]Faculty of Information Technology, Beijing University of Technology, No.100 Ping Le Yuan,ChaoYang, Beijing, China

[a] camlot25@gmail.com, [b] liaohs@bjut.edu.cn, [c] suhang@bjut.edu.cn, [d] hygao@bjut.edu.cn

*corresponding author

Keywords: Complex Event Detection, Macro Forest Transducers, Regular Tree Pattern, Extensible Markup Language(XML), Stream Query.

Abstract. Complex event detection technology aims at extracting valuable information from continuously incoming massive stream data quickly and accurately, which is one of the key components of complex event processing platform. XML is usually regarded as one of the data models of complex event processing platform. Although there are many efficient filtering algorithms for XML stream data, they are generally unable to support complex event queries that contain timing relationships. So we used the regular tree pattern to describe the complex event pattern, generated the macro forest transducers and the automatic for regular part matching based on the regular tree pattern grammar and proposed an efficient complex event detection method. The experimental results prove that the method have strong capability on complex event detection.

1. Introduction

In the era of big data, many applications based on Web generate plethora of detail information as event stream. Stream data[1] often needs to be processed in real-time due to its features like unboundedness in volume, randomness in sequence and requirement of processing in one-pass. XML has strong descriptive power since it is self-descriptive and semi-structured and is one of the main standard format for describing and transmitting Web data.

The applications use XML stream data as data model, such as sensor network, meteorological real-time analysis system and Internet security monitoring system, usually receive massive event flow data on low single event value density. The extraction of valuable information in the event flow becomes a challenge. As a result, complex event processing platforms[2] are derived. It can obtain valuable complex events from the stream data and make an immediate response by aggregating, filtering, matching and correlating simple events. Among them, the process of filtering and matching combination of simple events are called complex event detection. Complex event detection often detects complex events in streams by pattern matching. Current researches on complex event detection are mainly focused on two aspects[3], first, the richer complexity of event description language, and the second, more efficient event flow data pattern matching method.

In 2013, Barzan Mozafari et al. proposed a complex event processing language called XSeq[4]. They implemented complex event detection based on VPA (Visual Pushdown Automata) by adding the Kleene closure and sequence operators to XPath. In 2016, Yao Lu proposed a simple format for describe complex events named Regular Tree Pattern[5], which was extended by the XML tree pattern and it can describe the timing relationships between simple events by adding regular expressions on the tree pattern.

The macro forest transducers[6] is an extension to the macro tree transducers[7], which is an XML stream matching model. In 2014, Hakuta S. et al. proposed a method of translating MinXQuery [8] into the macro forest transducers, studied the XML stream processing technology based on macro forest transducers. However, there is no formal method for constructing a macro forest transducers. In 2015, Feng Xuezhi et al.[9] translated XPath into macro forest transducers, and gave the method of creating macro forest transducers template from query step. Nowadays the stream data matching methods based on macro forest transducers have good performance. Moreover, the macro forest transducers can expand the function by modifying and increasing the template.

This paper studied a complex event detection method based on macro forest transducers and described the complex events by the regular tree pattern to meet the requirements of complex event detection. The main contributions of this paper are as follows:

1. A complex event detection method based on macro tree transducer named CEDMFT was proposed by combining the macro forest transducers technology for tree pattern query and the automaton technology for stream data matching.

2. Based on the combination of macro forest transducers and automaton for regular expression matching on XML stream, an efficient algorithm for regular tree pattern matching was proposed.

The sections are organized as follows: Section 1 introduces the background knowledge related to this paper. Section 2 discuss the design ideas and implementation of CEDMFT. Section 4 gives the experimental results for CEDMFT. Section 5 summarizes the research contents and points out the further research direction.

2. Preliminaries

2.1. XML Forest

An XML document can be described as a hierarchical ordered tree. For XML stream data, the XML element in it can be seen as an XML tree, which rooted at the current XML tag, so the XML stream data can be viewed as a sequence of XML subtree. We define the XML subtrees as an XML forest.

