An Efficient XML Data Placement Scheme over Multiple Wireless Broadcast Channels

HOSSEIN MIRABI¹, MEGHDAD MIRABI²+ AND ALI BORJIAN BOROUJENI³

¹Department of Computer Engineering
University of Kashan
Kashan, Iran

²Department of Computer Engineering
Islamic Azad University
South Tehran Branch, Tehran, Iran

³School of Computer Engineering
Iran University of Science and Technology
Tehran, Iran

E-mail: {hossein22mirabi; meghdad.mirabi; web.ebox}@gmail.com

In this paper, we propose an XML data placement scheme by optimally partitioning XML data among multiple broadcast channels in such a way that it minimizes the average response time of mobile clients in processing of different XML queries. By performing several experiments, we have shown that our proposed XML data placement scheme improves the performance of XML query processing of mobile clients in terms of access time without increasing the tuning time.

Keywords: data placement, multiple broadcast channels, mobile wireless networks, XML query processing, XML stream

1. INTRODUCTION

An efficient way to disseminate data in wireless networks is data broadcasting [1-4]. In wireless broadcast networks, a broadcast server periodically disseminates data through a wireless broadcast channel and mobile clients independently tune the channel and process their queries by scanning the broadcast stream [1, 4-6]. In this way, the task of query processing is entirely pushed to the client side; thus, the load of the broadcast server is independent from the population of mobile clients. Moreover, the wireless broadcast channel can be accessed by any number of mobile clients; hence, communication between mobile clients is avoided [7-9].

Nevertheless, the main problem with data broadcast is that the performance of query processing in mobile clients degrades as the number of data items being broadcast increases [10-12]. This is due to the sequential access of data broadcast items. The increasing number of data broadcast items causes mobile clients to wait for a substantial amount of time before receiving the desired data item(s). Consequently, the advantages of data broadcast are diminished.

A common metric to estimate the cost of data access in a mobile wireless broadcast environment is access time. It is the interval between the times that a mobile client submits its query on the air and the time that the mobile client receives the query results from the
As XML (eXtensible Markup Language) [13] is emerging as a standard for data dissemination over the Internet, the use of XML for data broadcasting in wireless networks is rapidly increasing. Recently, many applications are using XML for data broadcasting in wireless environments such as traffic and travel information systems and weather information systems [14].

In order to selectively access XML data over an XML stream in a broadcast channel, an indexing method intermixed with XML data is required, where this index indicates the exact location of the XML data (i.e., the arrival time of XML data) in the broadcast channel. Hence, mobile clients can read such information from the index and stay in the energy saving mode (doze mode) until the requested XML data are received on the air. In recent years, several XML indexing methods have been proposed to selectively access XML data over an XML stream in a broadcast channel [14-19]. However, these indexing methods are designed to be used only in a single broadcast channel. In this paper, we assume that there are multiple disjoint physical broadcast channels. This assumption has wider applicability since there are many frequencies on which XML data can be transmitted in a wireless network. In addition, the use of multiple wireless broadcast channels allows better configurability, scalability, and fault tolerance [10-12].

In this paper, we propose an efficient scheme to place XML data over multiple broadcast channels in a way that it reduces the access time (i.e., metric to measure the response time) of mobile clients in processing of different XML queries. Hence, the main contributions of the paper are summarized as follows:

- We propose a novel structure for streaming XML data over multiple wireless broadcast channels.
- We propose a new XML data placement scheme to place XML data over multiple wireless broadcast channels by partitioning and replicating XML data.
- We devise algorithms for processing simple path XML queries as well as twig pattern XML queries over XML streams using multiple wireless broadcast channels.
- We evaluate the performance of our proposed XML data placement scheme over multiple wireless broadcast channels in processing of different XML queries in terms of access time and tuning time by performing several experiments using different XML data sets.

2. BACKGROUND AND PROBLEM STATEMENT

In this section, we first define the XML data model with the syntax and semantic of XPath queries. Then, we explain the problem related to this study in detail.

2.1 XML Data Model and XPath 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. 2 shows the XML tree corresponding to the XML document of Fig. 1.

In this paper, we use XPath [20] 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 involve child axis (“/”), descendant axis (“//”), and wildcard (“*”).

For example, the simple path XML query $Q = \text{"//SimodRecord/articles/author"}$ finds all of the authors for 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 $Q = \text{"//SimodRecord//volume/text() = \"11\"/title"}$ finds all of the titles for the articles published in the volume 11 of the Sigmod Record journal.

![Fig. 1. An example of XML document.](image-url)
2.2 Problem Statement

To generate an XML stream over a single wireless broadcast channel, the contents of the XML data must be transformed from the original XML document into a suitable representation in such a way that the XML data in the new format contains index information to provide selectively access XML data over the XML stream for mobile clients.

Several XML indexing methods have been proposed to selectively access XML data over XML stream [14-19]. However, these XML indexing methods are designed to be used only in a single broadcast channel and cannot be applied to multiple wireless broadcast channels.

![Fig. 2. An example of XML tree.](image)

Generally, the process of generating XML stream from an XML document over a single broadcast channel can be categorized into two categories. In the first category [16, 19], an XML stream is organized into two sections. The first section contains the structural information of the XML document, which takes the role of an index, whereas the second section contains the contents of XML nodes in the XML document. Fig. 3 shows the general structure of XML stream in this category. As shown in Fig. 3, the structural information of the XML document is transmitted prior to the contents of XML nodes through the single wireless broadcast channel. The process of XML querying at mobile clients is very simple. Mobile clients listen to the wireless broadcast channel and read the structural information of the XML document and then switch to the doze mode (the energy saving mode) until the desired XML node appears over the channel. Then, they switch to the active mode to download the content of XML node.

