IMPACT: International Journal of Research in Engineering & Technology (IMPACT: IJRET)
ISSN(E): 2321-8843; ISSN(P): 2347-4599
Vol. 2, Issue 4, Apr 2014, 97-110
© Impact Journals

A NOVEL STRATEGY FOR FINDING FREQUENT ITEMS IN DYNAMIC XML DISSEMINATION

P. NAVEEN[1] & S. K. MUTHUSUNDAR[2]

[1]Post-Graduate Student, Department of Computer Science Engineering, Sri Muthukumaran Institute of Technology, Chennai, Tamil Nadu, India

[2]Professor & Head, Department of Computer Science Engineering, Sri Muthukumaran Institute of Technology, Chennai, Tamil Nadu, India

ABSTRACT

This paper proposes a novel framework for commendably ascertaining frequently accessed data items over a wireless XML broadcasting scheme. A proficient XML dissemination scheme is used for supporting twig pattern queries in the wireless environment. Twig pattern queries that contain intricate conditions are quite common and critical in XML query processing. The mobile client can retrieve the required data satisfying the given twig pattern by performing bit-wise operations on the Lineage Codes in the relevant G-nodes. Counter-based algorithm tracks a subset of items from the input, and monitor counts associated with these items. It decides for each new arrival whether to store this item as frequent item or not. The proposed framework is reliable, efficient and user-friendly.

KEYWORDS: Attribute Summarization, Frequent Items, Twig Pattern Query, Wireless Broadcasting

INTRODUCTION

Mobile devices are the building blocks of Wireless environment. Recent advances in communication technology have greatly increased the functionality of mobile information services. An important application is to provide various types of real-time information such as stock quotes, weather conditions and traffic information, to clients. Data broadcasting (Acharya et al. 1995; Kaushik 2004) is one among the efficient data dissemination strategies which is very cost effective in disseminating a substantial amount of information to a large number of mobile clients.

Wireless broadcasting is a successful information diffusion move towards the wireless environment for the reason that: 1) the server can support an enormous number of mobile clients without extra costs (i.e., scalability), 2) the broadcast channel is shared by many clients (i.e., the effectual utilization of bandwidth), and 3) the mobile clients can receive data without sending request messages that consumes huge energy. In today's context, there is a need to consider energy conservation of mobile clients and also reduced query processing time to provide efficient and fast response to the users (i.e., latency-efficiency). To measure the energy-efficiency and latency-efficiency in wireless broadcasting (Chung et al. 2010; Imielinski1997), the tuning time and access time are used, respectively. The frequent items (Cormodeet al. 2010) problem is to process a stream of items and find all items that occurs more than a given fraction of the time. Typically, this is formalized as finding all items whose frequency exceeds a specified fraction of the total number of items. The items can represent packets on the Internet, queries made by the client and the weights are the size of the packets. If the items represent queries then the frequent items are now the (currently) popular terms. It is important to find algorithms which are capable of processing each new update very quickly, without blocking. It also helps if the working space of the algorithm is very small. Obtaining efficient and scalable solutions to the frequent items problem is also important since many streaming applications need to find frequent items as a subroutine of another, more complex computation.

This paper proposes a novel framework for finding frequent items in wireless XML streaming scheme that supports queries in the wireless mobile environment by addressing 1) a streaming unit called G-node which eliminates structural overheads of XML documents, and enables mobile clients to skip downloading of irrelevant data during query processing, 2) algorithms for generating wireless XML stream with G-nodes and query processing over the wireless XML stream, and 3) also algorithms for identifying frequent queries on client side from query directory that stores the queries and their responses.

This paper is organized in different sections addressing the problem statement, system architecture (Figure 1), G-node and attribute summarization technique, Lineage Encoding and related operators, algorithms for Query processing and for finding frequent queries, and performance of the proposed method followed by conclusions and suggestion for future work. The efficacy of the proposed scheme is validated through experiment results and comparison against the wireless XML streaming/conventional XML query processing methods (Park et al. 2010;Wang 2005).

