ISSN No. 0976-5697

Volume 2, No. 6, Nov-Dec 2011

International Journal of Advanced Research in Computer Science

RESEARCH PAPER

Available Online at www.ijarcs.info

Optimized Query Processing for Virtual and Distributed Xml Databases

M.Sai Kumar*
M.Tech Student

C.Balarengadurai
Associate Professor

J.Srikanth
HOD & Associate Professor

Aurora's Engineering College
Affiliated to JNTU,Hyderabad
Bhuvanagiri, Nalgonda-508116
*mugalasaikumar@gmail.com
balamtech@yahoo.co.in
jsrikanth@aurora.ac.in

Abstract: This paper examines the optimization convention for XQuery queries, which is basically used for querying XML documents, such queries exploit type information of the preimage XML document provided in the XML schema. The optimization convention discussed in this paper are widely applicable for nearly every type of the XQuery expression and is found to be extremely useful, specifically in the context of XQuery queries on XQuery views. The concept behind the work of this paper is transformation of the XML Schema definition into a graph, which is further extended to a graph displaying the XQuery expression. The later extended part of the graph is further utilized to delete the sub expressions of the XQuery expression that are found to be unused for reaching the final set of result. Further the experimental result of our work demonstrates the improvement of our optimization.

Keywords: indexing, labelling scheme, query optimization, XML storage.

I. INTRODUCTION

Initially the XML (extensible Mark-up Language) was used as a human consumable exchange format. However, with the fast developments in the web world, the usage of XML data grew exponentially. The requirement of advanced tools by the various applications on the web like search engine, ecommerce, e-learning portals the usage of XML became even more intense. The search engines were not just used for retrieve the full text queries but also request for more specific data (structure queries) by the communities. This need resulted in the requirement for storing querying large scale XML data with higher efficiency and reliability. In the past many XML query languages have been proposed like Lorel [1], Quilt [2], XML-GL [3], XPath [4], XQuery [5], XOM [6], XAL [7] and YATL [8]. All these query languages used regular path expression, thus using the conventional approach such as tree traversal may have performance degrade especially on concurrent access.

The structure of the paper is as follows. The coherent past work discussed in Section 2, section 3 describes our general approach for optimizing XQuery queries. Section 4 presents a performance analysis and we end up with the summary and conclusions in Section 6.

II. RELATED WORK

The past works [8], [10] and [20] used algebra for XQuery. The work of this paper is advanced from the previous works due to the incorporation of the optimization approach, the previous works just listed out the transformation and optimization rules based on the introduced algebra. [2] Discusses the extension of language XQuery to support views but does not provide any information on how to optimize. [17] Turns the XML document to a sufficient XML fragment before processing the XQuery queries, it incorporates a static path analysis of the XQuery queries which results into a set of projection paths formulated in XPath. The approach followed by us optimizes the XQuery expression instead of projecting as a XML document, [9] examines the complexities of XPath query evaluation on XML documents. The complexity as considered by us for our XPath query evaluation algorithm on an XQuery graph is a part of our proposed optimization steps for XQuery queries. In the contrast of [7] which uses a graph schemas to optimize regular path expressions within queries for semi structured data, our methodology does not optimize a path expression according to a schema, but avoid unnecessary transformation steps by eliminating query code for the generation of output, which is not used further. Furthermore, we introduce an Structured Schema Graph (and an XQuery graph as extended version for XQuery expressions), which contain additional information in comparison to graph schemas like a sibling relationship, and we deal with XPath as path language. [14] Deals with the problem of compatibility of XPath expressions without respect to schema information as e.g. by an XML schema definition. In comparison, we introduce a fast (but incomplete) test that checks whether or not a given XPath expression is valid according to a given schema and according to a given XQuery expression.

In comparison to all other approaches, we focus on the optimization of XQuery queries based on the schema of the input XML document in order to eliminate unnecessary query code, which computes not used intermediate results.