![Fig. 3. General structure of the XML stream in the first category.](image)
In the second category [14, 15, 18], XML stream is organized into a set of unit structures called node (e.g. S-Node in [15], DIX Node in [14], G-Node in [18]). Each node in the XML stream contains both the structural information and content of one or more XML nodes in the XML document. Fig. 4 shows the general structure of XML stream in the second category. The process of XML querying at mobile clients is very simple. Mobile clients listen to the wireless broadcast channel and read the structural information of nodes and then retrieve the contents of XML nodes if they are desired XML nodes. The main problem of these methods is that the performance of XML querying in terms of access time degrades as the number of XML data being broadcast increases. It is due to the sequential access of XML data in the wireless broadcast channel. The increasing number of XML data causes mobile clients to wait for a substantial amount of time before receiving the desired XML data.

The aim of this paper is to propose an XML data placement scheme by optimally partitioning XML data among multiple wireless broadcast channels in order to reduce the average access time (i.e. metric to measure the response time) of mobile clients in processing of different XML queries.

![Fig. 4. General structure of the XML stream in the second category.](image)

**3. OUR PROPOSED XML DATA PLACEMENT SCHEME**

In this section, we first define our proposed structure for streaming XML data over multiple wireless broadcast channels and then, we explain how the contents of XML nodes are placed on different wireless broadcast channels.

**3.1 XML Stream Structure over Multiple Wireless Broadcast Channels**

In our proposed XML data placement scheme, we use a broadcast channel called Base Transmitter Station (BTS) to disseminate index information and one or more broadcast channels called Derived Transmitter Station (DTS) to disseminate the contents of XML nodes. Fig. 5 (a) shows the general structure of index information in the BTS. As shown in Fig. 5 (a), the index information contains a set of paths. In the following, we explain the unit structure of each path in the BTS.

**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 “author” with the preorder 9 in the XML tree illustrated in Fig. 2 is the path “/SigmodRecord/issue/articles/article/authors/author”.
By grouping the set of XML nodes having the same root-to-node path, a set of unique root-to-node paths can be constructed called path summary (PS).

**Definition 4:** The path summary of an XML tree $T$ ($PS_T$) 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. 2 is shown in Table 1.

<table>
<thead>
<tr>
<th>Ordered ID</th>
<th>Path</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>/SigmordRecord</td>
</tr>
<tr>
<td>2</td>
<td>/SigmordRecord/issue</td>
</tr>
<tr>
<td>3</td>
<td>/SigmordRecord/issue/volume</td>
</tr>
<tr>
<td>4</td>
<td>/SigmordRecord/issue/number</td>
</tr>
<tr>
<td>5</td>
<td>/SigmordRecord/issue/articles</td>
</tr>
<tr>
<td>6</td>
<td>/SigmordRecord/issue/articles/article</td>
</tr>
<tr>
<td>7</td>
<td>/SigmordRecord/issue/articles/article/title</td>
</tr>
<tr>
<td>8</td>
<td>/SigmordRecord/issue/articles/article/authors</td>
</tr>
<tr>
<td>9</td>
<td>/SigmordRecord/issue/articles/article/initPage</td>
</tr>
<tr>
<td>10</td>
<td>/SigmordRecord/issue/articles/article/endPage</td>
</tr>
<tr>
<td>11</td>
<td>/SigmordRecord/issue/articles/article/authors/author</td>
</tr>
</tbody>
</table>

The unit structure of the $i$th root to-node path ($Path_i$) of a path summary in the BTS is illustrated in Fig. 5 (b).

**Definition 5:** Assume that $PS_T = \{Path_1, Path_2, \ldots, Path_m\}$ be the path summary of an XML tree $T$. The unit structure of the $Path_i$, $\forall 1 \leq i \leq m$, in the BTS contains the following fields:

- **Length of Path:** This field indicates the total length of the $i$th root-to-node path in the $PS_T$.
- **Path:** This field contains the $i$th root-to-node path in the $PS_T$.
- **Channel Number:** This field indicates the channel number (i.e. DTS number) which the XML nodes having the root-to-node path $Path_i$ are disseminated over it.
• **Offset 1**: This field contains the address (i.e., arrival time) of the first byte of the first XML node having the root-to-node path $Path_i$ in the DTS with the number specified in the Channel Number field.

• **Offset 2**: This field contains the address (i.e., arrival time) of the last byte of the last XML node having the root-to-node path $Path_i$ in the DTS with the number specified in the Channel Number field.

For example, Fig. 5 (c) shows the unit structure of the root-to-node path with the ordered ID 2 in the path summary shown in Table 1. The Length of Path field is “19” since the length of the root-to-node path is “19” characters. The Path field is “/Sigmod-Record/issue” since the second root-to-node path in the path summary is that path. The Channel Number field is “2”. It means that all the XML nodes with the root-to-node path “/SigmodRecord/issue” are placed in the DTS with the number “2”. The fields Offset 1 and Offset 2 are “100” and “200”, respectively. It means that the arrival time of the first byte of the first XML node and the last byte of the last XML node having the root-to-node path “/SigmodRecord/issue” are “100” and “200”, respectively.

In our proposed XML data placement scheme, we use multiple wireless broadcast channels (i.e., DTSs) to disseminate the contents of XML nodes. Fig. 6 (a) shows the general structure of nodes in the case that we use two DTSs for disseminating the contents of XML nodes. The unit structure of the $i$th node ($Node_i$) in the DTS is illustrated in Fig. 6 (b).

**Definition 6**: Assume that $Nodes_T = \{Node_1, Node_2, ..., Node_n\}$ be the set of XML nodes in an XML tree $T$. The unit structure of the $Node_i$, $\forall 1 \leq i \leq n$, in the DTS(s) contains the following fields:

• **FLAGS**: This field contains two flags that are $HAS - TEXT$ and $HAS - ATTRIBUTES$ which indicate whether the $i$th XML node ($Node_i$) contains text and/or attributes or not. The default value of these two flags is “0” which means the $i$th XML node ($Node_i$) does not have text and attributes.