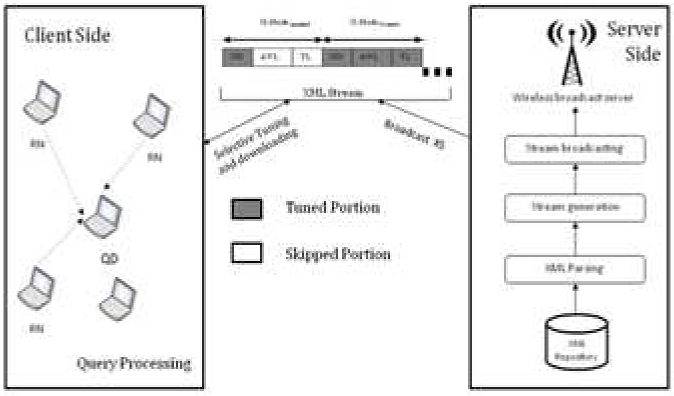


Figure 1: System Architecture

BACKGROUND

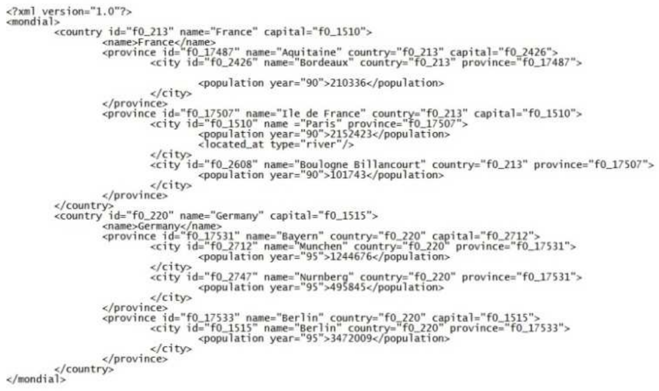
- **XML**

As a defacto standard for information representation and exchange over the internet, XML has been used extensively in many applications. Most of Web data sources are represented by XML specification. As a result, wide spectrum of users needs to query XML. An XML document contains hierarchically nested elements, and usually is modeled as a document tree. Elements, attributes, and texts are represented by nodes, and the parent-child relationships are represented by edges in the XML tree. Figure 2 shows a simple XML document that will be used as a running example in the paper.

- **XPath**

In this paper XPath (Berglund2002) is used as a query language. XPath is a well-accepted language for addressing parts of an XML document. It also serves a stand-alone query language for XML. Methods (Chen 2006; Cormode 2010) for efficient evaluation of XPath queries benefit systems for more powerful languages (e.g., XQuery) which incorporate XPath. An XPath query consists of a location path and an output expression. The location path is a sequence of location steps that specify the path from the document root to a desired element. The output expression specifies the portions or functions of a matching element that form the results. Each location step has an axis, a node test, and an optional predicate.

Q1://Country[@id="f0_162"]/Province/Text()



```
<?xml version="1.0"?>
<mondial>
        <country id="f0_213" name="France" capital="f0_1510">
                <name>France</name>
                <province id="f0_17487" name="Aquitaine" country="f0_213" capital="f0_2426">
                        <city id="f0_2426" name="Bordeaux" country="f0_213" province="f0_17487">

                                <population year="90">210336</population>
                        </city>
                </province>
                <province id="f0_17507" name="Ile de France" country="f0_213" capital="f0_1510">
                        <city id="f0_1510" name ="Paris" province="f0_17507">
                                <population year="90">2152423</population>
                                <located_at type="river"/>
                        </city>
                        <city id="f0_2608" name="Boulogne Billancourt" country="f0_213" province="f0_17507">
                                <population year="90">101743</population>
                        </city>
                </province>
        </country>
        <country id="f0_220" name="Germany" capital="f0_1515">
                <name>Germany</name>
                <province id="f0_17531" name="Bayern" country="f0_220" capital="f0_2712">
                        <city id="f0_2712" name="Munchen" country="f0_220" province="f0_17531">
                                <population year="95">1244676</population>
                        </city>
                        <city id="f0_2747" name="Nurnberg" country="f0_220" province="f0_17531">
                                <population year="95">495845</population>
                        </city>
                </province>
                <province id="f0_17533" name="Berlin" country="f0_220" capital="f0_1515">
                        <city id="f0_1515" name="Berlin" country="f0_220" province="f0_17533">
                                <population year="95">3472009</population>
                        </city>
                </province>
        </country>
</mondial>
```

Figure 2: An Example of XML Document

For example, the location path of the query //country[@id="f0_162"]/province/text() is //country[@id="f0_162"]/provinc. The output expression, text(), indicates that only the text content of the matching name appears in the result. In the first location step, //country[@id="f0_ 162"], // is the closure axis denoting descendant-or-self, country is the node test, and @id="f0_162" is the predicate. The predicate restricts the results to the province sub-elements of country that have an id sub-element whose content has value as f0_162.

- **Twig Pattern Query**

A twig pattern query consists of two or more path expressions, thus, it involves element selections satisfying complex patterns in tree-structured XML data. The twig pattern query is a core operation in XML query processing and popularly used as it can represent complex search conditions (Al-Khalifa et al. 2002; Tatarinov et al. 2002). For example twig pattern query "/mondial/country[name/text()="Germany"]/province/city" is to find cities located in the provinces of a country that has a child element "name" whose text content is "Germany".

