A New Query Engine using Novel Three Dimensional Index for Xml Documents

Atul D. Raut
J.D.I.E.T
Lohara
Yavatmal

M. Atique, Ph.D
PGDCSE SGBAU
SGBAU
Amravati

ABSTRACT
XML has gained prominence as data storage and exchange format for web applications. This is because there are certain features which are unique to XML like self descriptivism, extensibility and non proprietary text document storage. In spite of all these unique features XML has an inherent limitation of verbosity. This size problem of XML should be dealt with efficiently so that a good compression is achieved and at the same time the compressed data is directly queriable i.e. it should not require decompression at the time of querying. The proposed technique creates a new query engine based on novel three dimensional indexes consisting of structure, attribute and content index. The structure index consists of all unique root to leaf paths of the XML document, the content index stores the contents path wise i.e. all the contents of one particular type of path class is stored in one file and attribute index is created in manner similar to that of content index. Based on this three dimensional compact storage a new query engine is proposed which can answer xpath queries very efficiently. This approach dramatically reduces the storage requirement for XML coupled with efficient processing of xpath queries.

General Terms
Response time, inverted list, compact storage, algorithm

Keywords
Structure index, content index, attribute index.

1. INTRODUCTION
After relational database XML has gained wide acceptance particularly as standard for storage and exchange of data over the internet. Also the quantum of data processed by web applications like search engines, e-commerce and e-portals have increased enormously. This wide spread acceptance of XML is attributed to its desirable features of non proprietary storage in the form of text document and extensibility. In spite of its several advantages XML has a very serious drawback of verbosity i.e. the size of XML document is very large as compared to other popular storage formats like relational database. XML documents are verbose because of the repeated tags present in its structure. This size problem of XML and its wide spread acceptance necessitates the development of compact storage and indexing of XML for efficient querying of large volumes of XML databases. Indexing techniques used for relational database cannot be used directly for XML since XML data is ordered where as the relational data is unordered. Moreover XML contains structure in addition to data. The presence of structure makes the task of indexing much more difficult as compared to relational database. The most important factor for any efficient querying system is the time required to get result. This response time can be significantly reduced with the support of efficient indexing and storing technique for XML data.

For non redundant compact storage of XML and efficient processing of xpath queries we propose a new query engine based on novel three dimensional index structures comprising of structure, attribute and contents. The structure index is a compact summarization of all root to leaf paths and contains only unique paths including the attribute names. Since the leaf node contains the data leaf node here means one level above the leaf node. The size of structure index for majority of XML document is very small as compared to the XML document and only the structure index needs to be present completely in main memory during query processing. The structure index is constructed in the first scan of the document. This structure index provides a unique id to every path in it. This path id provides a link to the attribute and content index. The structure index implicitly represents the parent child and ancestor-descendant relationship. Such information had to be explicitly represented in the inverted list technique [1].

In the second scan of the document the attribute and content index are constructed. The attribute index stores attribute values for one path class in one file using the path id for that particular path in document order. The attribute index in addition to storing attribute values implicitly represents the sibling information for the nodes of the document. Again such information is explicitly represented in the inverted list technique. The content index is constructed in a manner similar to attribute index i.e. for particular path class all the contents are grouped and stored in one file in document order. The three indexes together provides path based element less XML storage.

The main contributions of the proposed technique can be summarized as below

1. It makes use of native storage for XML and does not rely on relational store.

2. Three indexes together implicitly represent the parent-child, ancestor-descendant and sibling information.

3. Query processing requires very less main memory as compared to document storage. Hence the query response time is very less.

4. It is suitable for partially as well as fully specified xpath query.

5. The three indexes together can answer partially as well as fully specified query without constructing the F&B bisimilarity graph which is required in case of FIX [2].
.

2. RELATED WORK

Till date several techniques have been implemented for querying XML documents using different indexing techniques by different researchers. These early techniques can be broadly classified into following types.