• **Preorder**: This field contains the preorder of the $i$th 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 $i$th 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 $i$th XML node ($Node_i$) in the XML tree $T$.

• **Text Information**: This part is used when the $i$th 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:
  o **Length of Text**: This field indicates the total length of the text content of the $i$th XML node ($Node_i$) in the XML tree $T$.
  o **Text**: This field contains the text content of the $i$th XML node ($Node_i$) in the XML tree $T$.

• **Attribute Information**: This part is used when the $i$th 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:
  o **Number of Attributes**: This field indicates the total number of attributes of the $i$th XML node ($Node_i$) in the XML tree $T$. 
o **Length of Attribute**: This field indicates the total length of the attribute name.

o **Attribute Name**: This field contains the name of attribute.

o **Length of Attribute Value**: This field indicates the total length of the attribute value.

o **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 *i*th XML node (Node$_i$) in the XML tree $T$.

For example, Fig. 6 (c) shows the unit structure of the node “author” with the pre-order “10” in the XML tree illustrated in Fig. 2. Note that the *text information* and *attribute information* of this node are represented in Fig. 1. In our proposed structure for the XML stream, each node in the DTS 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 [21, 22], 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. 2.

![Fig. 6. Structure of the nodes in the DTS(s).](image-url)
3.2 XML Data Placement over Multiple Wireless Broadcast Channels

In our proposed XML data placement scheme, the XML nodes have to be distributed across multiple DTSs in such a way that the workloads of DTSs be balanced. The following assumptions are used in our proposed XML data placement scheme:

Assumption 1: The total number of assigned DTSs for broadcasting the XML nodes is as follows: 1 \leq \text{Number of Assigned DTSs} \leq m, where m is the total number of root-to-node paths in the path summary.

Assumption 2: If the total number of DTSs assigned to disseminate the XML node is \( l \), \( \forall 1 \leq i, j \leq l \), the DTS \( i \) has higher priority to be used to disseminate the XML nodes than the \( DTS_j \) if and only if \( i < j \).

Assumption 3: \( \forall 1 \leq i, j \leq m \), the root-to-node path \( Path_i \) in the path summary has higher priority than the root-to-node path \( Path_j \) if and only if \( i < j \).

Assumption 4: \( \forall 1 \leq i \leq m \), all of the XML nodes of the root-to-node path \( Path_i \) in the path summary are placed only on a specified DTS.

Assumption 5: The order of XML nodes of the root-to-node path \( Path_i \) in a specified DTS is based on the preorder traversal of the XML nodes in the XML tree.

Assumption 6: If the total number of XML nodes of a specified root-to-node path (e.g., \( Path_i \)) is \( n \), the total size of the XML nodes of that path is \( \sum_{i=1}^{n} |Node_i| \) where \( |Node_i| \) is the size of \( i \)th XML node in bytes. It should be noted that the XML nodes may have different sizes and the size of each XML node is calculated based on its structure illustrated in Fig. 6 (b).

The process of assigning a DTS to disseminate all the XML nodes of a specified root-to-node path is based on the order of paths in the path summary and the total size of XML nodes in the paths. This process is done in such a way that the load balancing is provided across multiple DTSs.

Consider Table 2 as an example of the root-to-node paths in the path summary. Now, assume that we have only one DTS to disseminate all the XML nodes.

<table>
<thead>
<tr>
<th>Path</th>
<th>The Total Size of the XML Nodes in the Path</th>
<th>XML Nodes (XML Nodes are ordered based on their preorder)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Path1</td>
<td>50 Bytes</td>
<td>{Node11}</td>
</tr>
<tr>
<td>Path2</td>
<td>200 Bytes</td>
<td>{Node21, Node22, Node23, Node24, Node25}</td>
</tr>
<tr>
<td>Path3</td>
<td>150 Bytes</td>
<td>{Node31, Node32, Node33, Node34}</td>
</tr>
<tr>
<td>Path4</td>
<td>70 Bytes</td>
<td>{Node41, Node42, Node43, Node44, Node45, Node46}</td>
</tr>
<tr>
<td>Path5</td>
<td>80 Bytes</td>
<td>{Node51, Node52, Node53}</td>
</tr>
</tbody>
</table>
The process of XML data placement on a DTS is as follows: The XML nodes are placed on the DTD sequentially based on the priority of root-to-node paths in the path summary and their preorders. Fig. 7 shows the order of XML nodes on the DTS.

Fig. 7. XML data placement when the number of DTSs = 1.

Now, assume that we have two DTSs to disseminate the XML nodes. The process of XML data placement on two DTSs is as follows: in the Step 1, all of the XML nodes of the root-to-node path \( \text{Path}_1 \) are placed on the DTS1 and all of the XML nodes of the root-to-node path \( \text{Path}_2 \) are placed on the DTS2 since these two root-to-node paths have higher priority than other root-to-node paths in the path summary. As shown in Fig. 8 (a), the total number of bytes disseminated on the DTS1 is less than the total number of bytes disseminated on the DTS2 in this step (“50” < “200”). Therefore, in the Step 2, all the XML nodes of the root-to-node path \( \text{Path}_2 \) are placed on the DTS1. As shown in Fig. 8 (b), the total number of bytes disseminated on the DTS1 is equal to the total number of bytes disseminated on the DTS2 in this step (“200” = “200”). Therefore, in the Step 3, all the XML nodes of the root-to-node path \( \text{Path}_3 \) are placed on the DTS1 since the DTS1 has higher priority than the DTS2 (Refer to Assumption 2). As shown in Fig. 8 (c), the total number of bytes disseminated on the DTS2 is less than the total number of bytes disseminated on the DTS1 in this step (“270” > “200”). Therefore, in the Step 4, all the XML nodes of the root-to-node path \( \text{Path}_4 \) are placed on the DTS2. The result of XML data placement in the case that we have two DTSs is shown in Fig. 8 (d). By using this scheme to place the XML nodes on multiple wireless broadcast channels, we can provide load balancing in distributing the XML data over multiple broadcast channels.
Now, assume that we have three DTSs to disseminate the XML nodes. The process of XML data placement on three DTSs is as follows: in the Step 1, all of the XML nodes of the root-to-node path $Path_1$, $Path_2$, and $Path_3$ are placed on the DTS1, DTS2, and DTS3, respectively since these three root-to-node paths have higher priority than other root-to-node paths in the path summary. As shown in Fig. 9 (a), the total number of bytes disseminated on the DTS1 is less than the total number of bytes disseminated on the DTS2 and DTS3 in this step ("50" < "200" and "50" < "150"). Therefore, in the Step 2, all the XML nodes of the root-to-node path $Path_4$ are placed on the DTS1. As shown in Fig. 9 (b), the total number of bytes disseminated on the DTS1 is less than the total number of bytes disseminated on the DTS2 and DTS3 in this step ("120" < "200" and "120" < "150"). Therefore, in the Step 3, all the XML nodes of the root-to-node path $Path_5$ are placed on the DTS1. The result of XML data placement in the case that we have three DTSs is shown in Fig. 9 (c). By using this scheme to place the XML nodes on multiple wireless broadcast channels, we can provide load balancing in distributing of the XML data over multiple broadcast channels.