- **Structure Indexing**

Many techniques (Jun Pyo Park et al. 2013; Wang et al. 2005) using a structure index have been proposed for efficient XML query processing. The structure indexing directly captures the structural information of XML documents and is used for XML query processing. Conventional wireless XML streaming methods (Park et al. 2005; Park et al. 2006) using a structure index exhibit good performance for simple path query processing benefitting from the size reduction but they do not support twig pattern queries.

WIRELESS XML STREAM

In this section, stream organization for XML data is explained. This section(A) presents the unit structure of the stream called G-node. The next section (B) explains attribute summarization. Section (C) presents XML stream generation algorithm.

- **G-Node**

The wireless XML stream consists of the sequence of integrated nodes, called as G-node (Park 2010). These integrated groups are useful for selective access. The G-node is denoted by

Gp= (GDp, AVLp, TLp)

Definition 1: *The G-node is a data structure containing information of all the elements e_p whose location path is p, where GD_p is a group descriptor of G_p, AVL_p is a list containing all attribute values of e_p, and TL_p is a list containing all text contents of e_p.*

Group Descriptor: It is a collection of indices for selective access of a wireless XML stream. It is denoted by

$GD_p = (N_p, LP_p, CI_p, AI_p, TI_p, ST_p, ET_p)$

Node name (N_p) is the tag name of integrated elements, and Location path (LP_P) is an XPath expression of integrated elements from the root node to the element node in the document tree. Child Index (CI_P) is a set of addresses that point to the starting positions of child G-nodes in the wireless XML stream.

Attribute Index (AI_P) contains the pairs of attribute name and address to the starting position of the values of the attribute that are stored contiguously in Attribute Value List. Text Index (TI_P) is an address pointing to the starting position of Text List. Stream Created Time (ST_P) is the time when the XML stream is created and Expiry Time (ET_P) is the Validity period i.e., is the time to live period for the broadcasted data. Figure 3 shows an example of the Group Descriptor.

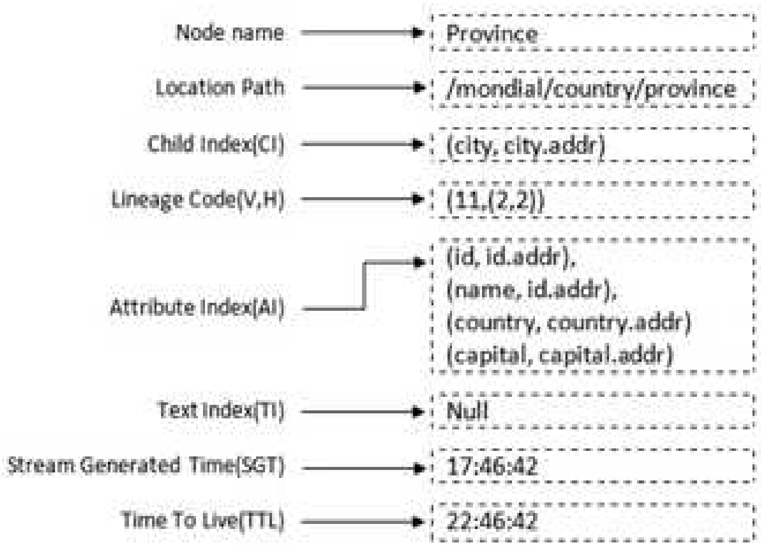


Figure 3: Group Descriptor (GD) for G-Node$_{\text{Province}}$

Attribute Summarization

Attribute summarization is a technique (Park 2005) used to reduce the size of wireless XML stream. A structural characteristic exists in the elements of XML data. It is that elements with the same tag name and location path often contain the attributes of the same name. In attribute summarization, the redundant attribute names of elements are eliminated, thus, the size of XML stream can be significantly reduced. Figure 4 illustrates summarized Attribute for G-node$_{\text{Country}}$.

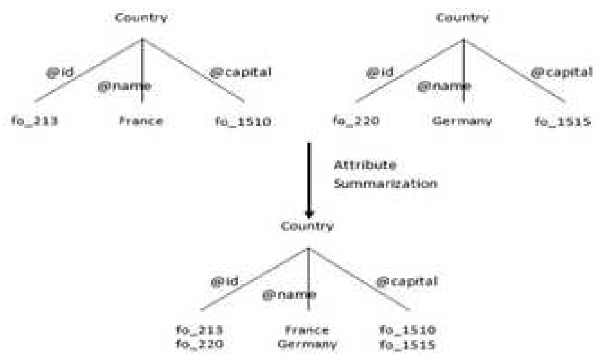


Figure 4: Attribute Summarization for G-Node$_{Country}$

- **Wireless Stream Generation**

An XML document to be distributed is retrieved from the XML repository (XML Data Repository2012) by the server and uses SAX (Simple API for XML) (SAX 2004) which is an event-driven parser to generate Stream. After the streaming of XML data, streamed XML data distributed via a broadcasting channel.

