JOURNAL OF INFORMATION SCIENCE AND ENGINEERING 33, 159-182 (2017)
DOI: 10.6688/JISE.2017.33.1.10

An Efficient Index and Data Distribution Scheme for XML Data Broadcast in Mobile Wireless Networks

MOHAMMAD JAVANI[1] AND MEGHDAD MIRABI[2,*]
[1]*Department of Computer Engineering*
University of Kashan, Kashan, Iran
[2]*Department of Computer Engineering*
Islamic Azad University, South Tehran Branch, Tehran, Iran
E-mail: Javani125@yahoo.com; meghdad.mirabi@gmail.com

In mobile wireless networks, a scalable way to disseminate XML data among many mobile clients is broadcasting of XML data. In these networks, a broadcast server disseminates XML data through a broadcast channel and mobile clients access the XML stream using energy-restricted portable devices. The main problem of XML data broadcast is that XML data can be accessed sequentially, and mobile clients have to wait for their desired data until they appear over the broadcast channel. By indexing the XML stream, mobile clients can efficiently decide when and where their desired data are available over the broadcast channel. The main question in indexing an XML stream is that how to mix the index with the XML data in such a way that mobile clients consume minimum time and energy during the process of XML queries. In this paper, we propose a new distribution scheme to place both the index information and XML data over a wireless broadcast channel by optimally partitioning the partial and relevant parts of both the indexes and XML data and then distributing them over a broadcast channel. Although the proposed distribution scheme increases the size of XML stream but the experimental results show that it does not degrade the performance of XML query processing of mobile clients.

Keywords: data distribution, mobile wireless networks, wireless broadcast channel, XML query processing, XML stream

1. INTRODUCTION

Typically, there are two types of entities in mobile wireless broadcast networks: mobile clients and broadcast server. In these networks, a broadcast server periodically disseminates data through a wireless broadcast channel and mobile clients independently tune in to the channel through mobile devices and process their queries by scanning the broadcast stream [1-4].

Generally, data broadcast allows mobile clients retrieve their desired data over a broadcast channel with minimum energy consumption since a large number of mobile clients can be served simultaneously with no additional energy consumption for sending requests to the broadcast server [5-8]. However, the main problem of data broadcast in mobile wireless networks is that data can be accessed sequentially, and mobile clients have to wait for their desired data until they appear over the broadcast channel [9, 10].

Building an index for broadcast data helps mobile clients in deciding when and where their desired data will be available over the broadcast channel [1-4]. By exploiting

Received August 26, 2015; revised February 23, 2016; accepted July 4, 2016.
Communicated by Hee Kap Ahn.
* Corresponding author.

mode) until their desired data are arrived over the channel. One crucial aspect of indexing the broadcast data is that how to mix the index with the data over the broadcast channel in such a way that mobile clients consume minimum time and energy to retrieve their desired data.

A common metric to estimate the cost of data access over a wireless broadcast channel is access time [1-4]. It is the interval between the times that a mobile client submits its query on the air to the time that the mobile client receives the query results from the broadcast channel. Also, a common metric to estimate the total amount of energy consumption of a mobile client to access the desired data over a broadcast channel is tuning time [1-4]. It is the total time that a mobile client remains in the active mode to get the desired data over the broadcast channel. As much as a mobile client remains in the active mode to get the desired data, it consumes more amounts of energy.

With the growing popularity of XML [11] as a standard for data dissemination over the Internet, the use of XML for data broadcasting over wireless broadcast channels has obtained a lot of attentions [12-22]. In recent years, several XML indexing methods have been proposed to selectively access XML data over an XML stream in a broadcast channel such as the indexing method proposed by [23-25]. These indexing methods are based on the structural information between XML nodes which is placed in the wireless XML stream and XML query processing at a mobile client is performed by exploiting the structural information in the index to reach the desired data. In these indexing methods, if a mobile client tunes in to the broadcast channel in the middle of a broadcast cycle, it should wait until the next broadcast cycle begins and this waiting time results in a long access time. We refer to this as the Delayed Query Processing Problem [13]. We describe this problem for the indexing method proposed by [25] in the following example.

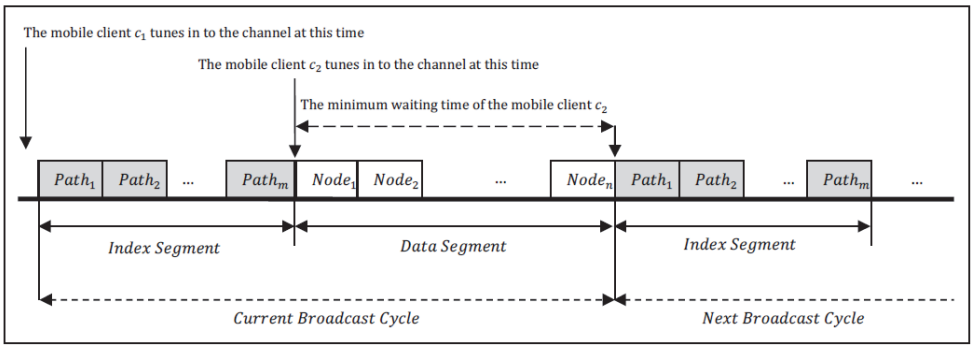


Fig. 1. General structure of the XML stream over the broadcast channel.

Example 1: Fig. 1 shows the general structure of an XML stream in a wireless broadcast channel. As shown in Fig. 1, the XML stream is divided into two segments: Index Segment and Data Segment. The index segment contains the structural information of XML nodes in the XML document (*i.e.* a set of root-to-node paths) which takes the role of index, whereas the data segment contains the contents of XML nodes in the XML document. Now, assume that the mobile client c_1 tunes in to the broadcast channel before the first index is broadcasted, while the mobile clients c_2 tunes in to the broadcast channel after the last index has been passed over. The mobile client c_1 can start XML query pro-

cessing within the current broadcast cycle but the mobile client c_2 have to wait until the next broadcast cycle begins since the index information is disseminated over the broadcast channel before the mobile client c_2 tunes in to the channel. This waiting time will increase access time.

To solve this problem, the partial and relevant parts of the index segment (*i.e.* the relevant paths) and the data segment (*i.e.* the relevant XML nodes) must be distributed in the current broadcast cycle in such a way that mobile clients can process XML queries without exploring the XML stream at the beginning of the next broadcast cycle. By exploiting this fact, we propose a new index and data distribution scheme for the XML indexing method proposed in [25] by dividing each broadcast cycle to a set of quantums. Each quantum contains a set of paths and XML nodes. Hence, the main contributions of this paper are summarized as follows:

- We propose a new distribution scheme for the proposed XML indexing method in [25] by distributing the partial and relevant parts of both the indexes and the XML data into suitable positions over the broadcast channel.
- We provide algorithms for generating a wireless XML stream and processing different types of XML queries over the broadcast channel based on our proposed distribution scheme.
- We evaluate the performance of our proposed distribution scheme in processing different types of XML queries by performing several experiments using different XML data sets.

The remainder of this paper is organized as follows: in Section 2, the background related to this study is described. In Section 3, our proposed XML index and data distribution scheme is presented. In Section 4, the process of XML querying at mobile clients based on our proposed distribution scheme is explained. In Section 5, the experimental results in processing different types of XML queries based on our proposed distribution scheme are presented. Finally, in Section 6, the paper is concluded with a conclusion and discussion of future works.

2. XML DATA MODEL AND XML QUERIES

Generally, an XML document can be modeled by a tree structure. In this tree structure, elements are represented by nodes and Parent-Child (P-C) relationships between the elements are represented by edges. Fig. 3 shows the XML tree corresponding to the ML document of Fig. 2.

```
<SigmodRecord>
    <issue>
        <volume>11</volume>
        <number>1</number>
        <articles>
            <article>
                <title>Architecture of Future Data Base Systems</title>
                <authors>
                    <author position="00">Lawrence A. Rowe</author>
```

Fig. 2. An example of XML document.

```
                    <author position="01">Michael Stonebraker</author>
                </authors>
                <initPage>30</initPage>
                <endPage>44</endPage>
            </article>
            <article>
                <title>Multisafe – A Data Security Architecture</title>
                <authors>
                    <author position="00">Robert P. Trueblood</author>
                    <author position="01">H. Rex Hartson</author>
                </authors>
                <initPage>45</initPage>
                <endPage>63</endPage>
            </article>
        </articles>
    </issue>
    <issue>
        <volume>12</volume>
        <number>2</number>
        <articles>
            <article>
                <title>Comparison and Mapping of the Relational and CO
    DASYL Data Models</title>
                <authors>
                    <author position="00">Gary H. Sockut</author>
                </authors>
                <initPage>55</initPage>
                <endPage>68</endPage>
            </article>
        </articles>
    </issue>
</SigmodRecord>
```

Fig. 2. (Cont'd) An example of XML document.