III. THE GENERAL OPTIMIZATION APPROACH

Our approach is the design of an optimization step for the reduction XQuery query independently pursuant to the current XML document. The main advantage of this approach is that this method can optimize the XQuery query without any requirement of connection with the database. Since the outcome of XQuery expressions often includes complete sub-trees of the input XML document, the schema information can be used by us as the of input XML document for the motive optimizing queries. The motive is fulfilled by us by the introduction of Structured Schema Graph, which is derived from the schema of the input XML document. We use this Structured Schema Graph for search algorithms, which are outlined in Section 3.2 and Section 3.4.

```
<xsd:schema xmlns:xsd='http://www.w3.org/2001/XMLSchema'>
<xsd:element name='conference'>
<xsd:complexType>
<xsd:choice minOccurs=1 maxOccurs='unbounded'>
<xsd:element name='tutorial'>
<xsd:complexType>
<xsd:element ref='author' minOccurs=0 maxOccurs='unbounded'/>
</xsd:complexType>
<xsd:attributeGroup ref='name'/>
</xsd:element>
<xsd:element name='paper'>
<xsd:complexType>
<xsd:sequence>
<xsd:element ref='author' minOccurs=0 maxOccurs="unbounded"/>
<xsd:element name='references' minOccurs=0 maxOccurs=1>
<xsd:complexType>
<xsd:element ref='conference' minOccurs=0
maxOccurs='unbounded'/>
</xsd:complexType>
</xsd:element>
</xsd:sequence>
</xsd:complexType>
<xsd:attributeGroup ref='name'/>
</xsd:element>
</xsd:choice>
</xsd:complexType>
<xsd:attributeGroup ref='name'/>
</xsd:element>
<xsd:element name='author'>
<complexType mixed='true'/>
</xsd:element>
<xsd:attributeGroup name='name'>
<xsd:attribute name='name' use='required'/>
</xsd:attributeGroup>
</xsd:schema>
```

Figure. 1A. Schema of input XML document p.xml

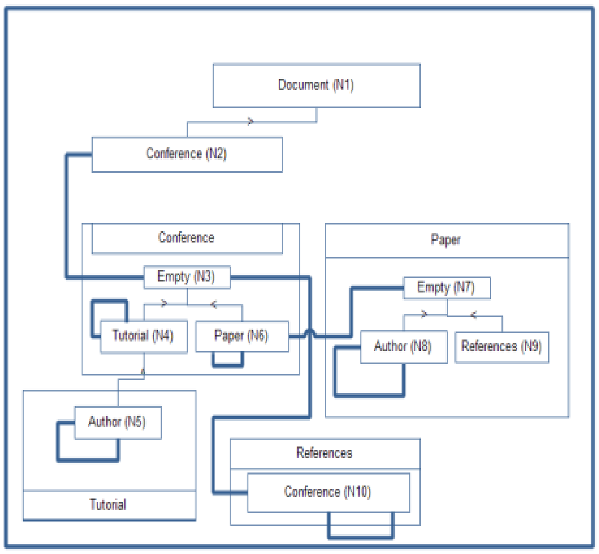

Figure. 1B. Structured Schema Graph of the schema defined in Fig. 1A

A. *Structured Schema Graph:*

The Structured Schema Graph is generated from an XML Schema definition. As an example, see the XML Schema definition.

Every individual node of an Structured Schema Graph represents an element of the node, which is termed as document node or else a dummy node (called: EMPTY node). Father the: EMPTY node represents a whole choice expression or a whole sequence expression. The edges are classified into three kinds as: parent-child edges, which represent the relation between element node, the document node and/or: EMPTY nodes, as that of a relation a parent and child. Between element nodes, a sibling edge represents a directed sibling relationship between element nodes, the document node and/or: EMPTY nodes. As a final point, an expression edge represents a relationship to a whole choice expression or a whole sequence expression between element nodes, the document node and: EMPTY nodes.

The generation of an Structured Schema Graph from an XML Schema definition can be done easily by following some general rules. The creation of a start node of type: Document-Node, for which the node representing that node is a child of the document node, which is also the root node of the XML document. From the example of Fig. 1A and Fig. 1B, N1 represents the document node and N2 represents the root element node.

B. *Rules:*

a. The addition of all (required, implied and fixed) attributes of any type of the corresponding XML element E to the nodes representing E is the second general rule. In the example of Fig. 1A and Fig. 1B, we add the attribute name to the node N2.
b. Further the transformation of the right-hand side of an element declaration is committed by the following rules:
c. The nodes of the Structured Schema Graph which represents the element are the parents of the representation of the right-hand sides of their element declarations. In the example of Fig. 1A and Fig. 1B, N1 is the parent of N2.
d. Whenever an element E1 can be a following sibling node of another element E2, we insert a sibling edge from E2 to E1. This is the case for repetitions of elements (see right-hand sides of element declarations of conference, tutorial, paper and references in Fig. 1A and Fig. 1B) and for sequences of elements.