Algorithm 1. Wireless XML stream generation

Input: A well-formed XML document D, TTL value

Output: Wireless XML Stream XS

```
01: ContentHandler.startDocument()
02: Path Stack = NULL;
03: G-node Queue GQ = NULL;
04: Set depth as 1 and nodeId as 0;
05: ContentHandler.startElement()
06: Increase depth and nodeId;
07: IF (path p of the current node e does not exist in PS)
08: Construct a new G-node G with Tag name, AI, AVL;
09: Push p into PS;
10: ELSE
11: Get a G-node G of path p from GQ;
12: Add attribute values to AVL of G;
13: END IF
14: Set the depth th parent id as nodeId;
15: Add (depth-1)th nodeId to parent list of G;
16: Add nodeId to element list of G;
```

```
17: Enqueue G into GQ;
18: ContentHandler.endElement()
19: Decrease depth;
20: ContentHandler.characters()
21: Get a G-node G of path p from GQ;
22: Add text content to TL of G;
23: Enqueue G-node G into GQ;
24: ContentHandler.endDocument()
25: WHILE (the end of GQ is not detected)
26: Get top entry G-node G, its child G-node Gc in GQ
27: Generate AI and TI of G;
28: Compare element list in G and parent list in Gc;
29: IF (a parent element exists) THEN
30: Compute no of elements n of same parent in Gc;
31: Add a 1-valued bit to Lineagecode(V) of Gc;
32: Add an integer n to Lineagecode(H);
33: ELSE Add a 0-valued bit to Lineagecode(V);
34: END IF
35: Set CI to the child G-nodes of Gp;
36: END WHILE
37: Flush G-nodes in GQ into wireless XML Stream XS;
```

LINEAGE CODING

Lineage Encoding (Park 2010) is a scheme that supports queries involving predicates. To denote parent-child relationship between XML elements in two G-nodes two kinds of lineage codes, i.e. vertical code denoted by Lineage Code(V) and horizontal code denoted by Lineage Code(H), are used.

- **Lineage Code Definition.** *Assume that a G-node C is a child of a G-node P. Let $E_C=\{c_1, c_2, . . . ,c_m\}$ and $E_P=\{p_1,p_2, . . . ,p_n\}$be the ordered sets of elements in C and P, respectively, where the elements are ordered in document order. We suppose that $g : E_C \rightarrow E_P$ is a function that maps an element in E_C to its parent element in E_P. Lineage Code of the G-node C is defined by Lineage Code(V, H), where Lineage Code (V) is a bit string $V = b_1 b_2 . . . b_n$ where*

$b_i=$	1,	*if the i-th element in E_P; p_i has at least one child element in E_C(i.e., there exists $c_j \in E_C$ such that $p_i = g(c_j)$)*
	0,	Otherwise.

($1 \leq i \leq n$) and Lineage Code (H) is an ordered list of positive integers ($n_1, n_2, \ldots, n_k$), where n_i, ($1 \leq i \leq k$) is the number of elements in E_C whose parent element is p_j in E_P where $j=f_V(i)$

fV(i) represents represents the position of a bit in the bit string V which is the ith one among those with a value of 1. Figure 5 shows an example of Lineage Codes in G-node$_{\text{country}}$, G-node$_{\text{province}}$, and G-node$_{\text{city}}$. Lineage Code(V) of a G-node$_{\text{city}}$ is the elements integrated in and are mapped to the elements in it parent. Lineage Code(H) of a G-node denotes the number of child elements that are mapped to the same parent element in document order.

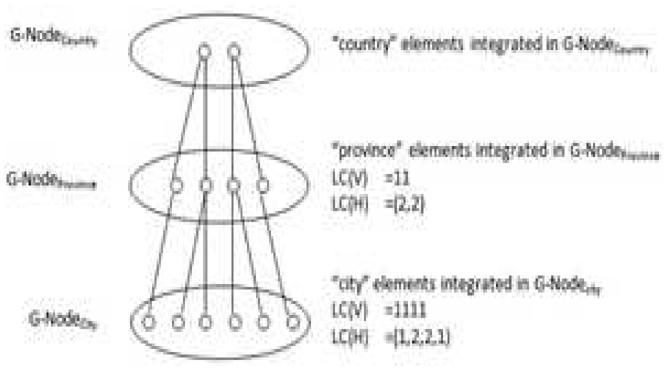

G-Node$_{\text{Country}}$
"country" elements integrated in G-Node$_{\text{Country}}$
G-Node$_{\text{Province}}$
"province" elements integrated in G-Node$_{\text{Province}}$
LC(V) =11
LC(H) =(2,2)
G-Node$_{\text{City}}$
"city" elements integrated in G-Node$_{\text{City}}$
LC(V) =1111
LC(H) =(1,2,2,1)