It should be noted that the process of XML data placement over multiple DTSs can be extended in the cases that the number of DTSs are 4 and 5 based on Assumption1. However, the explanations of these cases are omitted here in order to save the length of the paper.
4. XML QUERY PROCESSING OVER MULTIPLE WIRELESS BROADCAST CHANNELS

In this section, we explain how mobile clients can process different types of XML queries when XML data is disseminated on multiple wireless broadcast channels.

4.1 Process of Simple Path XML Queries

To process simple path XML queries over multiple wireless broadcast channels, the SimplePathQueryProcessor algorithm is devised as shown in Fig. 10.

![SimplePathQueryProcessor algorithm](image)

Given a simple path XML query $Q$, a mobile client listens to the BTS and reads the paths one after another until a path which is similar to the simple path XML query $Q$
appears on the air (Lines 4-7). By exploiting the field Channel Number of the path which is similar to the simple path XML query $Q$, the mobile client connects to the DTS having the number equals to the field Channel Number (Lines 10-11). By exploiting the two fields Offset1 and Offset2 of the path which is similar to the simple path XML query $Q$ (Lines 8-9), the mobile client jumps forward to the first node satisfying the simple path XML query $Q$ in the DTS and downloads the desired data (Lines 12-16). In the case that the simple path XML query $Q$ contains predicate conditions at the text content or attribute values, the algorithm checks the text content and/or attribute values and adds the current node into the XML query results set $R$ if the predicate conditions of the XML query $Q$ are satisfied (Lines 14-15).

**Example 1:** Assume that the mobile client $c_1$ submits the simple path XML query $Q = "/\text{SigmodRecord}/issue/number"$. By reading the path "/\text{SigmodRecord}/issue/number" from the BTS, the mobile client $c_1$ finds the channel number of DTS which all the XML nodes of the path "/\text{SigmodRecord}/issue/number" are disseminated on it. Then, the mobile client $c_1$ connects to that DTS and switches to the doze mode to conserve its battery power until the first candidate node appears on the air at the time equals to the field offset1 (or firstTime). When the first candidate node arrives on the air, the mobile client $c_1$ wakes up and downloads all the candidate nodes of the XML query $Q$ on the air.

### 4.2 Processing of Simple Path XML Queries having a Wildcard

Generally, simple path XML queries having a wildcard ("*") can be classified into two forms:

1) Simple path XML queries having a wildcard at the end of an XPath expression with the query pattern "/\theta_1/\theta_2/ \ldots /\theta_n/*" where $1 < k < n$, $\theta_k$ is a node name (or a tag name).

2) Simple path XML queries having a wildcard in the middle of an XPath expression with the query pattern "/\theta_1/\theta_2/ \ldots /\theta_1/*/\theta_1/\theta_2/ \ldots /\theta_n/*" where $1 < k < n$, $\theta_k$ is a node name (or a tag name).

To process a simple path XML query with a wildcard in the form of "/\theta_1/\theta_2/ \ldots /\theta_1/*", the mobile client first finds the first child path of the path "/\theta_1/\theta_2/ \ldots /\theta_n/*" from the BTS using the SimplePathQueryProcessor Algorithm. By exploiting the field Channel Number of the first child path, the mobile client connects to the DTS having the number equals to the field Channel Number. By exploiting the two fields Offset1 and Offset2, the first subset of the query results can be retrieved by downloading all the XML nodes which arrive at the interval $[\text{Offset1}, \text{Offset2}]$. After downloading all the XML nodes of the first child path of the path "/\theta_1/\theta_2/ \ldots /\theta_n/*", the mobile client connects to the BTS again to find the second child path of the path "/\theta_1/\theta_2/ \ldots /\theta_n/*" using the SimplePathQueryProcessor Algorithm. By exploiting the fields Channel Number, Offset1, Offset2 of the second child path of the path "/\theta_1/\theta_2/ \ldots /\theta_n/*", the mobile client connects to the DTS having the number equals to the field Channel Number to download the second subset of the query results. This process is continued until all the XML nodes of the child paths of the path "/\theta_1/\theta_2/ \ldots /\theta_n/*" are downloaded.

**Example 2:** Assume that the mobile client $c_2$ submits the simple path XML query $Q = "/\text{SigmodRecord}/issue/articles/article/*"$. The mobile client $c_2$ first connects to the BTS...
to find the first child path of the path "/SigmodRecord/issue/articles/article/*" which is "/SigmodRecord/issue/articles/article/title". Then, by exploiting the three fields Channel Number, Offset1, and Offset2 of the path "//SigmodRecord/issue/articles/article/title", the mobile client \( c_2 \) connects to the DTS having the number equals to the field Channel Number and downloads all the XML nodes which are on the air at the interval [Offset1, Offset2]. Note that this process is repeated for each path of the path "//SigmodRecord/issue/articles/article" in order to retrieve the whole results of the simple path XML query \( Q \).