2.2. Macro Tree Transducers

A macro forest transducers is defined as a five-tuple $M = (Q, \Sigma, \theta, q_0, R)$, where Q is the set of states, Σ and θ are input and output letters, q_0 is the initial state and $q_0 \in Q$, R is the set of state transition rules, as follows:

$$q(\sigma(x_1)x_2, y_1, \ldots, y_n) \rightarrow f$$
$$q(\%t(x_1)x_2, y_1, \ldots, y_n) \rightarrow f$$
$$q(\varepsilon, y_1, \ldots, y_n) \rightarrow f$$
$$f ::= q(x_k, f, \ldots, f) \mid y_l \mid f\,f \mid \varepsilon$$

where the first parameter of q represents the input forest, $q \in Q$, $\sigma \in \Sigma$, $\%t \in \Sigma$, and σ represents the current state q matches the input tag, $\%t$ represents the current state q does not match the input tag, ε represents the input of the current state q is empty. x_1 and x_2 bind the current node with the children of the current node, and the rest of the stream.

2.3. Regular Tree Pattern Matching

The process of complex event detection is to find out the combination of simple events from the event stream. The regular tree pattern is a tree structure with strong ability to describe the combination of simple events. It adds regular operators such as joint, or, Kleene closure on normal tree pattern, which can describe the timing relationships between events.

Table 1 Grammar of regular tree pattern.

Regular Tree Pattern Grammar
texp ::= rexp | node ({pred} | { "["pred"]" } | "("rexp")")
node ::= axis (tag | tag "→" var)
axis ::= "/" | "//"
var ::= String
pred ::= texp | node cmp const
cmp ::= "=" | ">" | "⩾" | "<" | "⩽"
const ::= String | Integer
rexp ::= texp | "("rexp")" | rexp"*" | rexp":"rexp | rexp"|"rexp

Table 1 shows the regular tree pattern grammar, where the ‘texp’ represents the regular tree pattern and its sub-pattern, node represents the tree pattern node, and the ‘rexp’ is the regular expressions representation of the order of the subtrees.

Regular tree pattern matching is the process of finding the tree structure fragment that matches the tree structure and the contents on the nodes at the same time. This paper is mainly for processing the XML stream data, the tags in the stream carry not only content information, but also structural information such as parent-child relationships, brotherhood and ancestral relationships.

3. CEDMFT

3.1. Idea

The macro forest transducer is an effective matching model for XML stream data, but the existing macro forest transducers can only handle XPath queries. XPath queries can’t describe the timing constraints of complex events and returning one or more matching sub-results. For the above two aspects, this paper has carried out the following extensions to the macro forest transducers:

1) The basic framework of the macro forest transducers is to decompose a query into several subqueries, each subquery corresponding to a query function. However, it is difficult to use multiple query functions to implement the stream data matching of the regular expression due to the stream data cannot be backtracked. Therefore, this paper used the automaton technology to handle the regular matching part and transfer the whole regular expression into one separate query function.

2) Unlike traditional automata techniques, each regular node in the regular expression can be a root of a tree pattern and also the regular matching will act on the match of the XML forest, with the current leftmost longest subsequence as a matching result. Thus, for each regular expression, we constructed a special automaton named SiblingFA and constructed a separate macro forest transducers for each tree pattern on regular expression. Afterwards we used the matching result of the internal macro forest transducers as transfer condition of the external SiblingFA. The combination of two kinds of automaton compose the overall framework.

3) For the demand of multi-return nodes in the regular tree pattern, the multi-return data model was established, and the returned nodes were organized into tree structures. In the tree pattern matching process, the lower return node is added to the parent node descendants through the nesting relation of the query function. In the regular expression matching process, the return nodes are organized into a number of matching brother node sequences, which are merged into the data structure of the parent node through the query function.