Figure 5: Lineage Codes of G-Node$_{\text{Country}}$, G-Node$_{\text{Province}}$, and G-Node$_{\text{City}}$

- **Selection Function**

In evaluating a given query with predicates, we should select a subset of elements of a particular type satisfying the given predicates, then have to find their child elements. A subset of the elements selected in a G-node can be represented by a bit string, called a selection bit string (SB) for the G-node, where 1-value bits identify the selected elements. In this section, selection functions to compute a selection bit string for the G-node is explained.

Select Children (C, SBp): Select Children, a function to obtain a selection bit string identifying a subset of elements in a particular child G-node. Given a set S of elements in a G-node P identified by a selection bit string SBp, the function determines a subset of elements in a child G-node C that are children of the elements in S. A selection bit string SBc for C can be computed based on the Lineage Code of C, (V, H), using Shrink & Mask and Unpack operators in order. Shrink & Mask(V, SB_P) first shrinks the bit string V by removing the bits with a value of 0 from V. It also shrinks the bit string SB_P, eliminating the bits in the same positions as those removed in V. Then, it calculates Bitwise AND of the two bit strings shrunken from V and SB_P. For example, Shrink & Mask (011110, 110011) = 1111 & 1001=1001.

Unpack(V, H) returns a bit string V_r which is obtained by concatenating bit strings S_i in order, where S_i is a bit string of the length n_i in which all bits are equal to v_i. For example, Unpack(1001, (2, 2, 2, 2) = 11000011.

Select Parents (C, SBC): Select Parents is a function to identify the parent elements of a subset of elements selected in a given G-node. Given a selection bit string SB_C for a G-node C, Select Parents(C, SB_C) computes a selection bit string SB_P for the parent G-node of C. The selection bit string SB_P can be derived from the Lineage Code of C, (V, H), using Pack and Expand & Mask operators in order. Pack(SB_C, H) shrinks the bit string SB_C using H to compute a bit string Vm indicating the elements in the parent G-node of C which are parents of the elements in C selected by SB_C. It returns a bit string $V_r = r_1 r_2 \ldots r_k$, where r_i = Bitwise-OR {$v_{p+1}, v_{p+2}, \ldots, v_{p+ni}$), where

$$p = \begin{cases} 0, & \text{if } i = 1 \\ \sum_{j=1}^{i-1} n_j, & \text{Otherwise.} \end{cases}$$

For example, Pack(10010, (3,1,1)) = 110.

Expand & Mask (V, V_m) expands the bit string V_m to the length of V by placing the bits in W to the same positions of the bits with a value of 1 in V in order and inserting the bits with a value of 0 in the remaining positions. Then it masks the bit string V by the expanded bit string. For example,

Expand & Mask(011010, 110) = 011010 & 011000 = 011000.

Get Selection Bit String of (N): This section explains the function to select elements in a G-node contained in the query tree of a given twig pattern query. The answer element is an element that satisfies given predicate conditions. The recursive function performs a traversal of the subtree rooted at G-node N in the post order, depth-first manner. First, it evaluates a selection predicate on the attributes or text over the elements in the G-node to obtain a selection bit string for the predicate. For each child node C of an internal node N, the algorithm computes its selection bit string SB_C recursively and then calculates the selection bit string SB_P for N which is derived from SB_C and Lineage Code of C using the Select Parents() function. Then, the result selection bit string for N is produced by performing bitwise AND operations over all the selection bit strings SB_P obtained from the child nodes of N.

TWIG PATTERN QUERY PROCESSING (OVER WIRELESS XML STREAM IN WIRELESS ENVIRONMENT)

In this section, the paper describes how a mobile client can retrieve the data of its interests to process the query over wireless XML stream. Assuming that there is no descendant axis in the user query, query processing for a Twig pattern query is presented in Section (A) and finding frequent data item is described in Section (B).

- **Twig Pattern Query Processing**

Algorithm 2 shows the Twig pattern query processing over the wireless XML stream. Twig pattern query processing involves three phases: Tree traversal phase, Subpaths traversal phase, and Main path traversal phase. The main path denotes a path from the root node to a leaf node while the subpaths denote branch paths excluding the main path in the query tree.