In this paper, we use XPath [26] as the query language. The results of an XPath query are derived based on the location path. A location path consists of location path steps. Processing the location path step will identify the set of XML nodes in the XML tree which satisfies the axis, node sets, and predicates in the location path step.

Definition 1: Let "/" denotes a child axis, "//" denotes a descendant axis, and "*" denotes a wildcard in an XPath expression. Let $XP^{[/,//,*]}$ denotes a fragment of XPath corresponding to XPath expressions that involve only "/", "//", and "*". A simple path XML query is a fragment of $XP^{\{/,//,*\}}$ that may involve child axis ("/"), descendant axis ("//"), and wildcard ("*").

For example, the simple path XML query Q = "/*SigmodRecord//authors/author*" finds authors for all of the articles published by the Sigmod Record journal.

Definition 2: Let "/" denotes a child axis, "//" denotes a descendant axis, "*" denotes a wildcard, and "[]" denotes a branching predicate in an XPath expression. Let $XP^{\{/,//,*,[]\}}$ denotes a fragment of XPath corresponding to XPath expressions that involve only "/", "//", "*", and "[]". A twig pattern XML query is a fragment of $XP^{\{/,//,*,[]\}}$ that involves child axis ("/"), descendant axis ("//"), wildcard ("*"), and branching predicate ("[]").

For example, the twig pattern XML query Q = "/*SigmodRecord/issue*[*volume/text*() = "11"]//title" finds titles for all of the articles published in the volume 11 of the Sigmord Record journal.

3. OUR PROPOSED DISTRIBUTION SCHEME

In our proposed distribution scheme, a broadcast cycle is divided into N parts called Quantum. Fig. 4 shows a broadcast cycle with four quantums (*i.e.* $N = 4$). Fig. 5 shows the general structure of the jth quantum in a broadcast cycle in our proposed distribution scheme. As shown in Fig. 5, a quantum is divided into three segments: Metadata Segment, Index Segment, and Data Segment. In the following, we explain the structures of these three segments in each quantum.

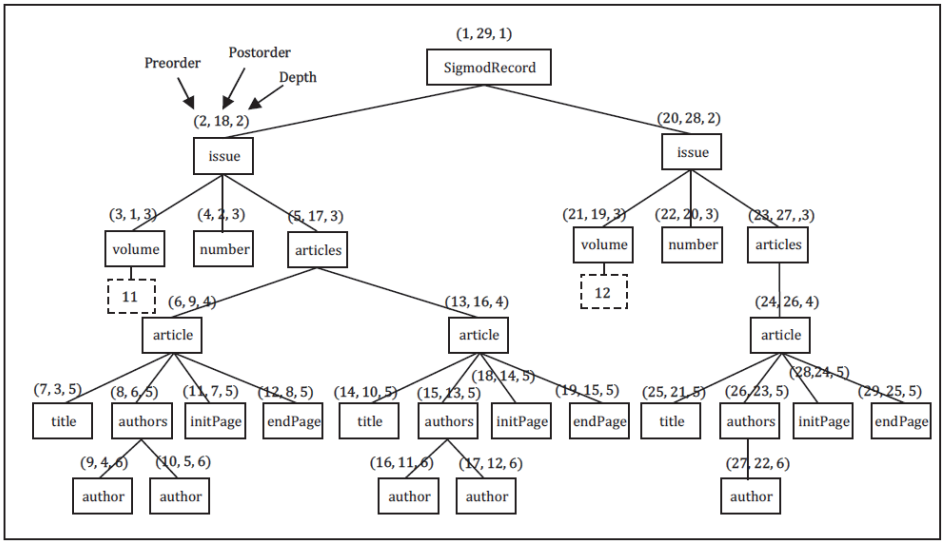


Fig. 3. An example of XML tree.

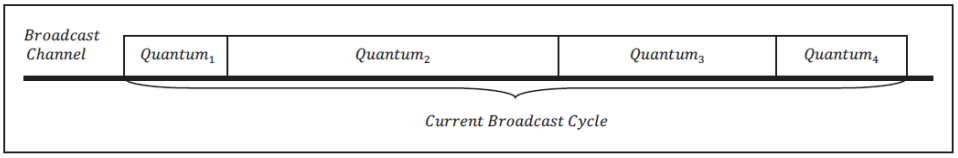


Fig. 4. General structure of a broadcast cycle.

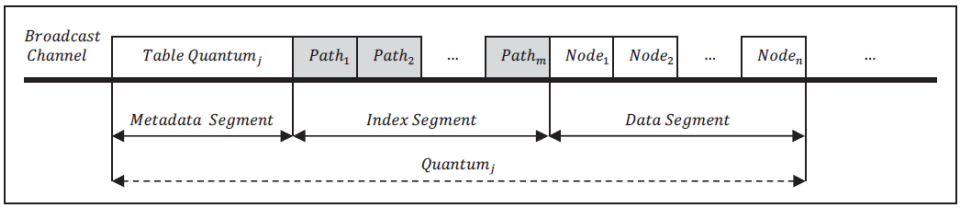


Fig. 5. General structure of the Quantum$_j$ in a broadcast cycle.

Definition 3: The root-to-node path of a node *e* in an XML tree *T* is a sequence of node names (or tag names) from the root node *r* to the node *e* which are separated by "/".

For example, the root-to-node path of the node "*article*" with the preorder 24 in the XML tree illustrated in Fig. 3 is the path "/*SigmodRecord/issue/articles/article*".

By grouping the set of XML nodes having the same root-to-node path, a set of unique root-to-node paths called path summary (PS) can be constructed.

Definition 4: The path summary of an XML tree (PS_T) is an ordered set of unique root-to-node paths in the XML tree *T* where the paths are ordered based on the breadth first traverse of the XML tree *T*.

For example, the path summary of the XML tree illustrated in Fig. 3 is shown in Table 1.

Table 1. An example of path summary.

Ordered ID	Path Name	Root-to-Node Path
1	$Path_1$	/SigmodRecord
2	$Path_2$	/SigmodRecord/issue
3	$Path_3$	/SigmodRecord/issue/volume
4	$Path_4$	/SigmodRecord/issue/number
5	$Path_5$	/SigmodRecord/issue/articles
6	$Path_6$	/SigmodRecord/issue/articles/article
7	$Path_7$	/SigmodRecord/issue/articles/article/title
8	$Path_8$	/SigmodRecord/issue/articles/article/authors
9	$Path_9$	/SigmodRecord/issue/articles/article/initPage
10	$Path_{10}$	/SigmodRecord/issue/articles/article/endPage
11	$Path_{11}$	/SigmodRecord/issue/articles/article/authors/author

As shown in Fig. 5, the metadata segment of a quantum contains a table quantum. The general structure of the table quantum in the metadata segment of the *j*th quantum of a broadcast cycle is illustrated in Fig. 6 (a).

Definition 5: Assume that $PS_j = \{Path_1, Path_2, \ldots, Path_m\}$ be the subset of the root-to-node paths in the path summary of the XML tree *T* which is placed in the *j*th quantum. The general structure of the Table Quantum in the metadata segment of the *j*th quantum contains the following fields:

- Length of $Quantum_j$: This field indicates the length of the *j*th quantum (in bytes) in a broadcast cycle except the length of the Table Quantum (*i.e. the length of Index Segment + the length of Data Segment*).
- Number of Paths: This field indicates the total number of the root-to-node paths in the path summary of the XML tree *T* which are placed in the *j*th quantum (*i.e. m*).
- Node Information: It is used when the field Number of Paths is not equal to "0". It contains a set of pairs with the following two fields:
  - Length of Last Node Name: $\forall 1 \le i \le m$, this field indicates the total length of the last node name in the *i*th path.
  - Last Node Name: $\forall 1 \le i \le m$, this field contains the last XML node name in the *i*th path.

It should be noted that two fields Length of Last Node Name and Last Node Name are repeated for each path placed in the *j*th quantum.

For example, Fig. 6 (b) shows the structure of the table quantum in the metadata segment of the first quantum in the case that there is four quantums in a broadcast cycle.

It should be mentioned that the field Number of Paths can be equal to "0" in the table quantum of the metadata segment of the *j*th quantum. It means that the *j*th quantum has only XML data. The general structure of the *j*th quantum in a broadcast cycle in this case is illustrated in Fig. 7.