- By traversing the tree or its compressed representation.
- By using IR style processing using inverted list.
- By combination of the first two. In this context structure index plays a vital role.
- By using a Relational Database Management System.
- Techniques which utilizes efficient data structures like B^+ tree, hash table etc

Compact storage of XML can be classified as queriable and non queriable. A queriable compression technique is one which does not require decompression at the time of querying, while non queriable compression technique requires decompression. Xmill is the best known non queriable compressor considering the compression ratio, compression time and the memory required [3]. XMLPPM is also a non queriable technique with better compression ratio as compared to XMILL but requires a longer compression time [4]. XGRIND, Xpress, Xquec, Tbitmap, PCIndex, RFX, ISX TwigInlab are some of the queriable compression techniques. XGRIND has lower compression ration and requires more compression time as compared to XMILL. Both XGRIND and Xpress requires two scans over the XML document[5][6]. ISX achieves compact XML storage by storing it in a concise structure, but it is a schema aware storage technique [7].

Compact storage coupled with efficient querying of XML document started with modified form of inverted list used in information retrieval. These techniques were successful in element less compact queriable storage of XML. For querying different join algorithms were proposed. These algorithm produced intermediate results which resulted in longer query response time[8][9]. To reduce the intermediate result the concept of structure index was introduced. Structure index quickly removed traversing of irrelevant paths for the given query [10][11]. Since relational database is the commonly used database technology many researchers shredded xml data into relational tables. The underlying indexing and query optimization techniques of the relational database can be utilized readily, but the xpath or xquery expression had to be converted to SQL query. This conversion required some time which increased the query response time[12][13][14]. Some of the recent proposal like XTC, Xquec, RFX, PCIndex, Tbitmap made use of efficient data structure to store and process XML document. The Tbitmap technique assigns bitmap consisting of sequence of 0's and 1's to every unique node in the tree. If the number of nodes increases the size of bit map increased. This ultimately increased the query response time[15]. The PCIndex uses dewey id for node to represent the structural relationship among the nodes. Again if the number of nodes increased the memory requirement and the query response time increased. [16]. XTC and XQuec are based on path synopsis and grouping of data based on path synopsis.. Xquec gives good compression and has greater query capabilities [17]. XTC is based on the concept of structure virtualization leading to element less XML storage. It also provides good compression and greater query capabilities [18]. RFX is a redundancy free XML storage. It makes use of layered storage of XML data which eliminates the redundancy present in the repeated tags of XML document [19]. In [20] Massih R Amini et. al. presents learning based summarization of XML document. The novelty of this approach is that the summarization is prepared not only from the contents but also from the logical structure of the document.. This leads to intelligent summarization. Different indexing and querying technique can be compared on the basis of various factors such as intermediate results size, index size, response time, I/O required, flexibility etc.

3. PATH SUMMARY BASED ELEMENT LESS XML STORAGE

Compact Storage of XML is very essential because of its verbose nature. Compact Storage of XML has been done in two ways either by native storage or using relational database. Native storage of XML started with inverted list techniques used in information retrieval. In these techniques the tree representation of XML document had to be traversed and element was given a unique ID or label. For retrieving information different join algorithms were proposed by different researchers. These join algorithm produced large intermediate results which increased the query response time .The proposed technique achieves element less storage by separately storing the contents and attribute values based on the path summary i.e. the structure index. In the first scan of the XML document the structure index is created .The content and attribute index are created in the second scan.

3.1 Structure Index

The structure index is a summarization of all root to leaf paths of XML document i.e. the structure index contains all unique root to leaf paths including the attribute names. In the tree representation of XML document the leaf node contains the data. Hence the root to leaf path here means root to one node above the leaf node. Consider the following fragment of XML document.

```
<mondial>
 <country car_code="AL"  area ="28750" capital=
       "city_cid_AL"  membership="org_BESC org _CE>
        <name> Albania</name>
        <population>32496732</population>
        <population_growth>1.34</population_growth>
       <city id="city_cid"  is_country_cop="yes">
                 <name>Traine</name>
                <longitude>19.8</longitude>
        </city>
   </country>
   <country> car_code="GR"  area ="139140" capital  =
"city_Greece  Athens" Memberships="org_BIS  org_ccc">
          <name> Greece </name>
          <population_growth>0.42</population_growth>
          <prounce id="prov_cid_Greece" country="GR">
              <name>Anotoliki  </name>
              <city id ="city_cidcia_Greece_9" country="GR">
                 <name>Komotini  </name>
```

```
        </city>
      </province>
  </country>
</ mondial/>
```

In the above XML document the element country contains large no. of attributes for the element country and moreover the element country contains large number of child elements. The structure index for the given XML document is as shown below.