In the Tree traversal phase, the mobile client first constructs a query tree. Then, traversing the query tree it selectively downloads group descriptors of the relevant G-nodes into the nodes in the query tree. Attribute values and texts involved in the given predicates are also downloaded into the relevant nodes. In the Subpaths traversal phase, the mobile client performs a post-order depth-first traversal starting from the highest branching node in the query tree using the GetSelectionBitStringOf() function. The selection bit string for the branching node is calculated from all the subpaths in a bottom-up manner. Finally, the Main path traversal phase propagates the selection bit string on the branching node along the main path using the SelectChildren() function . Finally, the mobile client retrieves the set of answer elements in the leaf node of the main path which satisfies the given twig pattern query.

Algorithm 2. Twig Pattern Query Processing

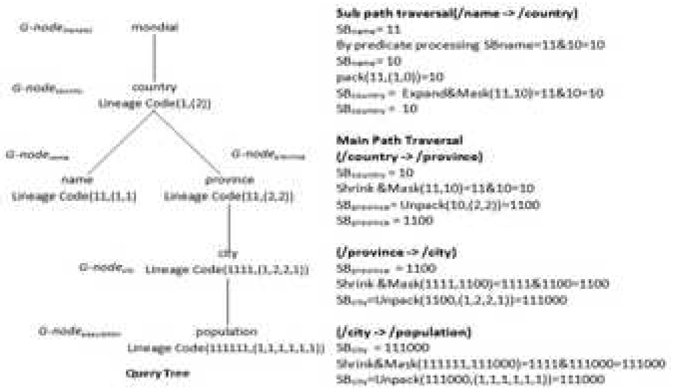
Input: Wireless XML Stream DS, a twig pattern query Q

Output: Result set R satisfying Q

```
01: result set R = Φ; // initialization
02: Initialize the selection bit string SB as 1;
03: Initialize Lineage Code of root G-node as (1, (1));
04: Initialize nextNode as address of root G-node in DS;
05: // Tree traversal phase
06: Construct a query tree T for Q;
07: REPEAT {
08: Tune a group descriptor GD of the G-node indicated
by nextNode;
09: IF (current node CN is the leaf node in TÞ THEN
10: Store AVL and TL the node in T;
11: ELSE
12: IF (CN contains predicate conditions PÞ THEN
13: Tune the relevant attribute values/text using AI/TI;
14: Store relevant attribute values/text into node in T;
15: END IF
16: Assign address of the next node in CI to nextNode;
17: END IF
18: } UNTIL (all nodes in T are completely traversed)
19: // subpaths traversal phase
20: Let N be the highest branching node in T;
21: SBN = Get Selection Bit String of(N);
22: // Main path traversal phase
```

```
23: Let MP be the main path in T starting from N;
24: P = N;
25: $SB_P = SB_N$;
26: REPEAT {
27: Let C be the child node of P in MP;
28: SBc = Select Children(C, SBp);
29: P = C;$SB_P = SB_C$;
30: } UNTIL (C is the leaf node)
31: Select R of elements in C by the selection bit string SBc;
32: Return R;
```

Figure 6 shows query processing steps for a twig pattern query, "/mondial/country[name/text()="France"]/province/city/population". In Tree traversal phase, the mobile client downloads group descriptors of six G-nodes (i.e., G-node$_{mondial}$, G-node$_{country}$, G-node$_{name}$, G-node$_{province}$, G-node$_{city}$ and G-node$_{population}$). In Subpaths traversal phase, the mobile client computes the selection bit string of the subpath (10) using the Get Selection Bit String Of() function. In Main path traversal phase, the mobile client performs the SelectChildren() function from the branching node (i.e., G-node$_{country}$) to the leaf node (i.e., G-node$_{population}$).



```
Sub path traversal(/name -> /country)
SBname= 11
By predicate processing SBname=11&10=10
SBname= 10
pack(11,(1,0))=10
SBcountry = Expand&Mask(11,10)=11&10=10
SBcountry = 10

Main Path Traversal
(/country -> /province)
SBcountry = 10
Shrink &Mask(11,10)=11&10=10
SBprovince= Unpack(10,(2,2))=1100
SBprovince = 1100

(/province -> /city)
SBprovince = 1100
Shrink &Mask(1111,1100)=1111&1100=1100
SBcity=Unpack(1100,(1,2,2,1))=111000

(/city -> /population)
SBcity = 111000
Shrink&Mask(111111,111000)=1111&111000=111000
SBcity=Unpack(111000,(1,1,1,1,1,1))=111000
```

Figure 6: Example of Twig Pattern Query Processing

- **Finding Frequent Data Items**

The frequent items problem is one of the most heavily studied questions in data streams research. Given a sequence of items, the problem is simply to find those items which occur most frequently. Typically, this is formalized as finding all items whose frequency exceeds a specified fraction of the total number of items.

