ASSOCIATIVE QUERY FOR MULTI-VERSION WEB DOCUMENTS

Yu ZHANG  Kang ZHANG

Department of Computing, Division of ICS
Macquarie University, Sydney, 2109 Australia
E-mail: {yzhang, kang}@ics.mq.edu.au

With the growing importance of the World Wide Web [1] as a truly global distributed system for information sharing, there has been a significant increase in the creation and modification of web documents. In order to produce and modify web sites efficiently, approaches to the creation of multi-version web sites have been developed. A multi-version Web site contains different versions of web documents and allow all the different versions to share one generic source page. Current practice of searching the Web using existing search tools is not adaptable for multi-version web sites. This paper proposes a database-like querying approach for more precise and complex querying of multi-version web sites. Associative query is achieved by generating virtual version tables (VVT) upon different versions of web documents and joining them together to retrieve the exact information needed. We provide a Java applet interface running on the client side for submitting queries. On the server side, JDBC is used to communicate with Oracle database to manipulate data in VVTs. On top of the JDBC and Oracle server, we run Java Jeeves & Servlet to circumvent the applet restrictions thus to provide a generic environment.

1. Introduction

With the growing importance of the World Wide Web [1] as a truly global distributed system for information sharing, there has been a significant increase in the creation and modification of web documents. In order to produce and modify variants of web sites efficiently, approaches to the creation of multi-version web sites have been developed. A multi-version Web site contains different versions of web documents and allow all the different versions to share one generic source page. Meanwhile, as more and more retrievals for documents of interest are being conducted by searching the World Wide Web, more precise queries are required so as to adequately extract Web-objects from millions of Web pages. Current practice of searching the World Wide Web is to use the existing commercial search tools such as AltaVista [2], WebCrawler [3], OpenText [4], etc, or search services such as Bigbook [5], Yahoo [6], etc. The search engines deployed by both search tools and search service provide keyword(s) indexing and some content indexing. The resources that the search engines index are the registered URL databases. For a web
site where multi-version codes [7, 8] share one generic source code, the URL that is registered with the search tools can only refer to one version of the web site. Consequently, those conventional query methodologies are not adaptable to multi-version sites. For instance, a web designer may intend to create a site that provides information such as news, analysis, company introductions, historic and real time prices, online ordering, etc., for each of the main stock markets in the world. For easy creation and modification, a multi-version site in which each stock market source page (a version) shares a generic information page would be the most appropriate. If the information that clients are trying to retrieve resides on the specific version of the stock market source page, querying through the existing search tools will definitely be unsuccessful, as only one specific version is searched. Another example that illustrates the limitation of the existing searching tools is when they are used in querying multiple language sites. Although most of the search tools provide language options for clients to retrieve documents written in the language they favour, search engines cannot return the URLs in which the language used to create the web documents differs from the language selected for retrieving URLs. Thus it can be seen that the existing search tools are only suitable for one version sites. Furthermore, these search tools cannot support the queries that combine different versions of a site.

This paper proposes an associative querying approach for information retrieval on multi-version web sites. We categorize objects in a generic HTML source page into three hierarchical classes, i.e., root, node and leaf. A hierarchical graph is created according to the categorization to facilitate retrieving during the query procedure. Virtual version tables (VVT) are generated upon different versions of web documents and joined together in retrieving the exact information from different web documents. We provide a Java applet interface running on the client side for submitting queries and browsing the results. Javier Jeelee is running on top of JDBC and Oracle database on the HTTP server side to manipulate multiple VVTs.

The organization of this paper is as follows: Section 2 gives a detailed explanation of multi-version web sites. Section 3 presents the hierarchical web data model and schema for the associate queries. Section 4 presents a user defined SQL-like query statement. Section 5 describes the prototype design and implementation of the system. We give summary and future work in Section 6.

2. Multi-version Web sites

We may view a Web site as a collection of all HTML documents of a domain which can be mapped to a directory on a HTTP server. With the significant growth of on-line information, web designers are under increasing pressure to produce and update Web sites. The conventional way of producing and modifying a Web site
is to create every single page and make changes on the copy of the source code of the page. Problems arise in the sites where the main frames of the pages are almost the same and while changes are needed for only part of the original documents. Using the conventional method to create and maintain these sites is time-consuming and wasteful in resources. By adding "footer" at the end of each document to represent the common part of different pages can only benefit small changes to a site. Therefore, to produce and modify variants of web sites efficiently, approaches to the creation of multi-version web sites have to be developed.