Path	Id
/mondial/country@carcode@area@capital@memberships/name	0
/mondial/country@carcode@area@capital@memberships/population	1
/mondial/country@carcode@area@capital@memberships/population_growth/	2
/mondial/country@carcode@area@capital@memberships/city@id@is_country_cap/name/	3
/mondial/country@carcode@area@capital@memberships/city@id@is_country_cap/longitude	4
/mondial/country@carcode@area@capital@memberships/province@id@country/name	5
/mondial/country@carcode@area@capital@memberships/province@id@country/city@id@country/name/	6

The concept of structure index has been used in many techniques [17] [21]. We further reduce the size of the structure index by using a unique structure compression technique. In the above structure index many paths differ only in the last component say for example path0 and path1 differ only in the last component. Hence path1 can be stored as /0/population/. The digit 0 indicates that the entries till the last component 'population' in path1 are similar to the entries in path 0 till the last component 'name'. Thus the structure index is compressed indirectly and hence does not require decompression .The modified structure for the above structure index is as shown below.

Path	Id
/mondial/country@carcode@area@capital@memberships/name	0
/0/population	1
/0/population _growth	2
/mondial/country@carcode@area@capital@memerships/city@id@country/name/	3
/3/longititude/	4
/mondial/country@carcode@area@capital@memberships/province@id@country/name	5
/mondial/country@carcode@area@capital@memberships/province@id@country/city@id@country/name/	6

By such partial storage of paths the size of the path index can be reduced to a significant extent.

Table1 shows the reduction in size of structure index due the structure compression technique.

Table 1: Reduction in size of Structure Index

Data Set	Size in MB	Structure Index	Modified Structure Index
Mondial	1.7	20KB	8.08 KB
Orders	5.12	239 bytes	174 bytes
LineItem	30.7	414 bytes	279 bytes

The structure index provides a unique id to every path in the structure index. This index id quickly removes traversing of all paths which are not present in the given query. The index id provided by the path index is the number of the file required to be accessed to access the attribute values for that path and if all the attribute values matches then the content file which stores the data for that particular path only. This kind of path based selective access significantly reduces the query response time. For example consider the following query

/mondial/country@carcode=AL@area=28750@capital=city_cid_AL@memberships=org_BSEC org_CE/population/

Without considering the attribute values the index id from the structure index for the above path is 1. Thus only the files named attrib1 and data1 will be accessed. Assuming that a file requires one block of storage the minimum number of blocks accessed for answering a query is three for queries which has data and only two for queries that has no data. The number of blocks/files to be accessed is directly proportional to the depth of tree i.e. the number of blocks/files to be accessed increases with the depth of the tree and decreases as the depth of the tree decreases. However for 80% of XML document the depth is 4 to 6 for 15% it is 6 to 10 and only for 5% it is 20 to 30 [22]. Hence we can say that the number of blocks access required is proportional to 0(n). The algorithm for constructing the structure given in [23] is presented in totality& considers the attribute names as well as its values. Since this algorithm considers the attribute values also it cannot be applied to mondial data set. This data set contains many attributes for the parent element and also the parent element contains many child elements .The algorithm for constructing the structure index presented here does not store the attribute value in the structure index, instead the values are stored separately in the attribute index based on structure index.

Algorithm Structure Index

1. Store of all child modes of the root node in reverse order on stack.
2. Repeat steps 2.1 to 2.4 till the stack is empty
   - 2.1 Pop the top node and put all its children nodes on top of stack.
   - 2.2 Traverse the popped node till the root node and again from root node till the current node.