Algorithm 3 shows Counter-based Algorithm. It stores *k* (item, counter) pairs. The natural generalization of the algorithm is to compare each new item against the stored items *T*, and increment the corresponding counter if it is among them. Else, if there is some counter with a zero count, it is allocated to the new item, and the counter set to 1. A grouping argument is used to argue that any item which occurs more than *n*/*k* times must be stored by the algorithm when it terminates where n represents the votes of n processors (total of access count). The items that are stored by the algorithm are the frequent items (queries).

Algorithm 3. Counter-Based Algorithm

Input: Details of data items, no of distinct items(k).

Output: List of frequently accessed data items

```
01: Set n as 0 and T as $\Phi$;
02: FOR each Data Item $D_i$;
03: n=n+1;
04: IF ($D_i \in T$)
03: Ci=Ci+1;
04: ELSE IF ( |T| < K ) THEN
05: $T=T \cup D_i$;
06: Ci=1;
07: ENDIF
08: ENDIF
09: ENDFOR
```

PERFORMANCE EVALUATION

All the algorithms discussed in the paper have implemented in Java using Sun's j2re 1.6.0_20. The experiments were conducted on a system with an Intel(R) Core(TM) i3 3.07 GHz processor and 2 GB main memory running Windows 7 Enterprise K operating system.

- **Experimental Settings**

A mondial document (XML Data Repository 2012) is used as a real data set in the experiment. Table 1 shows features of real data set used in the experiment. Wireless XML streaming, Lineage Encoding method and finding frequent items were implemented in the experiment. The wireless streams generated consist of the index segment followed by textual XML data. In the experiment, time taken to process user's queries was measured.

Table 1: Data Set Used in the Experiment

Data Set	Elements	Attributes	Text	Sizes	Max Depth
Mondial	22,323	47,423	7467	1.7MB	5

- **Experimental Results**

Table 2 shows features of streams generated by the other wireless XML streaming methods (Bruno et al. 2002; Jun Pyo Park et al. 2013: Kaushik et al. 2004; Park et al. 2009). When wireless XML stream generated with Lineage

Encoding (LE) is compared with other XML streaming methods, LE eliminates redundant attribute names, and the size of stream generated by LE is further reduced. The size of stream generated by LE is the smallest even though LE contains many indices including location paths, Lineage Codes, CIs, AIs, and TIs. The generation costs do not affect the access time and tuning time though generation time for LE is larger than PS and S-node approaches because the broadcast server only disseminates the pre-generated XML stream.

Table 2: Mondial Data Stream Generated by Wireless Streaming Methods (Park et al. 2010)

	TS	EC	SD	DIX	LE
# of Tag Names	22,423	22,423	22,423	22,423	33
# of Attribute Names	47,423	47,423	47,423	47,423	47,423
# of Attribute Values	47,423	47,423	47,423	47,423	47,423
# of texts	7,647	7,647	7,647	7,647	7,647
Size of the Indexes	2,693,219	3,192,163	358,768	269,076	40,631
Generation Time	203 ms	401 ms	499 ms	1,226 ms	967 ms

Figure 7 shows access time evaluation results on the real XML data set. The access time is decided by two factors: 1) the size of data stream, and 2) a correct prediction ensuring early termination of query processing. As shown in the table, Lineage Encoding exhibits the best performance because it generates the smallest data stream by eliminating redundant tag names and attribute names, and terminates query processing quickly compared to other wireless XML stream methods because approaches such as S-node (Kaushik et al. 2004) and DIX approaches (Park et al. 2009) explore the entire stream to find desired data dispersed over the stream and the sizes of indices are significantly larger as a result access times are significantly larger than the others as in the case of TS and EC.

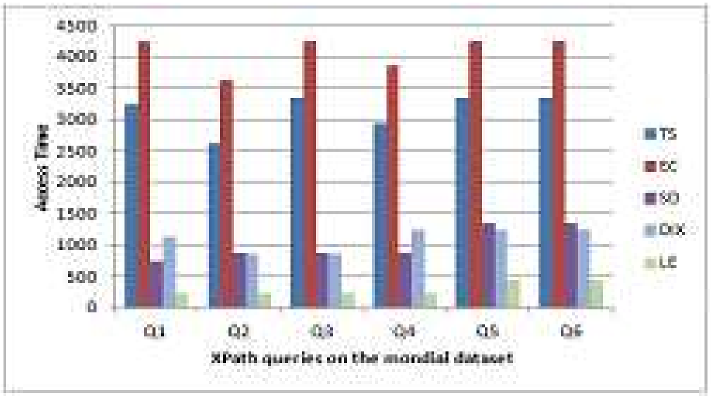


Figure 7: Access Time Evaluation on the Mondial Data Set

CONCLUSIONS

In this paper, in addition to the energy and latency efficient wireless XML streaming scheme that supports queries in the wireless environment, frequent queries are determined. The contribution of counter based algorithm plays a vital role in identifying frequent queries. The frequency threshold considered did not affect update throughput and it is faster. The space used by the algorithm at the finest accuracy level was less than 1MB and the cost was 100 times less. This range of sizes is small enough to fit within a second level cache.