To process a simple path XML query with a wildcard in the form of "//\( \theta_1//\cdots//\theta_n//\cdots//\theta_k//\cdots//\theta_l//\cdots//\theta_t//\cdots//\theta_s//\cdots//\theta_u//\cdots//\theta_v//\cdots//\theta_w//\cdots//\theta_z //"", the mobile client first finds all the child paths of the path "//\( \theta_1//\cdots//\theta_n//\cdots//\theta_k//\cdots//\theta_i//\cdots//\theta_l//\cdots//\theta_t//\cdots//\theta_s//\cdots//\theta_u//\cdots//\theta_v//\cdots//\theta_w//\cdots//\theta_z //"" using the SimplePathQueryProcessor Algorithm. Then, the mobile client concatenates each child path of the path "//\( \theta_1//\cdots//\theta_n//\cdots//\theta_k//\cdots//\theta_i//\cdots//\theta_l//\cdots//\theta_t//\cdots//\theta_s//\cdots//\theta_u//\cdots//\theta_v//\cdots//\theta_w//\cdots//\theta_z //"" with the path "//\( \theta_1//\cdots//\theta_n//\cdots//\theta_k//\cdots//\theta_i//\cdots//\theta_l//\cdots//\theta_t//\cdots//\theta_s//\cdots//\theta_u//\cdots//\theta_v//\cdots//\theta_w//\cdots//\theta_z //"". Assume that the number of child paths of the path "//\( \theta_1//\cdots//\theta_n//\cdots//\theta_k//\cdots//\theta_i//\cdots//\theta_l//\cdots//\theta_t//\cdots//\theta_s//\cdots//\theta_u//\cdots//\theta_v//\cdots//\theta_w//\cdots//\theta_z //"" is \( k \) Then, the mobile client must find \( k \) different subsets of query results. Hence, the result of the simple path XML query "//\( \theta_1//\cdots//\theta_n//\cdots//\theta_k//\cdots//\theta_i//\cdots//\theta_l//\cdots//\theta_t//\cdots//\theta_s//\cdots//\theta_u//\cdots//\theta_v//\cdots//\theta_w//\cdots//\theta_z //"" is the union of the \( k \) subsets of query results.

### 4.3 Processing of Simple Path XML Queries having a Descendant Axis

Generally, simple path XML queries having a descendant axis can be classified into two forms:

1) Simple path XML queries having a descendant axis at the end of an XPath expression with the query pattern "//\( \theta_i//\cdots//\theta_n//\cdots//\theta_k//\cdots//\theta_j//\cdots//\theta_l//\cdots//\theta_t//\cdots//\theta_s//\cdots//\theta_u//\cdots//\theta_v//\cdots//\theta_w//\cdots//\theta_z //"") where \( 1 < k < n \), \( \theta_i \) is a node name (or a tag name).

2) Simple path XML queries having a descendant axis in the middle of an XPath expression with the query pattern "//\( \theta_i//\theta_j//\cdots//\theta_n//\cdots//\theta_k//\cdots//\theta_j//\cdots//\theta_l//\cdots//\theta_t//\cdots//\theta_s//\cdots//\theta_u//\cdots//\theta_v//\cdots//\theta_w//\cdots//\theta_z //"") where \( 1 < k < n \), \( \theta_i \) is a node name (or a tag name).

To process a simple path XML query with a descendant axis in the form of "//\( \theta_i//\theta_j//\cdots//\theta_n//\cdots//\theta_k//\cdots//\theta_j//\cdots//\theta_l//\cdots//\theta_t//\cdots//\theta_s//\cdots//\theta_u//\cdots//\theta_v//\cdots//\theta_w//\cdots//\theta_z //"" the mobile client first finds the first path which satisfies the simple path XML query "//\( \theta_i//\theta_j//\cdots//\theta_n//\cdots//\theta_k//\cdots//\theta_j//\cdots//\theta_l//\cdots//\theta_t//\cdots//\theta_s//\cdots//\theta_u//\cdots//\theta_v//\cdots//\theta_w//\cdots//\theta_z //"" from the BTS using the SimplePathQueryProcessor Algorithm. By exploiting the three fields Channel Number, Offset1, and Offset2 of the first path which satisfies the simple path XML query "//\( \theta_i//\theta_j//\cdots//\theta_n//\cdots//\theta_k//\cdots//\theta_j//\cdots//\theta_l//\cdots//\theta_t//\cdots//\theta_s//\cdots//\theta_u//\cdots//\theta_v//\cdots//\theta_w//\cdots//\theta_z //"" the mobile client can connect to the DTS having the number equals to the filed Channel Number and download all the XML nodes at the interval [Offset1, Offset2]. This process is repeatedly done for each path which satisfies the simple path XML query "//\( \theta_i//\theta_j//\cdots//\theta_n//\cdots//\theta_k//\cdots//\theta_j//\cdots//\theta_l//\cdots//\theta_t//\cdots//\theta_s//\cdots//\theta_u//\cdots//\theta_v//\cdots//\theta_w//\cdots//\theta_z //"".