2.3 From current node traverse till one level above leaf node & store the path so formed in path array say t.

2.4 For I = 0 to t. length -1 do

2.4.1 Compare the path with the previous path and if it is similar to previous path mark it is repeated 'R'.

2.4.2 If it is not similar then obtain the component till the last component of the current path. Repeat the same thing for the previous path. If the two components match then the array index value is written instead of the component in the current path. If there is no match then the path cannot be compressed.

3. Remove the path marked as 'R' from the path array

3.2 Attribute and Content Index

Considering the XML document of section 3.1 the first path in this document is

/mondial/country@car_code@area@capital@memberships/name/

From the structure index the id for this path is 0. Hence the values of all the attributes present on this path will be stored in the file named attrib0 and the contents will be stored in the file named data0. The second path in this document is

/mondial/country@car_code@area@capital@memberships/population/

The node 'population' is sibling of node 'name' & this path has indexe id of 1 as shown by the path index. Hence the values of all this attributes on this path will be stored in the file named attrib1 in the following way,

/0/. The digit 0 indicates that the values for all the attributes on this path are stored in the file attrib0. The digit0 in the file attrib1 further indicates that the nodes' name' and 'population' are siblings. Storage of attribute values in this fashion implicitly represents the sibling relationship among the nodes in the XML document .The inverted list techniques used in [9][10] required explicit representation of all types of relationships in the XML Document. In this technique the ancestor_ descendant & parent child relationship is implicitly represented by the structure index. Thus most of the relationship gets implicitly represented by the structure & attribute index. Finally consider the path

/mondial/country@car_code@area@capital@memberships/city@id@is_country_cap/longtitude/

The index id for this path as shown by the structure index is 4. Hence the values of all the attributes on this path will be stored in the file named attrib4 in the following way

/0/3/

The digit 0 & 3 indicates that the attribute values for this path are stored in two files attrib0 & attrib3. The given path can be considered to be made up of two sub paths the first sub path is

/mondial/country@car_code@area@capital@memberships

the values of attributes on this sub path are stored in the file attrib0 .The second sub path is

/city@id@is_country_cap/lingtitude/

for which the attribute values are stored in the file attrib3. For getting results of this type of queries first the file attrib3 will be accessed & if all the attribute values matches then the file attrib0 will be accessed .Thus as the depth increases the no. of blocks/files to be accessed for the attributes values increases. But as pointed out in section 3.1 for 80% of XML documents the depth is just 4. .

We now present a brief algorithm for constructing the attribute & content index

Algorithm Content Attrib Index

1. Read the XML document
2. For the root node store all the children nodes on top of main stack & in reverse order.
3. U← 0
4. Load the structure index in path array P.
5. While s.count > 0 do

5.1 If u < > 0 then

5.1.1 Store all the siblings of the current root to leaf path on sibling stack s1 in reverse order.

5.1.2 Pop all the nodes from the main stack those are equal to the node on the sibling stack

5.1.3 While s1.count > 0 do

5.1.3.1 Pop the top node from s1 in t.

5.1.3.2 For the currently popped node obtain the root to leaf path

5.1.3.3 For i←0 to p.length-1 do

5.1.3.3.1 Compare the current path with every path in the path array P.

5.1.3.3.2 If match is found remember the array index in index id.

5.1.3.3.3 Write the content in file whose name has two parts the prefix will be data & suffix will be index id. For example if the index is 3 the name of file will be data3

End For

5.2 For the node t obtain the attribute path from root node to t & from t to one level above the leaf node. Store the attribute values with appropriate sibling information in a file whose name will have a prefix of attrib & suffix will be index id .For example if the index is 3 then the name of the file will be attrib3.