Whenever the XML Schema defines an element to be a complexType defined by a choice (see right-hand side of conference) or a sequence (see right-hand side of paper), then we create an extra: EMPTY node for easy access to the whole choice expression or the whole sequence expression, respectively. As an example, see node N3 in Fig. 1 representing the xsd: choice element in Fig. 1A and node N7 representing the xsd: sequence element in Fig. 1A.

C. *Compatibility of an XPath Expression According to a Schema*

Definition: The compatibility of an XPath expression XP pursuant to a schema is correct, if and only if there exists at least one document, which is valid according to the schema, where XP is evaluated to a non-empty result.

The issue of compatibility of some subclasses of XPath expressions (without respect to a schema) is in NP [14]. A fast compatibility test is presented by us as an alternative for checking the compatibility of an XPath expression pursuant to a schema. Even though the test is fast but it is incomplete in the following ways. If we are pretty sure about the compatibility of an XPath that it is not compatible for a schema, the test results as NOT compatible, else the test may result as compatible.

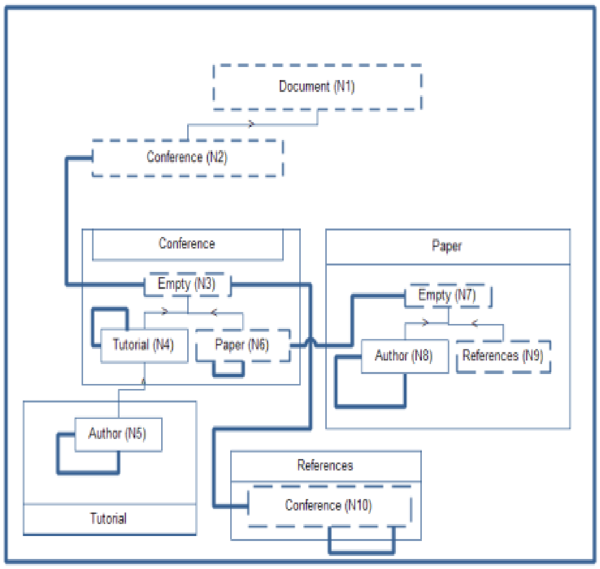

Figure 2. Structured Schema Graph with marked nodes according to the query (conference/paper/references/conference/paper)

The issue of uncertain results from the alternative test presented by us is addressed by us, by using a modified XPath expression evaluator to test the compatibility of XPath expression XP pursuant to the schema. The modified XPath evaluator test takes the schema and XP as the input. The First step is the creation of the Structured Schema Graph according to the schema by using the rules mentioned in Section 3.1. Next is the execution of the, modified XPath evaluator, which performs the job of verifying the hierarchical position of all the elements and attributes of path within XP. The modified XPath evaluator starts initiate the process of evaluation at the node of type: Document Node which represents the document node. Within the Structured Schema Graph, there is all necessary information in order to execute the modified XPath evaluator, as the parent-child-axis and the next sibling- axis are available in the Structured Schema Graph. The evaluator does not consume any element from the XPath expression XP and passes all the empty nodes. The Structured Schema Graph may contain loops in contrast of the XML document. Hence it becomes mandatory for the modified XPath evaluator to take loops into consideration at the time of revisiting a node but it did nit process the next location step inside the XP. To address this issue the modified XPath evaluator marks all the nodes that contribute to the successful evaluation of an XP.

For reference consider example, see the steps 1, 9 performed on the Structured Schema Graph of Fig. 2 for the successful evaluation of the XPath query /conference/ paper/references/conference/paper. The whole processing is completed till step 9, and as a result this XPath query is considered to be compatible. In general, with this technique, we can test whether or not a schema definition allows only XML documents for which a given XPath expression XP can never be successfully evaluated, i.e. for which the evaluation of the XPath expression returns an empty set.