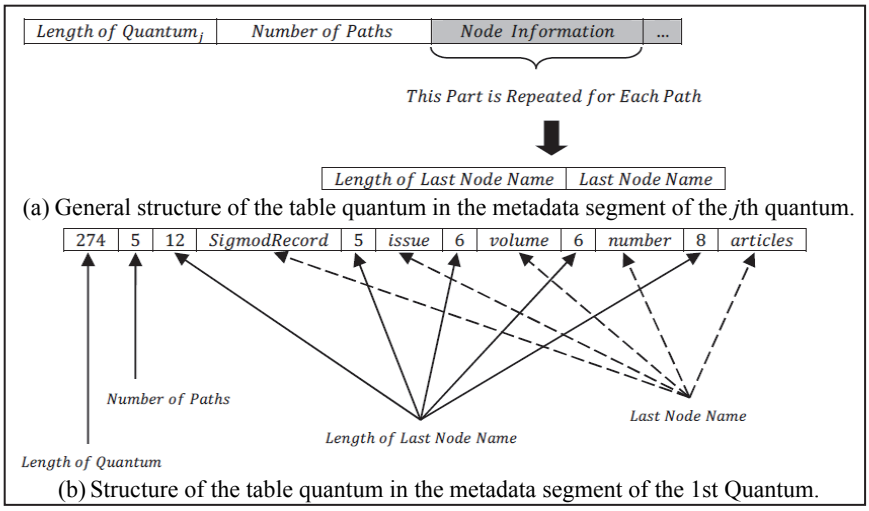


Fig. 6. Structure of the table quantum in the metadata segment.

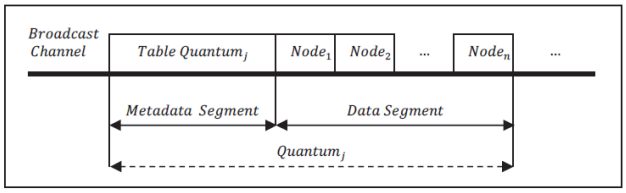


Fig. 7. General structure of the Quantum$_j$ in a broadcast cycle.

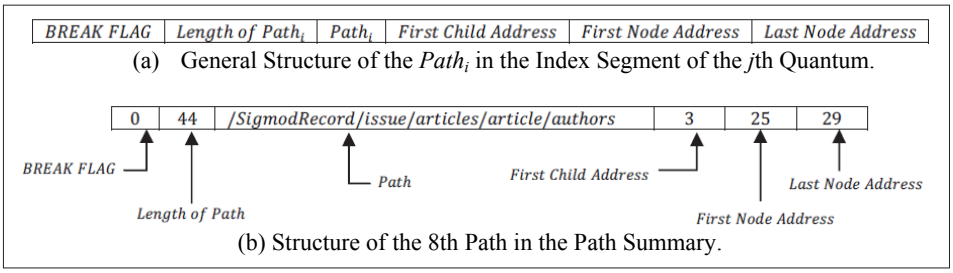
As shown in Fig. 5, the index segment of a quantum is a sequence of ordered unique root-to-node paths. The general structure of the *i*th root to-node path ($Path_1$) in the index segment of the *j*th quantum is illustrated in Fig. 8 (a).

Definition 6: Assume that $PS_j = \{Path_1, Path_2, \ldots, Path_m\}$ be the subset of the root-to-node paths in the path summary of the XML tree *T* which is placed in the *j*th quantum. The general structure of the $Path_i$, $\forall 1 \le i \le m$, in the index segment of the *j*th quantum contains the following fields:

- BREAK FLAG: This field indicates whether the data segment related to this index segment is broken or not. The default value of this flag is "0" which means the data segment is not broken.

- Length of $Path_i$: This field indicates the total length of the ith root-to-node path in the PS_j.
- $Path_i$: This field contains the ith root-to-node path in the PS_j.
- First Child Address: This field contains the distance between the $Path_i$ and the $Path_k$ in the index segment of the jth quantum where the $Path_k$ is the first child path of the $Path_i$.
- First Node Address: This field contains the address (*i.e.* arrival time) of the first XML node having the root-to-node path $Path_i$ in the data segment related to this index segment.
- Last Node Address: This field contains the address (*i.e.* arrival time) of the last XML node having the root-to-node path $Path_i$ in the data segment related to this index segment.

For example, Fig. 8 (b) shows the structure of the path with the ordered ID 8 in the path summary shown in Table 1. The First Child Address of the path with the ordered ID 8 in the path summary is the path with the ordered ID 11 and therefore, the distance between these two paths is 3. The two fields First Node Address and Last Node Address are filled based on the order of the XML nodes in the XML stream.



(a) General Structure of the $Path_i$ in the Index Segment of the jth Quantum.

(b) Structure of the 8th Path in the Path Summary.

Fig. 8. Structure of the $Path_i$ in the index segment.

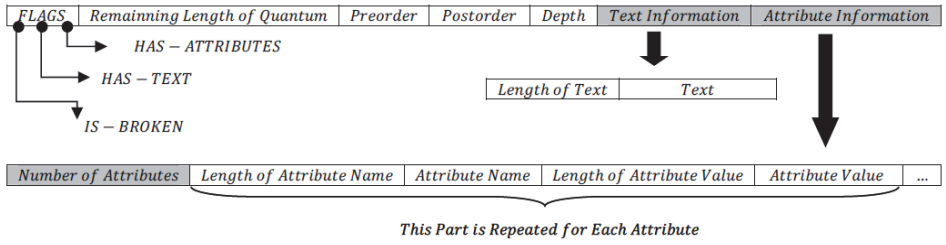
As shown in Fig. 5, the data segment of a quantum is a sequence of XML nodes. The general structure of the ith node ($Node_i$) in the data segment of the jth quantum is illustrated in Fig. 9 (a).

Definition 7: Assume that $Nodes_j = \{Node_1, Node_2, \ldots, Node_n\}$ be the subset of XML nodes in the XML tree T which is placed in the jth quantum. The general structure of the $Node_i$, $\forall 1 \le i \le n$, in the data segment of the jth quantum contains the following fields:

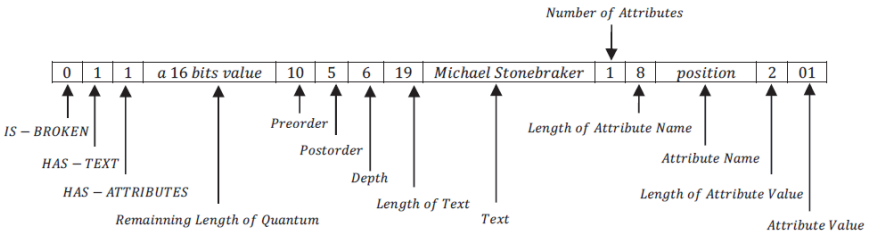
- FLAGS: This field contains three flags that are IS-BROKEN, HAS-TEXT and HAS-ATTRIBUTES. The flag IS-BROKEN indicates whether this data segment is broken or not. The default value of this flag is "0" which means the data segment is not broken. Two flags HAS-TEXT and HAS-ATTRIBUTES indicate whether the ith XML node ($Node_i$) contains text and/or attributes or not. The default value of these two flags is "0" which means the ith XML node ($Node_i$) does not have text and attributes.
- Remaining Length of Quantum: A 16-bits value which indicates the remaining length of the current quantum (*i.e.* the jth quantum).
- Preorder: This field contains the preorder of the ith XML node ($Node_i$) in the XML tree T when the XML tree T is traversed in the preorder sequence.
- Postorder: This field contains the postorder of the ith XML node ($Node_i$) in the XML tree T when the XML tree T is traversed in the postorder sequence.

- Depth: This field contains the depth of the ith XML node ($Node_i$) in the XML tree T.
- Text Information: This part is used when the ith XML node ($Node_i$) in the XML tree T has text. In this case, the flag HAS-TEXT will be set to "1". The Text Information contains two fields:
  - Length of Text: This field indicates the total length of the text content of the ith XML node ($Node_i$) in the XML tree T.
  - Text: This field contains the text content of the ith XML node ($Node_i$) in the XML tree T.
- Attribute Information: This part is used when the ith XML node ($Node_i$) in the XML tree T has at least one attribute. In this case, the flag HAS-ATTRIBUTE will be set to "1". The Attribute Information contains the following fields:
  - Number of Attributes: This field indicates the total number of attributes of the ith XML node ($Node_i$) in the XML tree T.
  - Length of Attribute Name: This field indicates the total length of the attribute name.
  - Attribute Name: This field contains the name of attribute.
  - Length of Attribute Value: This field indicates the total length of the attribute value.
  - Attribute Value: This field contains the value of attributes.

Note that the four fields Length of Attribute Name, Attribute Name, Length of Attribute Value, and Attribute Value are repeated for each attribute of the ith XML node ($Node_i$) in the XML tree T.



(a) General structure of the $Node_i$ in the data segment of the jth quantum.



(b) Structure of node "*author*" with the Preorder "10".

Fig. 9. Structure of the $Node_i$ in the data segment.