Else

5.3 Pop the top node from the main stack s in n

5.4 For the currently popped node n obtain the root to leaf path & also the attribute path with values.

5.5 For i←0 to P.length-1 do

5.6 Compare the current path with every path in the structure index

5.7 If match is found remember the index id

5.8 Store the contents in the file whose name will have a prefix of data & suffix of index id

5.9 Store the attribute values in the file whose name will have a prefix of attrib & suffix of index id.

End For

End if

End while

4. QUERYING XML DOCUMENT

Xpath queries can be of the following types [24]

1) Path expression with single type of relationship

Such path expression contain either parent child or ancestor_descendant relationship

Ex: /mondial/country/name

/mondial//name

2) Path expression with mixed type of relation which contain both parent child as well as anscestor_descendant relationship

/mondial/country//name

3) Twig query with single type of relationship

/mondial[/country/name]/country/population

Above path expression returns two different types of path classes the first is

/mondial/country/name and the second is

/mondial/country/population

4) Twig query with mixed type of relationship

/mondial[//name]/country/population

For answering path queries with single or mixed type of relationship a single lookup of structure index is sufficient to get the index id once the index id is obtained then only the files that are related to this particular path are accessed. If all the attribute values of the given query matches with attribute values stored in the selected files, then the content file related only with the given path is accessed. For example for the mondial dataset consider the different types of queries

1) /mondial/country/name

Since the query does not contain any attribute the index id is obtained from the structure index which is 0 in this case .Then the content is accessed by accessing only the file data0 .Thus only two blocks/files are accessed one for structure index & one for the file data0.

2)/mondial//name

This query will result in four types of path sets because of the ancestor_descendant relationship

a)/mondial/country/name

b)/mondial/country/city/name
c)/mondial/country/province/name

d)/ mondial/country/province/city/name

Once the path id is obtained for each path class the above process can be repeated to get the results.

3)/mondial/country@car_code='AL'@area='28750'@capital ='city_cid_AL'@membership='arg_BESC arg_CE/population

Since the query contains attribute values, first the values of all the attributes are stored in temporary variables & then equal to sign(=) along with its values is removed from the query to get the following query

/mondial/country@car_code@area@capital@membership/population/

The index id for the above path as shown by the structure index is 1 .As the query contain attributes the file attrib1 will be accessed to get '/0' stored in it.

The digit 0 indicates that the attribute values for this path are stored in file attrib0 & the node population on this path is a sibling of the node name on the path

/mondial/country@car_code@area@capital@memberships/name

If all the attribute values of the query matches with the attribute values stored in the file attrib0 then the index location is determined. Finally to get the contents on this path the file data0 is accessed & the contents at the index location is returned .The index location from the attribute file attrib0 is 0. Hence the contents at index location in the file data1 which is 32496732 is returned. Finally consider the query

4)/mondial/country@car_code=GR@area=139140@capital=city_Greece_Althens@memberships=arg_bis@arg_ccc/province@id=prov=cid_Greece@country=GR/city@id=city_cia_Greece _9@country=GR/name

Removing the attribute values & equal to sign we get the following query

/mondial/country@car_code@area@capital@city@memberships/provence@id@country/city@id@country/name As seen from the structure index the index id for this path is 6. Hence the file attrib6 is accessed to get the following contents

/0/5/'city_cia_Greece_9,GR

The two attribute values in this file are compared with the last two attributes of the given query Match is found therefore the file attrib5 will be accessed to get the following contents

/0/prov_cid_Greece,GR

Again the two attribute values in this file are compared with the next last two attributes of the given query. A match is found in this case therefore the file attrib0 will be accessed to get the following contents

AL,2870,city_cid_AL,org_BESC org_CE
GR,139140,city_Greece_Althens,arg_Bis arg _ccc

The first four attributes of the given query are matched with the attributes present at index location 0. Since there is no match these attribute are matched with the attribute present at index location 1. Since there is a match the file data6 will be

accessed and the contents at index location 0 i.e. Kamotini is returned.

All the indexes [i.e. Data storage] were created using Intel Pentium PIV 3.0GHz system with 2GB of DDR Ram & VB.Net running on Windows XP Platform. Since VB.Net makes use of DOM representation for XML document our technique requires high main memory for creation of indexes [i.e. Data storage], but once the indexes are created. query processing can be done on a system with low main memory starting from 256MB.

Table 2 shows the comparison between different compression techniques & path summary based element less XML storage(PSEXS) based on compression. The results of other techniques have been citied from [19].