D. XQuery Graph

```
(1)let $view :=
(2) <root>
(3) <result>
(4) {
(5) for $a in
(6) document('p.xml')/conference/paper
(7) let $b :=
(8) <single result>{$a}</single_result>
(9) return $b
(10) }
(11) {
(12) for $a in
(13) document('p.xml')/conference/tutorial
(14) let $b :=
(15) <single_result>{$a}</single_result>
(16) return $b
(17) }
(18) </result>
(19)</root>
(20)return
(21) $view/result/single_result/tutorial
```

Figure. 3A. Query for retrieving the tutorials

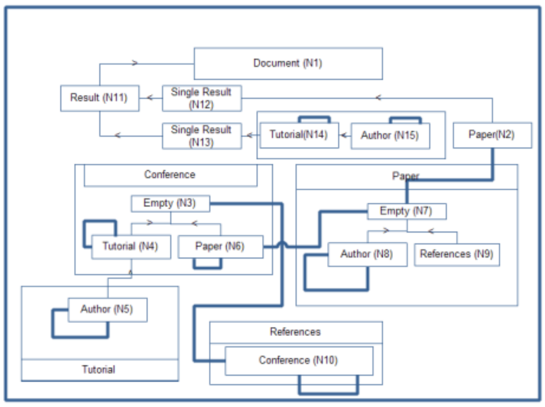

Figure. 3B. XQuery graph of the XQuery query of Fig. 3A

For the optimization rules, we consider that subset of XQuery, where the XQuery expression must conform to following rule Start in EBNF notation.

Start ::= (FunctionDecl)* FLRExpr.
FunctionDecl ::= "declare" "function" QName "(" ("$" QName ("," "$" QName)*)? ")" "{" ExprSingle "}".
FLRExpr ::= (ForClause | LetClause)+ "return" ExprSingle.
ForClause ::= "for" "$" VarName "in" ExprSingle.
LetClause ::= "let" "$" VarName ":=" ExprSingle.
ExprSingle ::= FLRExpr|IfExpr|PathExpr.
IfExpr ::= "if" "(" ExprSingle ")" "then" ExprSingle "else" ExprSingle.
PathExpr ::= ("/" RelativePathExpr?) | ("//"RelativePathExpr) | RelativePathExpr.
RelativePathExpr ::= (Step | PrimaryExpr)(("/"|"//") (Step | PrimaryExpr))*.
Step ::= ("child" | "descendant" | "attribute" | "self" | "descendant-or-self" | "following-sibling" | "following" | "parent" |"ancestor" | "preceding-sibling" | "preceding" | "ancestor-or-self") "::" (QName | "node()" | "*").
PrimaryExpr ::= "$" QName | Constructor | FunctionCall.
Constructor ::= ("element" | "attribute") QName "{" ExprSingle "}".
FunctionCall ::= QName "(" (ExprSingle ("," ExprSingle)*)? ")".

The subset presented above of XQuery consists of nested for-let-return clauses, if expressions, element and attribute constructors, declarations of functions and function calls. The general rules for the generation of XQuery graph from an XQuery expression is presented by us:

We generate an own XQuery graph for each variable assignment $view of the XQuery expression. Every expression within the variable assignment of $view, which generates output, gets its own new node N representing the output.

All the Variables which may be nested as well as not nested are completely replaced with their content. A new node N is set as parent node of every node in XQuery graph representing the output, which can also be generated as a child node to the output node N by the XQuery evaluator. Furthermore, we relate the new node N with every node in the XQuery graph representing output, which could be generated as sibling node to the output of node N by the XQuery evaluator, by a directed sibling relation. If the next generated output of the XQuery evaluator is a sub-tree of the input XML document specified by an XPath expression XP, we search within the Structured Schema Graph by the modified XPath evaluator as before, and we retrieve a node set SN of nodes of the Structured Schema Graph. We first copy the nodes of SN and copy all descendant nodes and all sibling nodes, which can be reached from the nodes of SN by parent-child relationships and sibling relationships. We copy also all parent-child relationships and sibling relationships for the copied nodes. Finally, we set the current node as parent node of the copies of SN. In the example of Fig. 3, which represents the XQuery graph of the XQuery query of Fig. 1, these copies consists of the node N2 and its descendant nodes N3 to N10, and the node N14 and its descendant node N15. We label the association with the XPath expression of the XQuery expression (or we use a reference; here, line number (5) and (12)).