For example, Fig. 9 (b) shows the structure of the node "*author*" with the preorder 10 in the XML tree illustrated in Fig. 3. Note that the text information and attribute information of this node are represented in Fig. 2. In our proposed structure for the XML

stream, each XML node in the data segment of a quantum contains its preorder, postorder, and depth information. This information is used to process twig pattern XML queries which will be explained in Section 4.

In order to process twig pattern XML queries, we need to determine the Parent-Child (P-C) and Ancestor-Descendant (A-D) relationships between the XML nodes in the XML tree. By labeling the XML nodes in the XML tree with the pre/post labeling scheme [25, 27], the structural relationships between the XML nodes can be easily determined. In this labeling scheme, each XML node is assigned with three values, preorder, postorder, and depth where preorder and postorder are the ordinal decimal numbers of the XML nodes in the preorder and postorder traversal sequences, respectively and the depth is the depth of the XML node in the XML tree. An XML tree labeled by the pre/post labeling scheme is illustrated in Fig. 3.

Fig. 10 shows the order of information in the first quantum when the broadcast cycle is divided into four quantums.

Metadata Segment:	274	5	12	*SigmodRecord*	5	*issue*	6	*volume*	6	*number*	8	*articles*
Index Segment:	$Path_1$			$Path_2$		$Path_3$			$Path_4$		$Path_5$	
Data Segment:	*SigmodRecord*	*issue*		*issue*	*volume*		*volume*	*number*		*number*	*articles*	*articles*

Fig. 10. Order of information in the first quantum.

As shown in Fig. 10, only five root-to-node paths (*i.e.* $Path_1$, $Path_2$, $Path_3$, $Path_4$ and $Path_5$) are placed in the index segment of the first quantum. Therefore, the data segment of this quantum contains the information of XML nodes of these paths. As shown in Fig. 3, the root-to-node path $Path_1$ (*i.e.* "/*SigmodRecord*") contains only one XML node "*SigmodRecord*", the root-to-node path $Path_2$ (*i.e.* "/*SigmodRecord*/*issue*") contains two XML nodes "*issue*", the root-to-node path $Path_3$ (*i.e.* "/*SigmodRecord*/*issue*/*volume*") contains two XML nodes "*volume*", the root-to-node path $Path_4$ (*i.e.* "/*SigmodRecord*/*issue*/*number*") contains two XML nodes "*number*", and the root-to-node path $Path_5$ (*i.e.* "/*SigmodRecord*/*issue*/*articles*") contains two XML nodes "*articles*". Therefore, the information of these nine XML nodes is placed in the data segment of the first quantum.

Definition 8: Assume that $PS_T = \{Path_1, Path_2, \ldots, Path_p\}$ be the set of root-to-node paths in the path summary of the XML tree T and $Nodes_T = \{Node_1, Node_2, \ldots, Node_q\}$ be the set of XML nodes in the XML tree T. The initial size of a quantum (*i.e.* $|InitialQuantum|$) is defined as follows:

$$\left|InitialQuantum\right| = \left\lceil \frac{\sum_{i=1}^{p} |Path_i| + \sum_{i=1}^{q} |Node_i|}{N} \right\rceil. \quad (1)$$

Where $\forall 1 \le i \le p$, $|Path_i|$ is the size of root-to-node path $Path_i$, $\forall 1 \le i \le q$, $|Node_i|$ is the size of XML node $Node_i$, and N is the total number of quantums in a broadcast cycle.

It should be noted that the sizes of both the paths and the nodes are calculated based on their structures which are defined in Definition 6 and Definition 7, respectively.

Example 2: Assume that $\forall 1 \le i \le p$ and $\forall 1 \le j \le q$, $\sum_{i=1}^{p}|Path_i| + \sum_{j=1}^{q}|Node_i| = 1170$

bytes and $N = 4$. Therefore, $|InitialQuantum| = 292.5$ *bytes* based on the Eq. (1). In this case, we assume that the initial size of all the quantums except the last quantum is equal to 292 bytes and the initial size of the last quantum is equal to 294 bytes since the size of a quantum cannot be a floating point value.

In our proposed distribution scheme, the initial sizes of all the quantums in a broadcast cycle are calculated based on the Eq. (1) but their actual sizes may be changed during the process of generating the XML stream.

Fig. 11 shows the XMLStreamGeneration algorithm which is designed to generate an XML stream from an XML tree based on our proposed distribution scheme. In the XMLStreamGeneration algorithm, the input is an XML tree *T* and the output is a broadcast XML stream *XS*. In the algorithm, the three variables *currentPath*, *currentNode*, *currentQuantum* are used to store the current root-to-node path in the path summary of the XML tree *T*, the current XML node with the minimum preorder having the root-to-node path *currentPath*, and the current quantum in a broadcast cycle, respectively (Lines 1-3). At the beginning of the algorithm, the XML stream *XS* is empty (Line 4). It should be noted that the XMLStreamGeneration algorithm only generates the XML stream *XS* for a broadcast cycle since the XML stream is the same in each broadcast cycle. In the XMLStreamGeneration algorithm, two temporary lists *Paths* and *Nodes* are used to store the lists of root-to-node paths and XML nodes, respectively. Before starting the process of generating the XML stream for the quantum *currentQuantum*, these two lists are empty (Lines 7-8). In our proposed distribution scheme, it is assumed that paths in the index segment of a quantum cannot to be broken. However, in the case that the current quantum does not have enough space to store all of the XML nodes of the root-to-node path *currentPath*, some XML nodes having the root-to-node path *currentPath* will be distributed in the next quantum. Therefore, a boolean variable named *BreakPathFlag* is used to demonstrate that whether there is an XML node having the root-to-node path *currentPath* in the data segment of the next quantum or not. The default value of this variable is false (Line 14). If the quantum *currentQuantum* has enough space to place the root-to-node path *currentPath* (Line 15), the path *currentPath* is added to the list *Paths* (Line 16).

XMLStreamGeneration Algorithm
Input: The path summary *PS* from the XML tree *T*, All of the XML nodes in the XML tree *T*
Output: A broadcast XML stream *XS*

1. *currentPath* ←∅; //*currentPath* is a temporary data structure to store the current root-to-node path in the path summary *PS*
2. *currentNode* ←∅; //*currentNode* is a temporary data structure to store the current XML node with the minimum preorder having the root-to-node path *currentPath*
3. *currentQuantum* ←∅; //*currentQuantum* is a temporary data structure to store the information of the current quantum in the current broadcast cycle
4. *XS* ←∅;
5. **while (** there is a quantum in the broadcast cycle**) {**
6. *currentQuantum* ← the next quantum in the broadcast cycle;
7. *Paths* ←∅; //*Paths* is a temporary list to store a set of root-to-node paths
8. *Nodes* ←∅; //*Nodes* is a temporary list to store a set of XML nodes
9. **If (**there are some XML nodes from the previous quantum**) {**
10. *Nodes* ← the remaining XML nodes of the previous quantum;
11. **}**//end if
while (there is still a root-to-node path in the path summary *PS* **) {**

Fig. 11. XMLStreamGeneration algorithm.

```
12.          currentPath ← the current root-to-node path in the path summary PS;
13.          BreakPathFlag ← false;
14.          if (the quantum currentQuantum has enough space to place the root-to-node path cur-
             rentPath) {
15.               Paths ← Paths ∪ {currentPath};
16.               if (the quantum currentQuantum  has enough space to place all of the XML nodes of
                  the root-to-node path currentPath) {
17.                    Nodes ← All of the XML nodes of the root-to-node path currentPath;
18.               } else {
19.                    BreakPathFlag ← true;
20.                    while (there is still an XML node having the root-to-node path currentPath) {
21.                         currentNode ← the current XML node having the root-to-node path cur-
                            rentPath;
22.                         BreakNodeFlag ← false;
23.                         if (the quantum currentQuantum has enough space to place the XML
                            node currentNode) {
24.                              Nodes ← Nodes ∪ {currentNode};
25.                         } else {
26.                           BreakNodeFlag ← true;
27.                           Break the XML node currentNode and store some parts of the XML
                              node currentNode in the Quantum currentQuantum and keep the re-
                              maining parts of the XML node currentNode to be stored in the next
                              quantum;
28.                           Break;
29.                         }//end if
30.                    }//end while
31.                    currentPath ← the next root-to-node path in the path summary PS;
32.                    Break;
33.               }//end if
34.          } else {
35.               If (the quantum currentQuantum has some unused spaces) {
36.                    Reduce the size of the quantum currentQuantum based on the unused spaces
                       and add the reduced size to the size of the next quantum;
37.               }//end if
38.               Braek;
39.          }//end if
40.     }//end while
41.     Generate the table quantum currentTableQuantum for the quantum currentQuantum based on
        the root-to-node paths in the list Paths and the XML nodes in the list Nodes;
42.     Place the table quantum currentTableQunatum in the quantum currentQuantum;
43.     Generate the index segment of the quantum currentQunatum based on the root-to-node paths of
        the list Paths;
44.     Place the index segment in the quantum currentQuantum;
45.     Generate the data segment of the quantum currentQunatum based on the XML nodes of the list Nodes;
46.     Place the data segment in the quantum currentQuantum;
47.     XS ← XS ∪ {currentQunatum};
48. }//end while
```

Fig. 11. (Cont'd) XMLStreamGeneration algorithm.