Table 2 Comparison between PSEXS and other techniques based on compression

Data Set	Size in MB	RFX MB	ISX MB	XMILL MB	XGRIND MB	PSEXS MB
Mondial	1.63	NA	NA	NA	NA	0.773
Orders	5.12	3	3	0.5	1.3	1.82
Shakesphere	7.5	5.1	5.3	0.9	2.1	3.7
Linestem	30.7	15.8	21	3.7	8.6	7.02
Treebank	84	NA	NA	NA	NA	45

5. EXPERIMENTAL RESULTS

Experimental results are given in terms of compression and query response time. The compression efficiency is measured in the form of CR1.

5.1 Compression Performance

The compression efficiency of any compression technique can be determined by the following two commonly used compression ratios [22].

1] CR1 =Size of Compressed Document / Size of original XML Document * 8 bits/byte

CRI indicates number of bits required to store a byte of original file. Lower value of CRI indicates god compression whereas a high value indicates poor compression.

2] CR2 =1- Size of Compressed File /Size of original File * 100%

CR2 gives the percentage (fraction) of file eliminated due to compression. Hence higher value of CR2 indicates good compression and lower value indicates poor compression.

Here we describe the benchmark datasets that are used to determine the compression and query performance of our technique. These data sets can be classified as a data centric or document centric. The data centric documents are those which have regular structure & may contain varying no. of attributes starting from very less to very high. Document centric XML documents, have an irregular structure, contains a lot of textual data & rarely contain attributes.

1) Orders: It is a data centric XML document with almost negligible attributes.

2) Shakesphere: It is document centric & contain records of plays written by Shakes sphere.

3) LineItem: It is a data centric XML document with every less attributes.

4)Mondial: It is data centric with large number of attributes for parent element & such parent element contain large number of child nodes.

5)Tree Bank: It is a highly skewed data set containing elements with varying depth & a maximum depth of 36.

Table 3 Comparison between PSEXS & other techniques based on CR1

Data Set	Size in MB	RFX MB	ISX MB	XMILL MB	XGRIND MB	PSEXS MB
Mondial	1.63	NA	NA	NA	NA	3.793
Orders	5.12	4.705	4.705	0.784	2.0399	2.854
Shakesphere	7.5	5.44	5.653	0.96	2.24	3.946
LineItem	30.7	4.103	5.454	0.961	2.23	1.823
Treebank	84	NA	NA	NA	NA	4.28

As seen from Table2 and Table3 XMILL has lowest value of CR1 for all the datasets which indicates that it has highest compression efficiency. But XMILL compression is non queriable i.e. it requires decompression before querying. The compression done by our technique is queriable i.e. it does not require decompression before querying & its CR1 is better than RFX, ISX & XGIND for many data sets**.**

5.2 Query Performance

Time required for getting the result of the query is very less because of the path based storage and path based selective retrieval as explained in Section3 and Section4. We now present the results of execution of different types of queries on the mondial dataset of 1.63MB size for which the three dimensional index consisting of structure, attribute and content is already constructed as explained in Section3

I) Path queries with single type of relationship (Low selectivity query)

Low selectivity means less condition i.e. no attributes and their values are specified in the query. Hence such query generates more result. Table 3 shows response time for path

queries with single type of relationship (Low selectivity query)

II] Path queries with mixed type of relationship (low selectivity)

Table 4 shows response time for path queries with mixed type of relationship (Low selectivity query)

III) Path queries with single type of relationship (high selectivity)

High selectivity means such queries will contain attributes & their values i.e. more condition. Hence such queries return a single result/value

1) /mondial/country@car_code=GR@area=131940@capital= cty-Greece-Athens@ memberships=org-BIS org-BSEC org-CE org-CCC org-ECE org-EBRD org-EIB org-CERN org-EU org-FAO org-G-6 org-IAEA org-IBRD org-ICC org-ICAO org-ICFTU org-Interpol org-IDA org-IEA org-IFRCS org-IFC org-IFAD org-ILO org-IMO org-Inmarsat org-IMF org-IOC org-IOM org-ISO org-ICRM org-ITU org-Intelsat org-MTCR org-NAM org-ANC org-NATO org-EN org-NSG org-OECD org-OSCE org- OAS org-PCA org-UN org-UNESCO org-UNIDO org-UNIKOM org-MINURSO org-UNOMIG org-UNHCR org-UPU org-WEU org-WFTU org-WHOorg-WIPOorg-WMOorg-WToOorg-WTrOorg-ZC/name/
141