**Example 3:** Assume that the mobile client \( c_3 \) submits the simple path XML query \( Q = "//SigmodRecord/issue/articles/article/" \). The mobile client \( c_3 \) first connects to the BTS to find the first path which satisfies the simple path XML query "//SigmodRecord/issue/articles/article/" which is the path "//SigmodRecord/issue/articles/article". By exploiting the three fields Channel Number, Offset1, Offset2 of the path "//SigmodRecord/issue/articles/article/" the mobile client \( c_3 \) connects to the DTS having the number equals to the filed Channel Number and download all the XML nodes at the interval [Offset1, Offset2]. This process is repeated for the other paths which satisfy the simple path XML query "//SigmodRecord/issue/articles/article/" to retrieve the whole results of the simple path XML query \( Q \).
To process a simple path XML query with a descendant axis in the form of "/θ_1/ θ_2/ ... /θ_j/θ_{j+1}/ ... /θ_n", the mobile client first finds all the paths which satisfy the simple path XML query "/θ_1/θ_2/ ... /θ_j/" from the BTS. Assume that \( 1 < j < t \), \( P_j \) be the \( j \)th path which satisfies the simple path XML query "/θ_1/θ_2/ ... /θ_j//". The mobile client concatenates each path \( P_j \) with the path "/θ_{j+1}/ ... /θ_n" and then searches for each constructed path in the BTS. Hence, the results set of the simple path XML query "/θ_1/θ_2/ ... /θ_j/θ_{j+1}/ ... /θ_n" can be retrieved by downloading all of the XML nodes having the constructed paths.

4.4 Processing of Twig Pattern XML Queries having a Predicate Condition

A twig pattern XML query is an XML query with two or more XPath expressions which contains predicate conditions. Generally, twig pattern XML queries having a predicate condition can be classified into two forms:

1) Twig pattern XML queries having a predicate condition at the end of an XPath expression with the query pattern "/θ_1/θ_2/ ... /θ_{n-i}/θ_{n-i+1}/ ... /θ_n/\text{() = "..."}[/θ_{n-i+1}/ ... /θ_n]" where \( 1 < k < n \), \( \theta_k \) is a node name (or a tag name).

2) Twig pattern XML queries having a predicate condition in the middle of an XPath expression with the query pattern "/θ_1/θ_2/ ... /θ_{i-1}/θ_i/θ_{i+1}/ ... /θ_n/\text{() = "..."}[/θ_i/θ_{i+1}/ ... /θ_n]" where \( 1 < k < n \), \( \theta_k \) is a node name (or a tag name).

To process a twig pattern XML query with a predicate condition in the form of "/θ_1/θ_2/ ... /θ_{n-i}/θ_{n-i+1}/ ... /θ_n/\text{() = "..."}[/θ_{n-i+1}/ ... /θ_n]", the mobile client first decomposes the twig pattern XML query into two XML queries: \( Q_1 =="/θ_1/θ_2/ ... /θ_{n-i}/" \) and \( Q_2 =="/θ_1/θ_2/ ... /θ_{n-i}/θ_i/\text{() = "..."}[/θ_{n-i+1}/ ... /θ_n]" \). These two XML queries can be processed by the SimplePathQueryProcessor algorithm. Note that the XML query \( Q_2 \) has a predicate condition over the text content and therefore, it can be processed by the SimplePathQueryProcessor algorithm (See Lines 14-15 in Fig. 10). After downloading the results of the queries \( Q_1 \) and \( Q_2 \), the mobile client uses the preorder, postorder, and depth information of the XML nodes in the query results to find the results set of the twig pattern XML query. This process is done by exploiting the following properties:

**Property 1:** An XML node \( \alpha \) labeled by \((\text{preorder}_\alpha, \text{postorder}_\alpha, \text{depth}_\alpha)\) is the ancestor node of an XML node \( \beta \) labeled by \((\text{preorder}_\beta, \text{postorder}_\beta, \text{depth}_\beta)\) if and only if \( \text{preorder}_\alpha < \text{preorder}_\beta \) and \( \text{postorder}_\alpha > \text{postorder}_\beta \).

For example, in Fig. 2, the node “issue” with the label (20, 28, 2) is the ancestor node of the node “title” with the label (25, 21, 5) since “20” < “25” and “28” > “21”.

**Property 2:** An XML node \( \alpha \) labeled by \((\text{preorder}_\alpha, \text{postorder}_\alpha, \text{depth}_\alpha)\) is a descendant node of an XML node \( \beta \) labeled by \((\text{preorder}_\beta, \text{postorder}_\beta, \text{depth}_\beta)\) if and only if \( \text{preorder}_\alpha > \text{preorder}_\beta \) and \( \text{postorder}_\alpha < \text{postorder}_\beta \).

For example, in Fig. 2, the node “article” with the label (13, 16, 4) is the descendant node of the node “issue” with the label (2, 18, 2) since “13” > “2” and “16” < “18”.


Property 3: An XML node \( \alpha \) labeled by \((\text{preorder}_\alpha, \text{postorder}_\alpha, \text{depth}_\alpha)\) is the parent node of an XML node \( \beta \) labeled by \((\text{preorder}_\beta, \text{postorder}_\beta, \text{depth}_\beta)\) if and only if \(\text{preorder}_\alpha < \text{postorder}_\beta \) and \(\text{depth}_\alpha = \text{depth}_\beta - 1\).

For example, in Fig. 2, the node “authors” with the label \((15, 13, 5)\) is the parent node of the node “author” with the label \((16, 11, 6)\) since “15” < “16”, “13” > “11”, and “5” = “6” – “1”.

Property 4: An XML node \( \alpha \) labeled by \((\text{preorder}_\alpha, \text{postorder}_\alpha, \text{depth}_\alpha)\) is a child node of an XML node \( \beta \) labeled by \((\text{preorder}_\beta, \text{postorder}_\beta, \text{depth}_\beta)\) if and only if \(\text{preorder}_\alpha > \text{postorder}_\beta\) and \(\text{depth}_\alpha = \text{depth}_\beta + 1\).

For example, in Fig. 2, the node “initPage” with the label \((11, 7, 5)\) is the child node of the node “article” with the label \((6, 9, 4)\) since “11” > “6”, “7” < “9”, and “5” = “4” + “1”.