E. Optimization

This section presents the most important feature of the work i.e. optimization. The general rules for the optimization of a query have been discussed in the next few paragraphs. The intiation step towards optimization of a query is as follows: At times when the content of a variable $view is queried by an XPath expression XP by $view/XP in the XQuery query, the following optimization steps are to be followed. The modified XPath evaluator is executed on the XQuery graph of the variable assignment which is of the view $view with the input XPath query XP. For the case of XQuery expression displayed in fig. 1, the XPath query XP is /result/single_result/tutorial for $view. All nodes have been marked by the modified XPath evaluator within the XQuery graph which are integral part of the successful evaluation of the query XP, similarly the evaluation of an XPath query is done on an Structured Schema Graph.

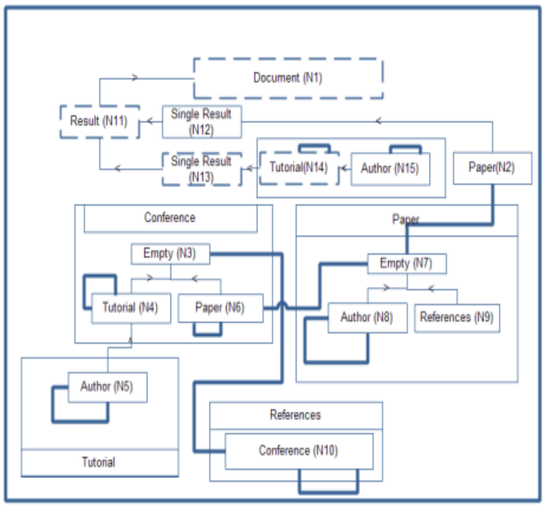

Figure. 4A. Marked nodes of Structured Schema Graph for (/result/single_result/tutorial)

```
let $view :=
<root>
<result>
{
for $a in
document('p.xml')/conference/tutorial
let $b :=
<single_result>{$a}</single_result>
return $b
}
</result>
</root>
return $view/result/single_result/tutorial
```

Figure. 4B. Optimized query of the query in Fig. 3A

The second step which includes deletion of the sub-expressions which are not so useful for the display of the results is commenced in the following way, all the sub-expressions of the XQuery expression from which the marked nodes in the XQuery graph has been generated and which do not assign variables are deleted but gradually i.e. all of them are not deleted.

The frequency and structure of execution of the result-statement is defined by the for-statement. Hence, only those for-statements are deleted which are retumstatement is reduced to an empty state.

All other sub-expressions which are unmarked are deleted by us and at last the optimized query remains. The pseudo code of the entire algorithm for optimizing XQuery queries is given in Fig. 4, which expects an XQuery query Q as input and which returns the optimized XQuery query Q' as output.

For the case of the XQuery query discussed in fig. 3A, firstly the XQuery graph is generated by us which is displayed in Fig. 10B for the variable assignment of $view. Further, the process of marking of nodes shown in Fig. 4A is done, followed by the last step of optimizing to the XQuery query in Fig. 4B.

F. Algorithm Optimize Query

```
Input: XQuery query Q conforming to rule Start in Section 3.3
Output: Optimized XQuery query Q'
(1) Generate abstract syntax tree T of Q
(2) Compute XQuery graph XG with marked nodes of T
(3) Mark all nodes in T, which correspond to marked nodes in XG
(4) while(all children of a symbol ExprSingle of a LetClause expression
are unmarked) do
(5) delete the whole LetClause expression
(6) For all nodes n in T do
(7) If(n and its child nodes are unmarked and (n is a symbol ExprSingle
and not(n is a parameter of a function call or n is a condition of an if-
statement)) ) then
(8) delete n (and its children)
(9) Compute Q' from the remaining nodes of T
```

Figure.5. Algorithm for optimizing XQuery queries

IV. PERFORMANCE ANALYSIS

An 1.7 GHz Intel Pentium 4 processor with 128 Megabyte RAM and windows 2000 operating system with JAVA VM build version 1.4.2 has been used as the test system for all experiments. We use the XQuery evaluator of Saxon version 7.9 [15].

The test input XML document of different sizes valid according the schema of Fig. 1A for all experiments are generated by us, where every paper and tutorial element consists of exactly one empty author element. Furthermore, we have used the XQuery query of Fig. 1 and the optimized query of Fig. 6. The mean results of 10 experiments have been displayed by us in the figures.