However, if the quantum *currentQuantum* does not have enough space to place the root-to-node path *currentPath*, the size of quantum *currentQuantum* is reduced and the unused space of the quantum *currentQuantum* is added to the size of the next quantum in the broadcast cycle (Lines 35-40). If the quantum *currentQuantum* has enough space to place all the XML nodes of the root-to-node path *currentPath* (Line 17), all the XML nodes of the root-to-node path *currentPath* are added to the list *Nodes* (Line 18). In the case that the quantum *currentQuantum* does not have enough space to place all the XML nodes of

the root-to-node path *currentPath* (Lines 19-34), the XML node *currentNode* is added to the list *Nodes* if the quantum *currentQuantum* has enough space to place the XML node *currentNode* (Lines 24-25). In the case that the quantum *currentQuantum* does not have enough space to place the XML node *currentNode* the XML node *currentNode* is broken and some parts of the XML node *currentNode* will be stored in the next quantum (Lines 26-30). The boolean variable *BreakNodeFlag* is used to demonstrate that whether the XML node *currentNode* is broken or not (Line 27). The XMLStreamGeneration algorithm generates the table quantum *currentTableQuantum* based on the root-to-node paths in the list *Paths* and the XML nodes in the list *Nodes* and then places it in the quantum *currentQuantum* (Lines 42-43). It also generates the index segment for the quantum *currentQuantum* based on the root-to-node paths in the list *Paths* and places it in the quantum *currentQuantum* (Lines 44-45). Then, the XMLStreamGeneration algorithm generates the data segment of the quantum *currentQuantum* based on the XML nodes in the list *Nodes* and places it in the quantum *currentQuantum* (Lines 46-47). Finally the quantum *currentQuantum* is added to the XML stream *XS* (Line 48). This process will be repeated for each quantum in the broadcast cycle in order to generate the final XML stream *XS*.

4. XML QUERY PROCESSING OVER A WIRELESS BROADCAST CHANNEL

To process simple path XML queries based on our proposed distribution scheme, the SimplePathXMLQueryProcessor algorithm is devised as shown in Fig. 12. Given a simple path XML query *Q*, the mobile client *c* tunes in to the broadcast channel and reads the current quantum (*i.e. currentQuantum*) from the current broadcast cycle (Line 3). Then, it obtains the information of the table quantum from the quantum *currentQuantum* (Line 4). In the case that the last node name in the simple path XML query *Q* is equal to one of the node names in the table quantum *tableQuantum* (Line 5), there is this probability that this quantum (*i.e. currentQunatum*) contains the result of the simple path XML query *Q*. Therefore, the algorithm searches to find a path similar to the simple path XML quary *Q* (Lines 5-14). When the path is found (Line 15), the mobile client *c* checks the flag BREAK FLAG in the index segment of the quantum *currentQuantum* to determine that whether the data segment related to this path is broken or not. In the case that the value of the flag BREAK FLAG is false (Line 16), the mobile client *c* uses the field *First Node Address* of the path *currentPath* to jump forward to the first XML node satisfying the simple path XML query *Q* in the data segment of the quantum *currentQunatum* and download all the candidate XML nodes from the data segment of the quantum *currentQunatum* (Lines 17-23). If the simple path XML query *Q* contains predicate conditions at the text content and/or attribute vales, the algorithm checks the text content and/or attribute values and adds the node *currentNode* into the XML query results set *R* if the predicate conditions of the simple path XML query *Q* are satisfied (Lines 20-22). If the value of the flag BREAK FLAG is true, the mobile client *c* has to switch to doze mode until the XML node with the address *First Node Address* arrives on the air (Line 25). In the case that the value of the flag IS-BROKEN is true (Line 28), the mobile client *c* has to download the broken part of the node *currentNode* from the next quantum (Line 29). After downloading the content of node *currentNode*, the algorithm

checks the text content and/or attribute values of the node *currentNode* and adds it into the XML query results set *R* if the predicate conditions of the simple path XML query *Q* are satisfied (Lines 31-33). In the case that the last node name in the simple path XML query *Q* is not equal to the node names in the table quantum *tableQuantum*, the mobile client *c* has to switch to doze mode until the next quantum arrives on the air (Lines 39-41).

Example 3: Assume that the XML document illustrated in Fig. 2 is streamed and disseminated over a broadcast channel as shown in Fig. 13. Now, assume that the mobile client *c* submits the simple path XML query *Q* = "/*SigmodRecord*/*issue*/*articles*/*article*/*title*" on the air. The mobile client *c* first tunes in to the broadcast channel and downloads the table quantum of the first quantum. It then searches to find the node name "*title*" in the first table quantum since this node name is the last node name in the simple path XML query *Q*. Since the node name "*title*" is not in the table quantum of the first quantum, the mobile client *c* switches to doze mode until the second quantum appears on the air. When the second quantum appears on the air, the mobile client *c* switches to active mode and downloads the table quantum of the second quantum. It then searches to find the node name "*title*" in the table quantum of the second quantum. The mobile client *c* finds the node name "*title*" in this table quantum. Therefore, there is this probability that the path similar to the simple path XML query *Q* is available in this quantum. Hence, the mobile client *c* downloads the paths in the index segment one after another and finds the path similar to the simple path XML query *Q*.

SimplePathXMLQueryProcessor Algorithm
Input: XML Stream *XS*, Simple Path XML Query *Q*
Output: XML Query Results Set *R*

```
1. R←∅;
2.   while (the XML Stream XS is not ended) {
3.     currentQuantum ← the next quantum in the current broadcast cycle;
4.     tableQuantum ← the table quantum of the quantum currentQuantum;
5.     if (the last node name in the simple path XML query Q == one of the node names in the table
       quantum tableQuantum) {
6.         currentPath←∅;
7.         pathIsFound ← false;
8.         while (the index segment of the quantum currentQuantum is not ended){
9.             currentPath ← the current path from the index segment of the quantum cur-
               rentQuantum;
10.            if (the path currentPath == the XML Query Q) {
11.                pathIsFound ← true;
12.                break;
13.            }//end if
14.        }//end while
15.        if (the flag pathIsFound == true) {
16.            if (the flag BREAK FLAG in the path currentPath == false) {
17.                Wait in doze mode until the XML node with the address First Node Ad-
                   dress arrives on the air;
18.                while (there is a candidate XML node in the data segment of the quantum
                   currentQuantum) {
19.                    currentNode ← the current node from the data segment of the
                       quantum currentQuantum;
20.                    if (the node currentNode satisfies the predicate condition in the
```

Fig. 12. SimplePathXMLQueryProcessor algorithm.

```
                                XML query Q) {
21.                                 R ← R ∪ {currentNode};
22.                             }//end if
23.                         }//end while
24.                 } else {
25.                         Wait in doze mode until the XML node with the address First Node Ad-
                            dress arrives on the air;
26.                         while (there is a candidate XML node in the data segment of the quantum
                            currentQuantum) {
27.                             currentNode ← the current node from the data segment of the
                                quantum currentQuantum;
28.                             if (the flag IS-BROKEN in the node currentNode ==true) {
                                    currentNode ←currentNode ∪ Retrieve the broken part of
            the node currentNode from the next quantum;
31.                             }//end if
32.                             if (the node currentNode satisfies the predicate condition in the
                                XML query Q) {
33.                                 R ← R ∪{currentNode};
34.                             }//end if
35.                         }//end while
36.             } //end if
37.         } else {
38.             Wait in doze mode until the next quantum arrives on the air;
39.         } //end if
40.     } else {
41.         Wait in doze mode until the next quantum arrives on the air;
42.     }//end if
43. }//end while
```

Fig. 12. (Cont'd) SimplePathXMLQueryProcessor algorithm.

The First Quantum in the Broadcast Cycle:

Metadata Segment:	274	5	12	*SigmodRecord*	5	*issue*	6	*volume*	6	*number*	8	*articles*
Index Segment:	$Path_1$			$Path_2$		$Path_3$		$Path_4$		$Path_5$		
Data Segment:	*SigmodRecord*		*issue*	*issue*	*volume*	*volume*	*number*	*number*	*articles*	*articles*		

The Second Quantum in the Broadcast Cycle:

Metadata Segment:	310	2	7	*article*	5	*title*
Index Segment:	$Path_6$				$Path_7$	
Data Segment:	*article*	*article*	*article*		*title*	*title*	*title*

Assume that the last node in the second quantum (*i.e.* node "*title*") is broken and some parts of this node are placed in the third quantum.