3.2. Macro Tree Transducers Query Decomposition Template

Table 3 shows the regular tree pattern decomposition rules, T1 are the tree pattern translation rules. we only give the main rules. T1-1 and T1-3 give the case where the node is not a return node, and the similar others are not listed here. In accordance with these rules, each regular tree pattern query will be translated into a corresponding state function q, and 'where' clause after the query decomposition process. In Rule T1-5, when the current tag is matched, the query with the predicate is decomposed into a query q' for the descendant predicates and $q_{1\text{-}5}(x_2)$ means do the same operation for the brother elements. The function setMark used in the translation template indicates whether the current state recorded in the upper node has been matched and when the argument is true, the node has been found. The function addNode represents the generation of a return node that contains current tag and add it into the state buffer, and also it will do setMark at the same time. T2 is the decomposition rule of the predicate. According to whether it is a return node, the current matching node or true node as a sign to meet the predicate. T3 is translation rules for the regular expression. When the tree with the current tag is satisfied with the matching tree pattern, it continues to match its brother stream, restarts the automaton and continues to match its brother flow when it does not satisfy all the tree patterns in the current state set. When the current input is empty, the result of SiblingFA is reduced.

3.3. Event Stream Matching Algorithm

We use a stack named execute stack(ES), which contains current states for stream $x_1(CS_1)$, current states for stream $x_2(CS_2)$ and reduce stats(RS). Each time a label come from the XML stream, we divided it into two cases. For the arrival of the start tag and the arrival of the end tag. For the matching of start tag, we use the decomposition rules of macro forest transducers, which contains the add node, set mark operation and do state transform by partition the input XML stream, to execute the matching from q_0. When meet the end tag, if it's only one layer left in the ES, we should check if there is a satisfied result in q_0, else we should check if the states in RS have sub-results matched condition.

Table 2 Regular tree pattern expression decomposition rules.

T1 : Texp ::= Name		
T1〖/tag〗 = q_{1-1}		(T1-1)
where q_{1-1}(tag(x_1)x_2)	= setMark(true)	
q_{1-1}(%t(x_1)x_2)	= q_{1-1}(x_2)	
q_{1-1}(ε)	= ε	
T1〖/tag->var〗 = q_{1-2}		(T1-2)
where q_{1-2}(tag(x_1)x_2)	= addNode(tag, ε) q_{1-2}(x_2)	
q_{1-2}(%t(x_1)x_2)	= q_{1-2}(x_2)	
q_{1-2}(ε)	= ε	
T1〖//tag〗 = q_{1-3}		(T1-3)
where q_{1-3}(tag(x_1)x_2)	= setMark(true)	
q_{1-3}(%t(x_1)x_2)	= q_{1-3}(x_1) or q_{1-3}(x_2)	
q_{1-3}(ε)	= ε	
T1〖//tag->var〗 = q_{1-4}		(T1-4)
where q_{1-4}(tag(x_1)x_2)	= addNode(tag, ε))q_{1-4}(x_1)q_{1-4}(x_2)	
q_{1-4}(%t(x_1)x_2)	= q_{1-4}(x_1)q_{1-4}(x_2)	
q_{1-4}(ε)	= ε	
T1〖/tag->var preds〗 = q_{1-5}		(T1-5)
where q_{1-5}(tag(x_1)x_2)	= addNode(tag, q'(x_1)) q_{1-5}(x_2)	
q_{1-5}(%t(x_1)x_2)	= q_{1-5}(x_2)	
q_{1-5}(ε)	= ε	
q' = T2〖preds〗		
T1〖//tag->var preds〗 = q_{1-6}		(T1-6)
where q_{1-6}(tag(x_1)x_2)	=addNode(tag,q'(x_1))q_{1-6}(x_1)q_{1-6}(x_2)	
q_{1-6}(%t(x_1)x_2)	= q_{1-6}(x_1)q_{1-6}(x_2)	
q_{1-6}(ε)	= ε	
q' = T2〖preds〗		
T1〖/tag->var rexp〗 = q_{1-7}		(T1-7)
where q_{1-7}(tag(x_1)x_2)	= addNode(tag, q'(x_1)) q_{1-7}(x_2)	
q_{1-7}(%t(x_1)x_2)	= q_{1-7}(x_2)	
q_{1-7}(ε)	= ε	
q' = T3〖rexp〗		
T1〖//tag->var rexp〗 = q_{1-8}		(T1-8)
where q_{1-8}(tag(x_1)x_2)	= addNode(tag,q'(x_1))q_{1-8}(x_1) q_{1-8}(x_2)	
q_{1-8}(%t(x_1)x_2)	= q_{1-8}(x_1) q_{1-8}(x_2)	
q_{1-8}(ε)	= ε	
q' = T3〖rexp〗		
T2 : Preds ::= Name		
T2〖/tag cmp const〗 = q_{2-1}		(T2-1)
where q_{2-1}(tag(x_1)x_2)	= if(cmp(value(tag), value(const))) addNode(true) else q_{2-1}(x_2)	
q_{2-1}(%t(x_1)x_2)	= q_{2-1}(x_2)	
q_{2-1}(ε)	= ε	
T2〖/tag->var cmp const〗 = q_{2-2}		(T2-2)
where q_{2-2}(tag(x_1)x_2)	= if(cmp(value(tag),value(const))) addNode(tag, null) q_{2-2}(x_2)	
q_{2-2}(%t(x_1)x_2)	= q_{2-2}(x_2)	
q_{2-2}(ε)	= ε	
T2〖[texp]〗= T1〖texp〗		(T2-3)
T2〖pred preds〗 = q_{2-4}		(T2-4)
where q_{2-4}(x_0)	= if(q'(x_0) != ε and q"(x_0) != ε) q'(x_0) q''(x_0) else ε	
q'	= T2〖pred〗	
q''	= T2〖preds〗	
T3 : Rexp ::= Name		
T3〖rexp〗 = SiblingFA		(T3-1)
where SiblingFA(t_1x_2)	= SiblingFA(x_2)	
SiblingFA(%t_1x_2)	= restartFA()FA(x_2)	
SiblingFA(ε)	= reduceResult()	
q' = T3〖texp〗		
T3〖texp〗= T1〖texp〗		(T3-2)
T3〖(rexp)〗= T3〖rexp〗		(T3-3)