The first experiment for which the test input XML documents contains the same amount of paper and tutorial elements. The file size of the XML document have been increased by us from 8 kilobytes of doocument to 8 megabytes, further in Fig. 6 we have shown the evaluation time as a function of the filesize in Kilobytes. We found that the evaluation of the optimized query is approximately 1.8 times faster than the earlier original query for file size larger than 1.5 Mega bytes; through Fig. 7 we have presented the extent of fastness in the evaluation of the optimized query as compared to that of non-optimized query.

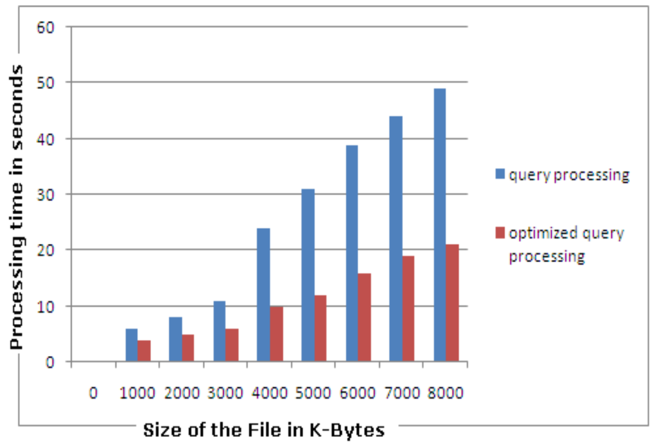

Figure. 6. Experiment 1: Evaluation time depending on filesize

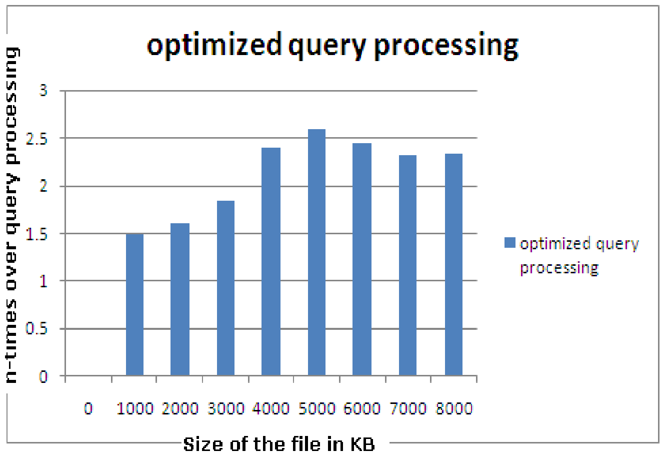

Figure. 7. Speed-up factor in Experiments conducted

V. CONCLUSIONS

Through this paper we have examined general rules of optimization, which are widely applicable for all XQuery queries and further which are very useful like in the context of XQuery queries on XQuery views. A new test have been introduced by us in the beginning of the work for the purpose of fast evaluation of the compatibility of XPath variable XP pursuant to the ordered schema, it exactly tells whiter or not a given schema permits only XML document. The extension of this tester which was incomplete in some sense was the second step of this paper. The extended, modified tester optimizes variable assignments of XQuery queries in such a way that the computation of irrelevant intermediate results is eliminated. We have proved by experimental results that the evaluation of the optimized queries saves processing costs depending on the amount of saved unnecessary intermediate results. Future work will include further optimization rules, which, for example, also optimize the order of operations within the XQuery expression.

VI. REFERENCES

[1]. S. Abiteboul et al, “The Lorel Query Language for Semistructed Data, Journal of Digital Libraries”, Vol 1, No 1, 1997, pp. 68-88.

[2]. J. Robie, J. Lapp, D. Schach, XML Query Language (XQL). Available http://www.w3.org/TandS/QL/QL98/pp/xql.html

[3]. S. Ceri et al, XML-GL : A Graphical Language for Querying and Reshaping XML Documents. Available http://www.w3.org/TandS/QL/QL98/pp/xml-gl.html

[4]. W3C, XML Path Language (XPath). Available http://www.w3.org/TR/xpath-datamodel/

[5]. W3C, XML Query (XQuery). Available http://www.w3.org/XML/XQuery

[6]. D. Zhang, Y. Dong, “A Data Model and Algebra for the Web”, Proceeding 10th International Workshop on Database and Expert System Application, IEEE Computer Society, 1999, pp. 711-714.