The Third Quantum in the Broadcast Cycle:

Metadata Segment:	292	3	7	*authors*	8	*initPage*	7	*endPage*		
Index Segment:	$Path_8$				$Path_9$			$Path_{10}$		
Data Segment:	*title*	*authors*	*authors*	*authors*	*initPage*	*initPage*	*initPage*	*endpage*	*endpage*	*endpage*

Assume that the last node in the third quantum (*i.e.* node "*endPage*") is broken and some parts of this node are placed in the fourth quantum.

The Forth Quantum in the Broadcast Cycle:

Metadata Segment:	294	1	6	*authors*			
Index Segment:	$Path_{11}$						
Data Segment:	*endpage*		*authors*	*authors*	*authors*	*authors*	*authors*

Fig. 13. An example of a broadcast cycle with four quantum.

By exploiting the field BREAK FLAG in the path which is similar to the simple path XML query *Q*, the mobile client *c* finds out that the data segment related to this path is broken. It also finds out the times that the candidate XML nodes are arrived on the air by exploiting the two fields *First Node Address* and *Last Node Address* in the path. The mobile client *c* switches to doze mode to conserve its battery power until the first candidate node at the time equals to the *First Node Address* appears on the air. When the first candidate node is arrived, the mobile client *c* switches to active mode and downloads the set of candidate XML nodes. When the content of the third candidate XML node is downloaded from the broadcast channel, mobile client *c* finds out that this node is broken. Therefore, the mobile client continues to download the broken part of the third candidate XML node from the next quantum until all parts of the content of the third candidate node are retrieved.

In order to save the length of the paper, we omit to explain the process of XML querying in other type of XML queries. Refer to the proposed XML indexing method in [25] order to have more information.

5. PERFORMANCE EVALUATION

In this section, we evaluate the performance of our proposed distribution scheme in processing different types of XML queries by performing several experiments. All the experiments were conducted on a system with the Interl(R) Core(TM) i7 Q720 @1.60 GHz processor and 3GB RAM running Windows 7 Ultimate where the server and client modules were implemented in Java.

5.1 Experimental Setting

We logically modeled the wireless stream as a binary file, where the broadcast server writes a byte stream on the file and the mobile clients read the file and process the XML queries.

In our simulation model, we assumed that the broadcast bandwidth is fully utilized for XML data broadcasting. To measure the access time and tuning time, we considered only the activity of a mobile client since the activity of a mobile client does not affect the XML query processing performance at the other mobile clients.

We assumed that the XML stream is broadcasted and accessed in units with a fixed size (*i.e.* buckets) and thus we measured the access time and tuning time in processing different types of XML queries by the number of buckets. A bucket is the smallest logical unit in a wireless broadcast channel. In the view of assumption that the network speed is fixed, the number of buckets can be converted into time since the elapsed time for reading a bucket is computed as the bucket size divided by the network speed [4]. To measure the performance variation based on the number of buckets, we used three different bucket sizes, 64, 128, and 256 bytes in our experiments. However, we only present the experimental results for the cases that the bucket size is set to 128 bytes since the experimental results are not dependent on the bucket size.

To measure the performance variation based on the types of XML data sets, we used two XML data sets. Table 2 shows the characteristics of the XML data sets used in

our experiments.

To measure the performance variation based on the types of XML queries, we used different types of XPath queries. The list of XPath queries used in our experiments is shown in Table 3.

Table 2. XML data sets.

Data Set	Size (KB)	Number of Elements	Max Depth	Max Fan Out
Shakespeare	1,061	25,339	7	162
Mondial	1,743	22,423	5	955

Table 3. XPath query sets of the different XML data sets.

XML Data Set	Query Name	XPath Expression
Shakespeare	SH1	/PLAYS/PLAY/ACT
	SH2	/PLAYS/PLAY/PERSONAE/PGROUP
	SH3	/PLAYS/PLAY/ACT/EPILOGUE/SPEECH/SPEAKER
	SH4	/PLAYS/PLAY/ACT/EPILOGUE/TITLE[text()="EPILOGUE"]
	SH5	/PLAYS/PLAY/ACT/SCENE/SPEECH/STAGEDIR[@SPEAKER="BERTRAM"]
	SH6	/PLAYS/*/ACT[SCENE/SPEECH/SPEAKER/text()="MARK ANTONY"]
	SH7	/PLAYS/*/ACT[SCENE/SPEECH/SPEAKER/text()="MARK ANTONY"]/*/TITLE
	SH8	/PLAYS/*/ACT[SCENE/SPEECH/SPEAKER/text()="MARK ANTONY"]/*//LINE
Mondial	MO1	/mondial/country/religions
	MO2	/mondial/country/city/name
	MO3	/mondial/country/province/city/population
	MO4	/mondial/country/name[text()="France"]
	MO5	/mondial/country/province/city[@country="f0_418"]
	MO6	/mondial/country/*/city[population]
	MO7	/mondial/country/*/city[population]/name
	MO8	/mondial/country/*/city[population]//name

Table 4. Initial quantum sizes of the different XML data sets.

Data Set	Initial Quantum Size		
Shakespeare	196	388	31501
Mondial	492	966	28648

In our experiments, we only applied our proposed distribution scheme to the XML indexing method proposed in [25] since our proposed distribution scheme is proposed for this XML indexing method. We implemented our proposed distribution scheme by dividing each broadcast cycle to a set of quantums. To measure the performance variation based on the size of quantums in a broadcast cycle, we used three different initial quantum sizes for each XML data set. The list of initial quantum sizes for each XML data set in our experiments is shown in Table 4.

We implemented six different indexing methods that are OSA, TSA, SPA, DIX, C-DIX, and PS+Pre/Post. The OSA, TSA, and SPA methods are the wireless XML streaming methods in the S-Node approach proposed by [23]. The DIX and C-DIX methods are the distributed XML indexing methods proposed by [13]. The PS+Pre/Post method is the XML indexing method proposed in [25]. It should be noted that the five indexing methods OSA, TSA, SPA, DIX, and C-DIX cannot process the twig pattern XML queries

having wildcards, descendant axes, and predicate conditions.

The performance metrics used in our experiments were the access time ratio and the tuning time ratio. They are defined as follows:

Access Time Ratio = (Number of buckets to read in the XML stream from the moment a query is submitted to the moment the query results are retrieved) / (Total number of buckets in the XML stream)×100. (2)

Tuning Time Ratio = (Number of buckets to read in the XML stream when the mobile client is in the active mode) / (Total number of buckets in the XML Stream)×100. (3)

5.2 Experimental Results on Access Time

Figs. 14 and 15 show the ratio of access time in processing the different types of XML queries on the different XML data sets. As shown in Figs. 14 (a) and 15 (a), the ratio of access time in our proposed distribution scheme with the different initial quantum sizes is less than the ratio of access time in the OSA, TSA, SPA, DIX, C-DIX indexing methods for the XML queries with the query types 1, 2, 3, 4, and 5. It means that the performance of XML querying in our proposed distribution scheme is better than the performance of XML querying in the OSA, TSA, SPA, DIX, C-DIX indexing methods in terms of access time.

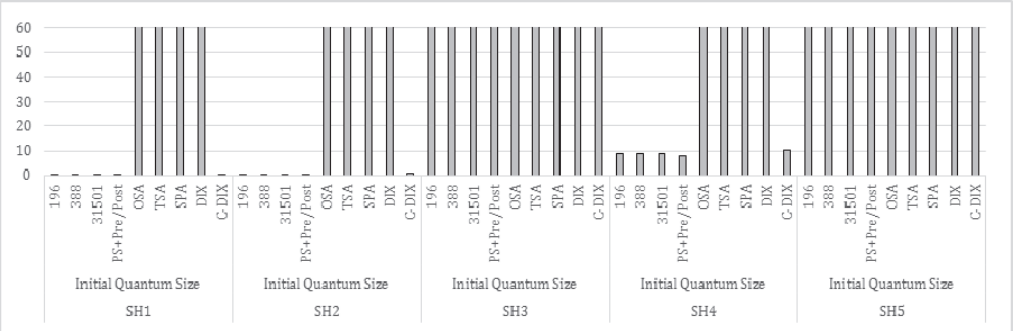


Fig. 14. (a) Access time ratio on the Shakespeare data set for the XML queries with the query types 1-5.