2)/mondial/country@car_code=GR@area=131940@capital=cty-Greece-Athens@ memberships=org-BIS org-BSEC org-CE org-CCC org-ECE org-EBRD org-EIB org-CERN org-EU org-FAO org-G-6 org-IAEA org-IBRD org-ICC org-ICAO org-ICFTU org-Interpol org-IDA org-IEA org-IFRCS org-IFC org-IFAD org-ILO org-IMO org-Inmarsat org-IMF org-IOC org-IOM org-ISO org-ICRM org-ITU org-Intelsat org-MTCR org-NAM org-ANC org-NATO org-EN org-NSG org-OECD org-OSCE org-OAS org-PCA org-UN org-UNESCO org-UNIDO org-UNIKOM org-MINURSO org-UNOMIG org-UNHCR org-UPU org-WEU org-WFTU org-WHO org-WIPO org-WMO org-WToO org-WTrO org-ZC/province@id=prov-cid-cia-Greece-2@country=GR@capital=cty-cid-cia-Greece-Komotini/city@id=cty-cid-cia-Greece-9@country=GR@province=prov-cid-cia-Greece-2/name/
165

The value at the end of the query indicates the query response time in milliseconds. Since the mondial data set contains large number of attributes for parent element the test query becomes very long. From the above results which although are system dependent it is clear that query response time for all types of queries is very less. If the query response time is calculated in terms of no. of blocks/files accessed i.e. in a system independent manner still the time will be less because of the element less path based storage & selective path based access as explained in section 3&4.

Table 4. Response time for Path queries with single type of relationship (Low selectivity query)

Query	Time in Milliseconds
1)/mondial/country/name	94
2)/mondial/country/city/name	125
3)/mondial/country/province/city/name	313
4) /mondial//name	110

Table 5. Response time for Path queries with mixed type of relationship (Low selectivity query)

Query	Time in Milliseconds
1)/mondial/country//name	109
2)/mondial/country/province//name	125
3)//mountain/name	110
4)//mountain/height	112
5)//desert//latitude	110

6. CONCLUSION

In this paper a new query engine based on novel three dimensional index structures consisting of structure, attribute and content index for non-redundant compact XML storage and efficient querying is proposed. This results into path summary based element less XML storage. The experimental results for compression show that this technique achieves a good compression ratio and most importantly the compression is queriable. Since the technique uses the DOM representation the main memory and the compression time required is more as compared to other techniques, but the results of query performance are good for both low as well as high selectivity query.

7. REFERENCES

[1] S. Al. Khalifa, H. V. Jagdish, N Koudas, J. M. Patel, D Srivastava and Y Wu, " Structural Joins: A Primitive for Efficient XML Query Pattern Matching," In Proc. of the *18th International Conference on Data Engineering (ICDE*), San Jose, CA, pp. 141-152, February 26-March 1 2002.

[2] Zhang N, Tamer M. " FIX: Feature-based indexing technique for XML documents," In Proc of *32nd VLDB Conference*, Seoul, Korea , pp. 259-270, September 12-15 2006.

[3]H. Liefke and D. Suciu, "XMill: an efficient compressor for XML data," In Proc of *ACM SIGMOD international*

conference on management of data pages, pp. 153-24,2000.

[4] J. Cheney, "Compressing XML with multiplexed hierarchical PPM models," In Proc of *IEEE Data Compression Conference*, pp. 163-172, 2000.

[5]P. Tolani and J. Haritsa, "XGRIND: A query-friendly XML compressor," In Proc *18th International Conference on Data Engineering (ICDE)* IEEE Computer Society, pp. 225-234, 2002.