In this paper, the proposed framework process the twig pattern query for the mobile client by retrieving the required data satisfying the given predicates by performing bit-wise operations on the Lineage Codes in the relevant G-nodes. In the future, we plan to analyze the issues that were not addressed in this paper. First, as network failures may occur in wireless broadcasting environment as communication is unstable, the indexing mechanism should take into account network failures. Second, issues related to security were not considered.

REFERENCES

1. Acharya, S., Alonso, R., Franklin, M., and Zdonik, S. (1995). *Broadcast Disks: Data Management for Asymmetric Communication Environments. Proc. ACM SIGMOD Int'l Conf. Management of Data Conference.* Pp. 199-210.
2. Al-Khalifa, S., Jagadish, H. V., Koudas, M., Patel, J. M., Srivastava, D., and Wu, Y. (2002). *Structural Joins: A Primitive for Efficient XML Query Pattern Matching. Proc. Int'l Conf. Data Eng. (ICDE).* Pp. 141-152.
3. Berglund, A., Boag, S., Chamberlin, D., Fernandez, M. F., Kay, M., Robie, J., and J. Simeon. (2002). *XML Path Language (XPath) 2.0.* Technical Report W3C.
4. Bruno, N., Srivastava, D., and Koudas, N. (2002). Holistic Twig Joins: Optimal XML Pattern Matching," Proc. ACM SIGMOD International Conference on Management of Data. Pp. 310-321.
5. Chen, Y., Davidson, S. B., and Zheng, Y. (2006). *An effective Xpath query processor for xml streams in* ICDE. Pp.79-85.
6. Chung, Y. D., Yoo, S., and Kim, M. H. (2010). *Energy- and Latency- Efficient Processing of Full- Text Searches on a Wireless Broadcast Stream. IEEE Trans. Knowledge and Data Engineering.* Pp. 207-218.
7. Cormode. G., and Hadjieleftheriou, M. (2010). *Finding Frequent Items in Data Streams. The VLDB Journal.* Pp. 3-20.
8. Imielinski, T., Viswanathan, S., and Badrinath, B. (1997). *Data on Air: Organization and Access. IEEE Trans. Knowledge and Data Engineering.* Pp. 353-372.
9. Jun Pyo Park., Chang-Sup Park., and Yon Dohn Chung. (2013). *Lineage Encoding- An Efficient Wireless XML Streaming Supporting Twig Pattern Queries. IEEE Transactions on Knowledge and Data Engineering.*
10. Kaushik, R., Krishnamurthy, R., Jeffrey F Naughton., and Ramakrishnan, R. (2004). *On the Integration of Structure Indexes and Inverted Lists. 20th International Conference on Data Engineering (ICDE'04).* Pp. 829-836.
11. Park, C. S., Kim, C. S., and Chung, Y. D. (2005). *Efficient Stream Organization for Wireless Broadcasting of Xml Data. Proc. International Conference of Asian Computing Science.* Pp. 223-235.
12. Park, J. P., Park, C. S., and Y.D. Chung. (2009). *Attribute Summarization: A Technique for Wireless XML Streaming. Proc. International Conference on Interaction Sciences.* Pp. 492-496.
13. Park, J. P., Park, C. S., and Y.D. Chung. (2010). *Energy and Latency Efficient Access of Wireless XML Stream. Journal on Database Management.* Pp. 58-79.
14. Park, S. H., Choi, J. H., and Lee, S. (2006). An Effective, Efficient XML Data Broadcasting Method in Mobile Wireless Network. Proc. 17th International Conference on Database and Expert Systems Applications (DEXA). Pp. 358-367.

15. SAX (Simple API for XML), http://www.saxproject.org, 2004.

16. Tatarinov, I., Viglas, S., Beyer, K., Shanmugasundaram, J., Shekita, E., and Zhang., C. (2002). Storing and Querying Ordered XML Using a Relational Database System. Proceedings of ACM SIGMOD Conference. Pp. 204-215.

17. Wang, W., Wang, H., Lu, H., Jiang, H., Lin, X., and Li, J. (2005). *Efficient processing of xml path queries using the disk-based f&b index," in VLDB.* Pp. 145–156.

18. XML Data Repository, http://www.cs.washington.edu/ research/xmldata sets, 2012.