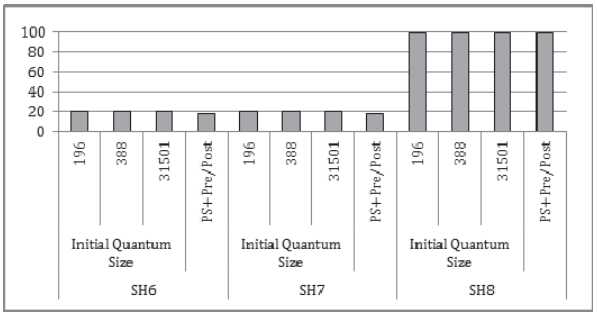


Fig. 14. (b) Access time ratio on the Shakespeare data set for the XML queries with the query types 6-8.

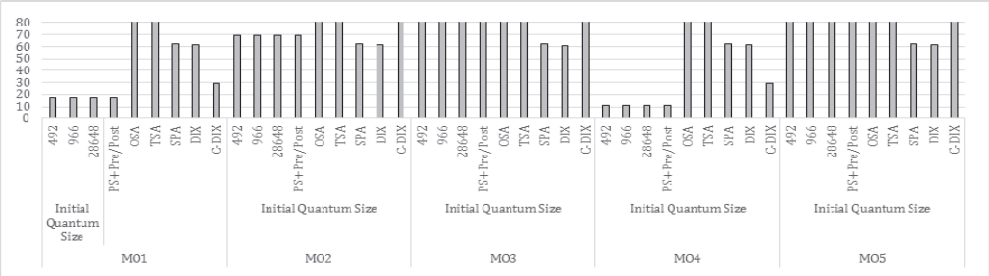


Fig. 15. (a) Access time ratio on the Mondial data set for the XML queries with the query types 1-5.

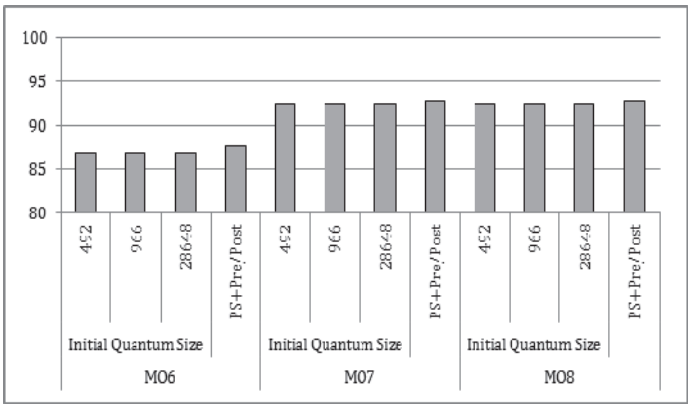


Fig. 15. (b) Access time ratio on the Mondial data set for the XML queries with the query types 6-8.

Also, as shown in Figs. 14 (a), (b) and 15 (a), (b), the ratio of access time in our proposed distribution scheme with the different initial quantum sizes is almost equal to the ratio of access time in the PS+Pre/Post indexing method. It means that:

1. The quantum size in a broadcast cycle does not affect the ratio of access time.
2. Our proposed distribution scheme does not degrade the performance of XML querying at the mobile clients in term of access time.

Although the performance of XML querying in terms of access time in our proposed distribution scheme and the PS+Pre/Post indexing method are almost equal but the index information in the PS+Pre/Post indexing method is disseminated at the first part of a broadcast cycle. It means that mobile clients which tune in to the broadcast channel after the index information is disseminated over the broadcast channel must wait until the next broadcast cycle beings. This waiting time increases the access time (Refer to Example 1). However, in our proposed distribution scheme, the partial and relevant parts of the index information and the XML data are distributed in the current broadcast cycle in such a way that mobile clients can process XML queries without exploiting the XML stream at the beginning of the next broadcast cycle. This is the advantage of our proposed distribution scheme compared to the PS+Pre/Post indexing method.

5.3 Experimental Results on Tuning Time

Figs. 16 and 17 show the ratio of tuning time in processing the different types of XML queries on the different XML data sets. As shown in Figs. 16 (a) and 17 (a), the ratio of tuning time in our proposed distribution scheme with the different initial quantum sizes is less than the ratio of tuning time in the OSA, TSA, SPA, DIX, C-DIX indexing methods for the XML queries with the query types 1, 2, 3, 4, and 5. It means that the performance of XML querying in our proposed distribution scheme is better than the performance of XML querying in the OSA, TSA, SPA, DIX, C-DIX indexing methods in terms of tuning time.

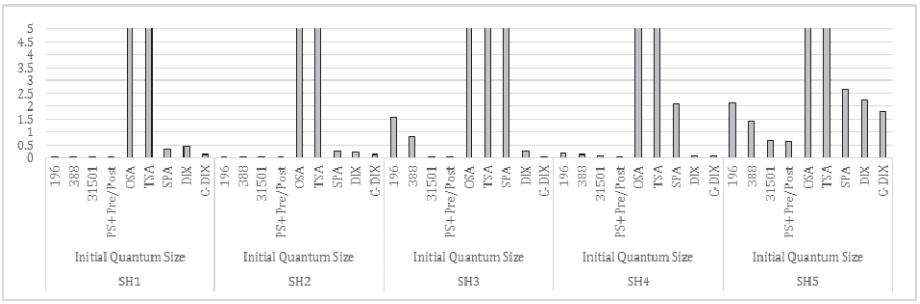


Fig. 16. (a) Tuning time ratio on the Shakespeare data set for the XML queries with the query types 1-5.

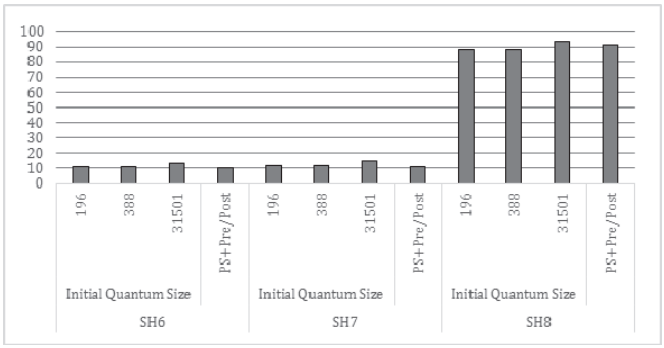


Fig. 16. (b) Tuning time ratio on the Shakespeare data set for the XML queries with the query types 6-8.

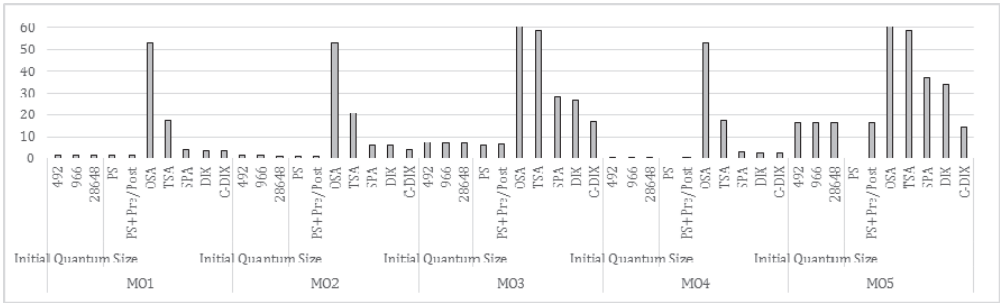


Fig. 17. (a) Tuning time ratio on the Mondial data set for the XML queries with the query types 1-5.

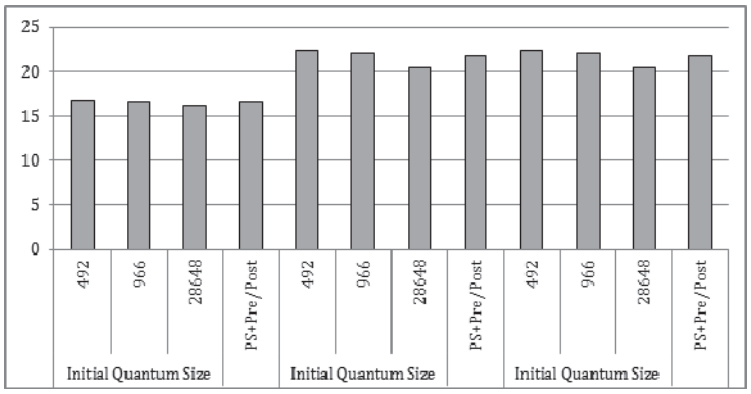


Fig. 17. (b) Tuning time ratio on the Mondial data set for the XML queries with the query types 6-8.