![Diagram of a multi-version Web site](image)

**Figure 1 A multi-version Web site**

In a multi-version web site, a generic source page on the web acts as a template for other HTML documents in the same site. The generic source page represents the common part of other documents in the site. It is also the index page of the site. Each HTML document (except the generic source document) can be seen as a
version of the site. The requests from clients are associated with version labels which can be interpreted by CGI or Java Servlet program on the WWW server to point to an appropriate version of a HTML document on the same server. The appropriate documents are then retrieved and loaded to the template of the generic source page. Thus, different versions of the site are generated upon different requests from clients. Figure 1 depicts an example of a multi-version stock market site as described in Section 1.

3. Multi-version Web schema

3.1 Multi-version document Modeling

Querying multi-version sites in this approach requires both page generation and information retrieval. In order to facilitate this retrieval, we view one version of a HTML document generated by the generic source code as a compound object of a hierarchical structure with an affiliated version label. Objects in one specific version are categorized into three hierarchical classes - root, node and leaf, as illustrated in Fig2.

Class Root = {[Title].Version Label}

Class Node = { [Href],
               [Heading],
               [Node],
               [Include],

Class leaf = {[Url].Version Label,
               [Image].Version Label,
               [Audio].Version Label,
               [Applet].Version Label,
               [HTML].Version Label,
               .......... [String], [Real], [Integer]

Figure 2 Hierarchical classes of Web document

The title of the page is known as the root class. Node classes includes links, includes, headings. They also contain other node classes. Leaf classes are
disjointable objects such as graphic files with extensions .gif, JPEG, or MPEG, etc audio files with extensions .au, and real, integer, html file, java applets. A version Label is assigned as an attribute to root class and objects in a leaf class when submitting the request for a specific version. A hierarchically structured graph is created when the version document is generated. Information retrieval is accomplished by querying the graph of the specific version. The generation of a version graph can be briefly described as follows:

```java
public Node versionGraph(String url, int version_label)
{
    Class Root, Node, Leaf;
    if(Leaf)
        (Vector)Root.addElement(Leaf);
    If(Node)
    {
        (Vector)Root.addElement(Node);
        (Vector)Node.addElement(versionGraph(Node, versin label));
    }

    return(Leaf);
}
```

---

**Figure 3** A specific version graph
The above pseudo code shows how a version graph is generated with individual nodes and leaves added onto the graph and also figure 2 describes a graph of a particular version of a multi-version Web site.

3.2 Virtual Version Table schema

The above sections have discussed the generating and modeling a specific version of a multi-version web site. Querying through the structured graph of the specific version of the site will return the information that fits the clients’ requests. In many circumstances, the information which clients want cannot be retrieved by just querying one version. For the purpose of querying objects which need to cooperate with different versions or different multi-version sites, we create virtual version tables (VVT) based on the version graphs. A virtual version table supports the combination according to the primary key. The VVT schema comprises 4 attributes: i.e. url, version label, keyword and primary key. All the attributes are string data types. Attribute url is used for associating different sites. Values of the attributes are assigned according to url, version labels and keywords specified in the query. Primary keys are set up for the VVTs. The value of the primary key in the first table is the same as the keyword value. The value of the next virtual table’s primary key is the same as the prior table’s keyword.

**Virtual Version Table 1:**

<table>
<thead>
<tr>
<th>Version Label</th>
<th>Kw</th>
<th>Pk</th>
</tr>
</thead>
<tbody>
<tr>
<td><a href="http://www.comp.mq.edu.au">http://www.comp.mq.edu.au</a></td>
<td>Version1</td>
<td>Keyword1</td>
</tr>
</tbody>
</table>

**Virtual Version Table 2:**

<table>
<thead>
<tr>
<th>Version Label</th>
<th>Kw</th>
<th>Pk</th>
</tr>
</thead>
<tbody>
<tr>
<td><a href="http://www.mpce.mq.edu.au">http://www.mpce.mq.edu.au</a></td>
<td>Version2</td>
<td>Keyword2</td>
</tr>
</tbody>
</table>

.......

**Virtual Version Table N:**

......