[6] J. Min, M. Park and C. Chung, "XPRESS: A queriable compression for XML data," In Proc. of the *ACM SIGMOD International Conference on Management of Data*, San Diego, California, 2003.

[7] R. Wong, F. Lam and W. Shui, "Querying and maintaining a compact XML storage," in *16thinternational conference on World Wide Web*, Banff, Alberta, Canada, 2007.

[8]Peter Bunaman , Martin Grohe, Christioph Koch," Path Queries on Compressed XML," In proceedings of the *29 th VLDB conference*, Berlin Germany ,2003.

[9] N. Bruno, N. Koudas, and D. Srivastava, "Holistic Twig Joins: Optimal XML Pattern Matching", In Proc. Of *21st ACM SIGMOD Int'l Conference on Management of Data* (SIGMOD'02), pp. 310–321, 2002.

[10]Raghav Kaushik, Rajasekar Krishnamurthy, Jeffery F. Naughton, Raghu Ramkrishnan,"On the Integration of Structure Index and Inverted List," In Proc. of the *204 ACM SIGMOD international conference on management of data*, Paris, France, pp.779-790, June 13-18 2004.

[11]Atique M, Raut AD, "Non redundant compact Xml storage for efficient indexing and querying of Xml documents," In: Communications in Computer and Information Science, VIT Vellore, pp 109-113, December 2012.

[12]Ibrahim Dweib, Ayman Awadi and Joan Lu.(2009,June). MAXDOR: Mapping XML Document into Relational Database. *The Open Information System Journal. 3*, pp. 108-122.

[13]Zhuyan Chan et. Al," Index Structures for Matching XML Twigs using Relational Query Processor," In Proceeding of *Data engineering workshop ICDEW* ,5-8 April 2005.

[14]Igor Totarinov, Stratis D Vigals, Kevin Beyer et.al.," Storing and Querying Ordered XML using a Relational Database System," In Proc. Of *ACM SIGMOD Int'l Conference on Management of Data*, Madison Wisconsin USA, pp. 204-215, 2002.

[15]Yin Fu Huang and Shin-Hang Wang," An efficient XML Processing based on combining T bitmap and Index Techniques," In Proc. *IEEE Symposium on Computers and Communication ISCC* 2008, Marrakech, Morocco, July 6-9 2008,pp 858-863.

[16]Li Ying, MaJun Sun Yun, "Applying Dewey Encoding to Construct XML Index for Path and Keyword Query," In Proc. *First International Workshop on Database Technology and Application 09*, Wuhan, Hubie,China, pp553-556, , 25-26 April 2009.

[17]A. Arion, A. Bonifati, G. Costa, S. D'Aguanno, I. Manolescu, and A. Pugliese, "XQueC: Pushing queries to compressed XML data," in *Proceedings of the 29th International Conference on Very Large Data Bases (VLDB'03)*, 2003.

[18]Christain Mathis et. al. "Storing and Indexing XML Documents upside down," *CSRD*, vol 24, pp 51-68, 2009

[19]Radha Senthilkumar, Priyaa Varshinee and A. Kannan. Designing and Querying a Compact Redundancy Free XML Storage. *The Open Information System Journal.* 3, pp. 98-107, June 2009.

[20]Massih R Amini, Anatosios Tambros, Nicolas Usunier, Mounia Lolmas. Learning based summarization of XML documents. *Information Retrieval.* 10, pp 233-255,2007.

[21] Anderi Arion, et.al. Path Summaries and Path Partitioning in Modern XML Databases. *World Wide Web*, vol 11 pp 117-151, 2008.

[22] Wilfred NG, Wai-Yeung Lam, and James Cheng. Comparative Analysis of XML Compression Technologies. *World Wide Web: Internet and Web Information Systems*, 9, pp 5–33, 2006.

[23]A. D. Raut, M Atique. Efficient querying of structure and contents for XML documents. *International Journal of Computer Applications*, 45, pp 30-37, 2012.

[24]Su-Cheng Haw and Chien-Sing Lee. Structural Query Optimization in Native XML Database: A Hybrid Approach. *Journal of Applied Sciences.* 7(20), pp. 2934-2946, 2007.