Example 4: Assume that the mobile client \( c_4 \) submits the twig pattern XML query \( Q = \langle /\text{SigmodRecord}/\text{issue}[\text{volume}/\text{text}()] = "12\rangle \rangle \). The mobile client \( c_4 \) first decomposes the XML query \( Q \) into two XML queries \( Q_1 = \langle /\text{SigmodRecord}/\text{issue} \rangle \) and \( Q_2 = \langle /\text{SigmodRecord}/\text{issue}[\text{volume}/\text{text}()] = "12\rangle \rangle \). The mobile client \( c_4 \) uses the SimplePathQueryProcessor algorithm to process the XML queries \( Q_1 \) and \( Q_2 \). The results set of query \( Q_1 \) is the two nodes “issue” labeled by \((2, 18, 2)\) and \((20, 28, 2)\). The results set of query \( Q_2 \) is the node “volume” labeled by \((21, 19, 3)\). By exploiting the Property 3, the mobile client \( c_4 \) finds the results set of the twig pattern XML query \( Q \) at the node “issue” labeled by \((20, 28, 2)\) since this node is the parent node of the node “volume” labeled by \((21, 19, 3)\).

To process a twig pattern XML query with a predicate condition in the form of \( /\text{c}/ \theta_1/ \ldots/ \theta_n/ [\theta_1/ \theta_2/ \ldots/ \theta_m/\text{text()} = "\ldots"] \), the mobile client first decomposes the twig pattern XML query into two XML queries: \( Q_1 = \langle /\text{c}/ \theta_1/ \ldots/ \theta_n/ [\theta_1/ \theta_2/ \ldots/ \theta_m/\text{text()} = "\ldots"] \rangle \) and \( Q_2 = \langle /\text{c}/ \theta_1/ \ldots/ \theta_n/ [\theta_1/ \theta_2/ \ldots/ \theta_m/\text{text()} = "\ldots"] \rangle \). Then, the mobile client decomposes the twig pattern XML query \( Q_1 \) into two XML queries \( Q_{11} = \langle /\theta_1/ \theta_2/ \ldots/ \theta_n/\text{text()} = "\ldots" \rangle \) and \( Q_{12} = \langle /\theta_1/ \theta_2/ \ldots/ \theta_n/\text{text()} = "\ldots" \rangle \). The XML queries \( Q_{11}, Q_{12}, \) and \( Q_2 \) can be processed by the SimplePathQueryProcessor algorithm. After downloading the results of the queries \( Q_{11}, Q_{12}, \) and \( Q_2 \) the mobile client uses the preorder, postorder, and depth information of the XML nodes in the results of the queries \( Q_{11}, Q_{12}, \) and \( Q_2 \) to find the result of the twig pattern XML query. This process is done by exploiting the properties 1, 2, 3, and 4.

Example 5: Assume that the mobile client \( c_5 \) submits the twig pattern XML query \( Q = \langle /\text{SigmodRecord}/\text{issue}[\text{volume}/\text{text}()] = "11\rangle \rangle /\text{articles}/\text{article}/\text{title} \rangle \). The mobile client \( c_5 \) first decomposes the XML query \( Q \) into two XML queries \( Q_1 = \langle /\text{SigmodRecord}/\text{issue}[\text{volume}/\text{text}()] = "11\rangle \rangle /\text{articles}/\text{article}/\text{title} \rangle \) and \( Q_2 = \langle /\text{SigmodRecord}/\text{issue}[\text{volume}/\text{text}()] = "11\rangle \rangle /\text{articles}/\text{article}/\text{title} \rangle \). Then, the mobile client \( c_5 \) decomposes the XML query \( Q_1 \) into two XML queries \( Q_{11} = \langle /\text{articles}/\text{article}/\text{title} \rangle \) and \( Q_{12} = \langle /\text{articles}/\text{article}/\text{title} \rangle \). The mobile client \( c_5 \) uses the SimplePathQueryProcessor algorithm to process the XML queries \( Q_{11}, Q_{12}, \) and \( Q_2 \). The results set of query \( Q_{11} \) is the two nodes “issue” labeled by \((2, 18, 2)\) and \((20, 28, 2)\). The results set of the query \( Q_{12} \) is the node “volume” labeled by \((3, 1, 3)\). The results set of the query \( Q_2 \) is the three nodes “title” labeled by \((7, 3, 5)\), \((14, 10, 5)\), and \((25, 21, 5)\). By exploiting the Property 3, the mobile client \( c_5 \) finds the result of the query \( Q_1 \) at the node “issue” labeled by \((2, 18, 2)\). It also finds the results set of the twig pattern query \( Q \) at the two nodes “title” labeled by \((7, 3, 5)\) and \((14, 10, 5)\) by exploiting the Property 2.
5. PERFORMANCE EVALUATION

In this section, we evaluate the performance of our proposed XML data placement scheme in processing different types of XML queries by performing several experiments. All the experiments were conducted on a system with the Intel 3.0 GHz processor and 4GB RAM running on Windows 7 Ultimate 64-bits where all the codes were implemented in C# with the Microsoft .NET Framework 4.0.

5.1 Experimental Settings

We logically modeled the XML stream on multiple broadcast channels as binary files, where the broadcast server writes byte streams on the files and the mobile clients read the files and process the XML queries.

In our simulation model, we assumed that all the broadcast channels have the same bandwidth and the broadcast bandwidth is fully utilized for broadcasting XML data.

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 performance of XML query processing 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.

To measure the performance variation based on the types of XML data sets, we used several real and syntactic XML data sets. Table 3 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 4.

5.2 Experimental Results on Access Time

Figs. 11 (a)-(d) show the access time of a mobile client in processing of the different types of XML queries on the different XML data sets when the total number of wireless broadcast channels is varied. As shown in Figs. 11 (a)-(d), the access time of a mobile client in processing of the different types of XML queries is improved when the number of wireless broadcast channels increases. It means that disseminating XML data over multiple broadcast channels improve the access time of mobile clients in XML query processing.

Table 3. XML data sets.