*Figure 4  Virtual version table schema & its association*
We describe the schema as well as the VVTs created for different versions of a site and different sites in Figure 3. The underlying structure of the VVTs is a set of graphs. Queries will go through the VVTs and then explore the graphs to extract the desired information. For combination purpose, each table is identified by its keyword. The number of tables equals the number of keywords in the query, tables are joined by the primary keys.

4. Querying the Web

In this section, we introduce our SQL-like query language for multi-version Web sites.

A query statement in the associative Web query a combination of SELECTCT, FROM, WHERE, and AND clauses. The syntax of a query template is shown below:

```
SELECT keywords(x1,...xn)
FROM url.versin1
AND url.version2
.....
AND url.versionn
WHERE Condition[(x1...xn), (v1,...Vn)]
```

The SELECT clause consists of keywords of the query. The FROM clause consists of all the virtue version tables that have the same keyword value. Operations on the of conditions such as logic AND are supported in the query. The FROM ... AND clause forms and associates the VVTs during the query procedure according to the table schema discussed in Section 3.2. The WHERE clause is a conditional search expression. The appropriate version of a site can be found by matching the label value in the WHERE clause, so that more than one version a Web page graph can be simultaneously searched by the select operation in the query.

The associative query interpreter translates the query statement according to algorithm depicted in Figure 5.
\begin{figure}
\centering
\begin{verbatim}
Compute KW1 = \{x1 | url.version1 = true\}
For all x1 \in KW1
    Compute KW2 = \{x2 | url.version2 = true\}
    For all x2 \in KW2
        ... 
        Compute KWn = \{xn | url.versionn = true\}
        For all xn \in KWn
            If condition[(x1 ... xn),(v1 ... vn)] = true
                Write(KW(x1, ... xn))
            End if
        End for
        ... 
    End for
End for
\end{verbatim}
\caption{The loop for parsing query statement}
\end{figure}

5. Design and Implementation

The architecture of the associative query system consists of four major components: user interface, query interpreter, DBMS for Virtual Version Tables (VVTs), and version file system.

In order to support the portability, the query system is implemented in Java Applet which can be loaded and run by a Web browser on the client side. The input form of the user interface can be used as a template for the most common queries to make it easy for the user to submit queries. If a query is more complicated, it can always be typed into an alternative text field. After the query is entered, it may be submitted by pressing the appropriate button. At that point, the Java applet collects all the data from the input fields and assembles the associative queries.

Queries from clients are first received by the query interpreter which also checks the syntax. The query interpreter is a Java servlet program running on the multi-version Web server. Once a recognized query is received, the interpreter...
produces its object code and then executes it. The execution includes mapping the desired version of the site in the query to the coordinate version label in the multi-version file system thus pointing to the appropriate version of the HTML document.

As the virtual version tables are created during the query process, a DBMS is used to manage and manipulate such databases. A Java database connection (JDBC) bridges between the query interpreter and DBMS to handle the communications between the Java servlet and the Oracle DBMS. The DBMS generates the appropriate urls and versions which can be passed as variables to the interpreter. The html documents can then be retrieved accordingly and sent to the applet interface.

![System architecture](image)

**Figure 6** System architecture

Multi-version file system stores all the versions of the HTML documents of the site and version labels. Version labels are linked with the appropriate documents and also associated with the corresponding items on the user interface.

By running Java Jeeves prior to the the servlet program to circumvent the restrictions of the applet, a generic user environment is provided.

The system architecture is shown in Figure 5.

6. Conclusions and future directions

This paper has presented the formal semantics in terms of a graph model for querying multi-version sites. Virtual version tables are proposed for combining different versions of a HTML document and different sites. A prototype implementation of this approach has also been outlined.

Several directions for extending this work are addressed as follows:
Firstly, as the execution of the query is influenced by various factors, such as the size of the transferred document, the bandwidth of the network and the load of the Web servers which have been accessed. Apart from improving the above mentioned system features, it should be noted that to improve the overall efficiency of the query, an algorithm optimising the query needs to be considered.

Secondly, the query interpreter does not have knowledge handling capability, for a more complex query, it will need user to do some analysis before submitting. Building an agent based interpreter to handle knowledge analysising for complex queries is under consideration.

Lastly, the current approach is adaptable to one or more multi-version web sites that specified by users in their queries. So taking advantages of the exist search engines, sending mobile search agents to query through those search engines would be a great challenge to extend this work.

Reference