Table 3 Algorithm for CEDMFT.

```
Algorithm ESMA
Input: XML stream $x_0$, initial state of macro forest transducers $q_0$
Output: Query Result
1.  ES.push({$q_0$}, ε, ε))  // the content in the parenthesis are ($CS_1$, $CS_2$, RS)
2.  while ES not empty do
3.    if current input element is start tag then  //  do matching
4.      // tag($x_1$)$x_2$ is the decomposition of $x_0$
5.      newCS$_{1,2}$ = ε
6.      FOR $q_m \in CS_1$
7.        if $q_m$ is a SiblingFA then
8.          newCS$_1$.add(SiblingFA($x_1$))
9.        else
10.         newCS$_1$.add($q_m$($x_1$)) // use corresponding decomposition rule do state transform
11.         if $q_m$ is matched tag and it is not indecomposable then
12.           ES.top.RS.add($q_m$)  // add $q_m$ to the topest layer of the EStack
13.     FOR $q_n \in CS_2$
14.       if $q_n$ is a SiblingFA then
15.         SiblingFA($x_2$)
16.       else
17.         newCS$_2$.add($q_n$($x_2$)) // use corresponding decomposition rule do state transform
18.     ES.push(newCS1, newCS2, ε)
19.   else  // do reducing
20.     if ES has only one layer
21.       return the result in $q_0$
22.     ES.pop()
23.     FOR $q_r \in$ ES.top.RS
24.       reduce($q_r$)  // ensure if the node is satisfied by checking the marks set by the function setMark
```

4. Experiment

4.1. Experiment Setup

In these experiments, we evaluated our algorithms under the condition of Intel Core i7 2.5GHz CPU, 16GB memory, OS X Yosemite and Java 1.8 runtime environment.

Table 4 Three XML baseline dataset.