<table>
<thead>
<tr>
<th>Data Set</th>
<th>Size (KB)</th>
<th>Number of Elements</th>
<th>Number of Attributes</th>
<th>Max Depth</th>
<th>Max Fan Out</th>
</tr>
</thead>
<tbody>
<tr>
<td>University Courses</td>
<td>278</td>
<td>10,546</td>
<td>0</td>
<td>4</td>
<td>703</td>
</tr>
<tr>
<td>Shakespeare</td>
<td>1,061</td>
<td>25,339</td>
<td>0</td>
<td>7</td>
<td>162</td>
</tr>
<tr>
<td>XMark</td>
<td>1,155</td>
<td>17,132</td>
<td>39</td>
<td>12</td>
<td>255</td>
</tr>
<tr>
<td>Mondial</td>
<td>1,743</td>
<td>22,423</td>
<td>47,423</td>
<td>5</td>
<td>955</td>
</tr>
</tbody>
</table>
Table 4. XPath query sets of the different XML data sets.

<table>
<thead>
<tr>
<th>XML Data Set</th>
<th>Query Name</th>
<th>XPath Expression</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>University Courses</strong></td>
<td>UC1</td>
<td>/root/course/title</td>
</tr>
<tr>
<td></td>
<td>UC2</td>
<td>/root/course/*/start_time</td>
</tr>
<tr>
<td></td>
<td>UC3</td>
<td>/root/course/time</td>
</tr>
<tr>
<td></td>
<td>UC4</td>
<td>/root/course/crse[text()=&quot;300&quot;]</td>
</tr>
<tr>
<td></td>
<td>UC5</td>
<td>/root/course/place/building</td>
</tr>
<tr>
<td></td>
<td>UC6</td>
<td>/root/course/crse[text()=&quot;300&quot;]/time/start_time</td>
</tr>
<tr>
<td><strong>Shakespeare</strong></td>
<td>SH1</td>
<td>/PLAYS/PLAY/ACT</td>
</tr>
<tr>
<td></td>
<td>SH2</td>
<td>/PLAYS/PLAY/*/PGROUP</td>
</tr>
<tr>
<td></td>
<td>SH3</td>
<td>/PLAYS/PLAY//SPEAKER</td>
</tr>
<tr>
<td></td>
<td>SH4</td>
<td>/PLAYS/PLAY/ACT/SCENE/SPEECH/STAGEDIR[@SPEAKER=&quot;BERTRAM&quot;]</td>
</tr>
<tr>
<td></td>
<td>SH5</td>
<td>/PLAYS/PLAY/ACT/SCENE/SPEECH/STAGEDIR[@SPEAKER=&quot;BERTRAM&quot;]</td>
</tr>
<tr>
<td></td>
<td>SH6</td>
<td>/PLAYS/PLAY/ACT/SCENE/SPEECH/STAGEDIR[text()=&quot;MARK ANTONY&quot;]/TITLE</td>
</tr>
<tr>
<td><strong>XMark</strong></td>
<td>XM1</td>
<td>/site/regions/samerica/item</td>
</tr>
<tr>
<td></td>
<td>XM2</td>
<td>/site/categories/*/description/parlist</td>
</tr>
<tr>
<td></td>
<td>XM3</td>
<td>/site/open_auctions/date</td>
</tr>
<tr>
<td></td>
<td>XM4</td>
<td>/site/people/person/address/country[text()=&quot;United States&quot;]</td>
</tr>
<tr>
<td></td>
<td>XM5</td>
<td>/site/people/person/profile[@income=&quot;9876.00&quot;]</td>
</tr>
<tr>
<td></td>
<td>XM6</td>
<td>/site/closed_auctions/closed_auction[annotation]/price</td>
</tr>
<tr>
<td><strong>Mondial</strong></td>
<td>MO1</td>
<td>/mondial/country/religions</td>
</tr>
<tr>
<td></td>
<td>MO2</td>
<td>/mondial/country/*/name</td>
</tr>
<tr>
<td></td>
<td>MO3</td>
<td>/mondial/country/population</td>
</tr>
<tr>
<td></td>
<td>MO4</td>
<td>/mondial/country/province/city[name[text()=&quot;Strasbourg&quot;]</td>
</tr>
<tr>
<td></td>
<td>MO5</td>
<td>/mondial/country/province/city[@country=&quot;0 418&quot;]</td>
</tr>
<tr>
<td></td>
<td>MO6</td>
<td>/mondial/country/province/city/population[text()=&quot;10000&quot;]/name</td>
</tr>
</tbody>
</table>

Fig. 11. Access time on the different data sets.
5.3 Experimental Results on Tuning Time

Figs. 12 (a)-(d) show the tuning time of a mobile client in processing of the different types of XML queries on the different XML data sets when the total number of wireless broadcast channels is varied. As shown in Figs. 12 (a)-(d), the tuning time of a mobile client in processing of the different types of XML queries does not change with increasing the number of wireless broadcast channels. It means that disseminating XML data over multiple broadcast channels does not degrade the performance of XML query processing in terms of tuning time.

![Tuning time graphs for different data sets](image)

Fig. 12. Tuning time on the different data sets.

5. CONCLUSION

In this paper, we proposed a new structure for streaming XML data over multiple wireless broadcast channels by exploiting the path summary technique and the pre/post labeling scheme. We also proposed a novel XML data placement scheme to place XML data among multiple wireless broadcast channels in order to improve the performance of XML query processing in terms of access time at the mobile clients. By performing several experiments on the real and syntactic XML data sets, we demonstrated that our proposed XML data placement scheme improve the performance of mobile clients in processing of XML queries in terms of access time.

REFERENCES


**Hossein Mirabi** obtained his Master degree in Computer Engineering from the University of Kashan, Kashan, Iran in 2015. His research interests include information storage and retrieval, pattern recognition and machine learning.

**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.

**Ali Borjian Boroujeni** obtained his Master degree in Software Engineering from the School of Computer Engineering, Iran University of Science and Technology, Tehran, Iran in 2015. His research interests include XML database and security in distributed computing.