[7]. F. Frasincar, G. Houben, C. Pau, “XAL : An Algebra for XML Query Optimization”, 13th Australasian Database Conference, 2002, pp. 49-56.

[8]. V. Christophides, S. Cluet, J. Simeon, “On Wrapping Query Languages and Efficient XML Integration”, ACM SIGMOD International Conference on Management of Data, ACM Press, 2000, pp. 141-152.

[9]. Gottlob, G., Koch, C., and Pichler, R., The Complexity of XPath Query Evaluation, In Proceedings of the 22th ACM SIGMOD-SIGACT-SIGART symposium of Principles of database systems (PODS 2003), San Diego, California, USA, 2003.

[10]. Grinev, M., and Kuznetsov, S., Towards an Exhaustive Set of Rewriting Rules for XQuery Optimisation: BizQuery Experience, ADBIS 2002, LNCS 2435, pp. 340-345, 2002.

[11]. S. Groppe, and S. Böttcher, XPath Query Transformation based on XSLT stylesheets, Fifth International Workshop on Web Information and Data Management (WIDM’03), New Orleans, Louisiana, USA, 2003.

[12]. Groppe, S., Böttcher, S., and Birkenheuer, G., Efficient Querying of transformed XML documents, 6th International Conference of Enterprise Information Systems (ICEIS 2004), Porto, Portugal, 2004.

[13]. Sven Groppe, Stefan Böttcher, Reiko Heckel, Georg Birkenheuer. Using XSLT Stylesheets to Transform XPath Queries. Eighth East-European Conference on Advances in Databases and Information Systems (ADBIS 2004), Budapest, Hungary, September 2004.

[14]. Hidders, J., Satisfiability of XPath Expressions, DBPL 2003, LNCS 2921, pp. 21 – 36, 2004.

[15]. Kay, M. H., Saxon - The XSLT and XQuery Processor, http://saxon.sourceforge.net, April 2004.

[16]. Lakshmanan, L., Ramesh, G., Wang, H., Zhao, Z., On Testing Satisfiability of Tree Pattern Queries, In Proceedings of the 30th VLDB Conference (VLDB 2004), Toronto, Canada, 2004.

[17]. Marian, A., and Siméon, J., Projecting XML Documents. In Proceedings of the 29th VLDB Conference, Berlin, Germany, 2003.

[18]. Oracle, Oracle XQuery Technology – Preview, http://www.oracle.com/technology/tech/xml/xquery/index.html., 2004.

[19]. Papakonstantinou, Y., and Vianu, V., DTD Inference for Views of XML Data, In Proceedings of the Nineteenth ACM SIGACT-SIGMOD-SIGART

Symposium on Principles of Database Systems (PODS 2000), Dallas, Texas, USA, 2000.

[20]. Paparizos, S., Wu, Y., Lakshmanan, L. V. S., and Jagadish, H.V., Tree Logical Classes for Efficient Evaluation of XQuery, SIGMOD 2004, Paris, France, 2004.

Short Biodata for the Author

Mr. M.sai kumar received his B.Tech in Computer science and Engineering from Sri Indu college of engineering and Technology , JNTU, Hyderabad and Pursuing M.Tech in Computer science (Software engineering) from Aurora's Engineering College, JNTU, Hyderabad.

Mr. C. Balarengadurai received his B.Tech in Information Technology from Anna University Chennai, M.Tech in Information Technology from Sathyabama University Chennai and pursing his Ph.D in Artificial Intelligence at Manonmaniam Sundarnar University Tirunelveli, Tamilnadu. He is working as Associate professor in Computer Science & Engineering at Aurora's Engineering College with a teaching experience of 6 years. His area of interest includes Computer Networks, Compiler Design, and Artificial Intelligence and Wireless Networks.

Mr. Srikanth Jatla working as associate professor and Head of the Department of Computer Science and Engineering at Aurora's Engineering College with a teaching experience of 12years. He is a B.E and M.Tech in computer science and pursuing his PHD. In Data Stream Mining at JNTU, Hyderabad. His areas of interest include data structures, principles of programming language, algorithm analysis and complier design.