Dataset	DBLP	Treebank	XMark
Size(MB)	133.9	86.1	116.5

Table 5 XMark benchmark dataset in different size.

XMark Generate Factor	0.1	0.5	1	5	10
Size(MB)	11.9	59	118.6	595.7	1218.6

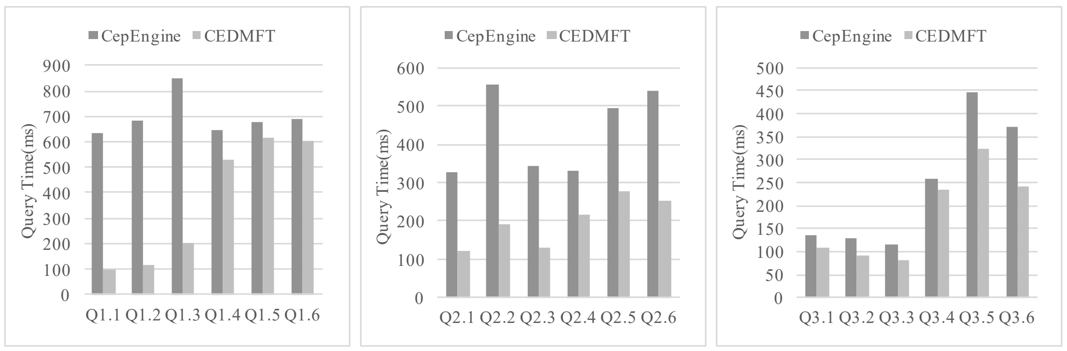
Table 4 and Table 5 show the datasets used in the experiment. Table 4 for the three XML reference dataset, respectively, two real datasets DBLP and Treebank, and a parameter based on the manually generated dataset XMark. Table 5 is the XMark dataset generated manually according to the different parameters.

Table 6 shows the query cases used in the experiment, where Q1 are the queries on the dataset DBLP, Q2 are the queries on the Treebank and Q3 are the queries on the dataset XMark. Each set of queries contains two normal tree pattern queries and three regular tree pattern queries.

Table 6 Query cases.

Name	Dataset	Regular Tree Pattern
Q1.1	DBLP	//dblp/article[/author][//title][//url][//ee]//year
Q1.2	DBLP	//dblp[//article[/author->A][/title][/pages][/year]]
Q1.3	DBLP	//dblp[//article(/author->A:/title:/pages:/year)]
Q1.4	DBLP	//dblp[//article(/author->A:/title:(/pages|/year))]
Q1.5	DBLP	//dblp[//article(/author->A:/title:(/pages*):/year)]
Q2.1	Treebank	//S/VP//PP[//NN][//NP[//CD]/VBN]/IN
Q2.2	Treebank	//VP[/PP[/IN][/NP->N[/CD][/VBN]]]
Q2.3	Treebank	//VP[/PP(/IN:/NP->N[/CD][/VBN])]
Q2.4	Treebank	//VP[/PP(/IN|/NP->N[/CD][/VBN])]
Q2.5	Treebank	//VP[PP(/IN*|/NP->N[/CD][/VBN])]
Q3.1	XMark	//item[/location][//mailbox/mail//emph][/description//keyword]
Q3.2	XMark	//people[//person[/address->A[/zipcode]][/profile[/education][//age]]]
Q3.3	XMark	//people[//person(/address->A[/zipcode]:/profile[/education][//age])]
Q3.4	XMark	//people[//person(/address->A[/zipcode]|/profile[/education][//age])]
Q3.5	XMark	//people[//person((/address->A[/zipcode])*:/profile[/education][//age])]
Q3.6	XMark	//people[//person[/address->A[/zipcode=6]][/profile[/education][//age]]]

4.2. Experiment Results and Analysis



(a) Test on DBLP (b) Test on Treebank (c) Test on XMark

Figure 1 Comparison of average query time on different dataset.