As shown in Figs. 16 (a), (b) and 17 (a), (b), the ratio of tuning time in our proposed distribution scheme with the different initial quantum sizes is almost equal to the ratio of tuning time in the PS+Pre/Post indexing method. It means that:

1. The quantum size in a broadcast cycle does not affect the ratio of tuning time.
2. Our proposed distribution scheme does not degrade the performance of XML querying at the mobile clients in term of tuning time.

Although the performance of XML querying in terms of tuning time in our proposed distribution scheme and the PS+Pre/Post indexing method are almost equal but the index information in the PS+Pre/Post indexing method is disseminated at the first part of a broadcast cycle. It means that mobile clients which tune in to the broadcast channel after the index information is disseminated over the broadcast channel must wait until the next broadcast cycle beings. This waiting time increases the access time (Refer to Example 1). However, in our proposed distribution scheme, the partial and relevant parts of the index information and the XML data are distributed in the current broadcast cycle in such a way that mobile clients can process XML queries without exploiting the XML stream at the beginning of the next broadcast cycle. This is the advantage of our proposed distribution scheme compared to the PS+Pre/Post indexing method.

7. CONCLUSION AND FUTURE WORKS

In this paper, we proposed a new XML index and data distribution scheme for the XML indexing method proposed in [25] by partially partitioning both the XML indexes and data and then distributing them into suitable positions over a broadcast channel. We also devised an algorithm to generate a wireless XML stream based on our proposed distribution scheme. In addition, we devised algorithms to process different types of XML queries over a broadcast channel based on our proposed distribution scheme. By performing several experiment using different XML data sets, we demonstrated that our proposed distribution scheme does not degrade the performance of XML querying at the mobile clients in terms of access time and tuning time.

In the future, we intend to investigate other issues which are not considered in this paper. First, we attempt to propose a caching strategy for mobile clients in order to improve the performance of XML query processing in terms of access time. Second, we attempt to propose a novel XML data placement scheme over multiple wireless broadcast channels in the case that the broadcast channels have different bandwidths and error rates.

REFERENCES

1. T. Imieĺinski and B. R. Badrinath, "Data management for mobile computing," *SIGMOD Record*, Vol. 22, 1993, pp. 34-39.
2. T. Imielinski, S. Viswanathan, and B. R. Badrinath, "Energy efficient indexing on air," in *Proceedings of ACM SIGMOD International Conference on Management of Data* 1994, pp. 25-36.
3. S. Acharya, R. Alonso, M. Franklin, and S. Zdonik, "Broadcast disks: Data management for asymmetric communication environments," in *Proceedings of ACM SIGMOD International Conference on Management of Data*, 1995, pp. 199-210.
4. T. Imielinski, S. Viswanathan, and B. R. Badrinath, "Data on air: Organization and access," *IEEE Transactions on Knowledge and Data Engineering*, Vol. 9, 1997, pp. 353-372.
5. Y. D. Chung and M. H. Kim, "An index replication scheme for wireless data broadcasting," *Journal of Systems and Software*, Vol. 51, 2000, pp. 191-199.
6. Y. D. Chung and M. H. Kim, "Effective data placement for wireless broadcast," *Distributed and Parallel Databases*, Vol. 9, 2001, pp. 133-150.
7. Y. D. Chung, S. H. Bang, and M. H. Kim, "An efficient broadcast data clustering method for multipoint queries in wireless information systems," *Journal of Systems and Software*, Vol. 64, 2002, pp. 173-181.
8. M.-S. Chen, K.-L. Wu, and P. S. Yu, "Optimizing index allocation for sequential data broadcasting in wireless mobile computing," *IEEE Transactions on Knowledge and Data Engineering*, Vol. 15, 2003, pp. 161-173.
9. C.-C. Lee and Y. Leu, "Efficient data broadcast schemes for mobile computing environments with data missing," *Information Sciences*, Vol. 172, 2005, pp. 335-359.
10. Y. D. Chung, S. Yoo, and M. H. Kim, "Energy and latency efficient processing of full-text searches on a wireless broadcast stream," *IEEE Transactions on Knowledge and Data Engineering*, Vol. 22, 2010, pp. 207-218.
11. T. Bray, J. Paoli, C. M. Sperberg-McQueen, E. Maler, and F. Yergeau, "Extensible markup language (XML) 1.0 (5th ed.) W3C recommendation," http://www.w3.org/TR/REC-xml/, 2008.
12. Y. D. Chung and J. Y. Lee, "An indexing method for wireless broadcast XML data," *Information Sciences*, Vol. 177, 2007, pp. 1931-1953.
13. J. P. Park, C.-S. Park, and Y. D. Chung, "Energy and latency efficient access of wireless XML stream," *Journal of Database Management*, Vol. 21, 2010, pp. 58-79.
14. J. P. Park, C.-S. Park, and Y. D. Chung, "Lineage encoding: An efficient wireless XML streaming supporting twig pattern queries," *IEEE Transactions on Knowledge and Data Engineering*, Vol. 25, 2012, pp. 1559-1573.

15. Y. Qin, W. Sun, Z. Zhang, and P. Yu, "An efficient document-split algorithm for on-demand XML data broadcast scheduling," in *Proceedings of the IET Conference on Wireless*, *Mobile and Sensor Networks*, 2007, pp. 759-762.
16. W. Sun, Y. Qin, P. Yu, and Z. Zhang, "On-demand XML data broadcast in wireless computing environments," in *Proceedings of the 3rd International Conference on Wireless Communications*, *Networking and Mobile Computing*, 2007, pp. 3035-3038.
17. Y. Qin, W. Sun, Z. Zhang, P. Yu, and Z. He, "Query-grouping based scheduling algorithm for on-demand XML data broadcast," in *Proceedings of the 4th International Conference on Wireless Communications, Networking and Mobile Computing*, 2008, pp. 1-4.
18. J. P. Park, C.-S. Park, M. K. Sung, and Y. D. Chung, "Attribute summarization: A technique for wireless XML streaming," in *Proceedings of the 2nd International Conference on Interaction Sciences: Information Technology*, *Culture and Human*, 2009, pp. 492-496.
19. Y. Qin, W. Sun, Z. Zhang, P. Yu, Z. He, and W. Chen, "A novel air index scheme for twig queries in on-demand XML data broadcast," in *Proceedings of the 20th International Conference on Database and Expert Systems Applications*, LNCS 5690, 2009, pp. 412-426.
20. W. Sun, P. Yu, Y. Qing, Z. Zhang, and B. Zheng, "Two-tier air indexing for on-demand XML data broadcast," in *Proceedings of the 29th IEEE International Conference on Distributed Computing Systems*, 2009, pp. 199-206.
21. Y. Qin, H. Wang, and L. Sun, "Cluster-based scheduling algorithm for periodic XML data broadcast in wireless environments," in *Proceedings of the 25th IEEE International Conference on Advanced Information Networking and Applications Workshop*, 2011, pp. 855-860.
22. J. Wu, P. Liu, L. Gan, Y. Qin, and W. Sun, "Energy-conserving fragment methods for skewed XML data access in push-based broadcast," in *Proceedings of the 12th International Conference on Web-Age Information Management*, 2011, pp. 590-601.
23. C.-S. Park, C. S. Kim, and Y. D. Chung, "Efficient stream organization for wireless broadcasting of XML data," in *Proceedings of the 10th Asian Computing Science Conference on Advances in Computer Science: Data Management on the Web*, LNCS 3818, 2005, pp. 223-235.
24. S.-H. Park, J.-H. Choi, and S. Lee, "An effective, efficient XML data broadcasting method in a mobile wireless network," in *Proceedings of the 17th International Conference on Database and Expert Systems Applications*, LNCS 4080, 2006, pp. 358-367.
25. M. Mirabi, H. Ibrahim, and L. Fathi, "PS+Pre/Post: A novel structure and access mechanism for wireless XML stream supporting twig pattern queries," *Pervasive and Mobile Computing*, Vol. 2013, http://dx.doi.org/10.1016/j.pmcj.2013.09.009.
26. A. Berglund, S. Boag, D. Chamberlin, M. F. Fernández, M. Kay, J. Robie, and J. Siméon, "XML path language (XPath) 2.0 (2nd ed.)," http://www.w3.org/TR/xpath-20/, 2010.
27. M. Mirabi, H. Ibrahim, N. I. Udzir, and A. Mamat, "An encoding scheme based on fractional number for querying and updating XML data," *Journal of Systems and Software*, Vol. 85, 2012, pp. 1831-1851.

Mohammad Javani obtained his Master degree in Computer Engineering from the University of Kashan, Iran in 2016. His research interests include formal methods, communication networks, and mobile computing.

Meghdad Mirabi obtained his Master and Ph.D. degrees in Computer Science from the Universiti Putra Malaysia (UPM), Malaysia in 2009 and 2013, respectively. His research interests include data management in distributed (Mobile, P2P, Grid, and Cloud) computing and security in distributed computing.