Figure 1 shows the comparison on query execution time of two systems, CEDMFT and CepEngine[5] proposed by Yao Lu. It can be seen from the figure that the CEDMFT is much faster than CepEngine when querying the normal tree pattern. After the addition of join, or, the closure operator, the query performance is still superior to the CepEngine.

Figure 2 shows the throughput change for normal tree and regular tree pattern queries on XMark of different sizes when containing joint, or and Kleene closures. Throughput is the amount of data that can be processed per unit of time (MB/s). The overall trend of the polyline in the graph shows that the effect of the operation on the query is gradually reduced when the query document is increased, and the throughput gradually becomes stable as the amount of data increases. The query efficiency of the normal tree pattern and the comparison predicates is the highest, and when the amount of data is large enough, it can reach 3.3GB/s on average, and the comparison predicates does not obvious affect the efficiency of the system. The regular tree pattern part is more complicated with the traditional automaton, and the determination is more difficult, so the efficiency is obviously lower than that of the general tree mode. But its throughput is still at a high level, in the amount of data is large enough, the average can reach 1.2GB/s.

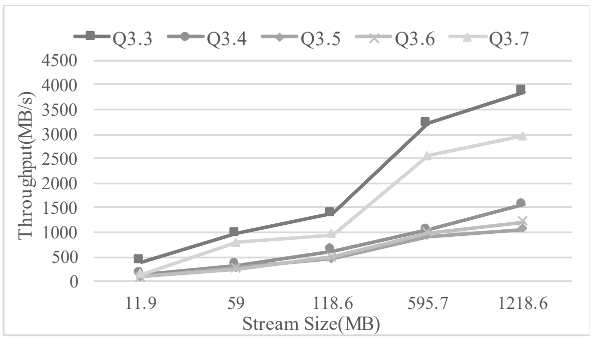


Figure 2 Throughput of different query on different XMark.

5. Conclusion and future work

This paper presents a complex event detection method based on macro forest transducers and implements a complex event detection system on the basis of it. It gives the translation method from regular tree pattern queries into macro forest transducers and also gives the event matching algorithm. By comparing with the complex event detection engine named CepEngine, it can be concluded that the complex event detection method based on macro forest transducers is a kind of complex event detection method with high performance. For the future, CEDMFT can then be used on a complex event processing platform to matching and filtering events. In addition, based on the good scalability of the macro forest transducers and the regular tree pattern, the semantics such as position predicates can be added to the regular tree pattern schema to enhance the support for complex event detection.

References

[1] Nishizawa I, Imaki T. Stream data processing system and stream data processing method: US, US7644110[P]. 2010-1-5.
[2] Buchmann A, Koldehofe B. Complex event processing[J]. IT-Information Technology Methoden und innovative Anwendungen der Informatik und Informationstechnik, 2009, 51(5): 241-242.
[3] Agrawal J, Diao Y, Gyllstrom D, et al. Efficient pattern matching over event streams[C]//Proceedings of the 2008 ACM SIGMOD international conference on Management of data. ACM, 2008: 147-160.
[4] Mozafari B, Zeng K, D'antoni L, et al. High-performance complex event processing over hierarchical data[J]. ACM Transactions on Database Systems (TODS), 2013, 38(4): 21.
[5] Yao Lu. Complex Event Detection based on Regular Tree Pattern Matching[D]. Beijing: Beijing University of Technology, 2016.
[6] Perst T, Seidl H. Macro forest transducers[J]. Information Processing Letters, 2004, 89(3):141-149.
[7] Engelfriet J, Vogler H. Macro tree transducers[J]. Journal of Computer and System Sciences, 1985, 31(1): 71-146.
[8] Hakuta S, Maneth S, Nakano K, et al. XQuery streaming by forest transducers[C]//Data Engineering (ICDE), 2014 IEEE 30th International Conference on. IEEE, 2014: 952-963.
[9] Xuezhi Feng, Husheng Liao, Hang Su. Construction of Macro Forest Transducers from XPath[J]. Transactions on Computer Science and Technology, 2015,4(3): 50-58.