Abstract

The number of Web Usage Mining (WUM) applications is growing continuously, especially due to the business interest in e-commerce Web sites and the related Web-marketing applications. The application of WUM results goes beyond the subject of our thesis since one important part of our thesis deals with the problem of selection of webviews using techniques of WUM techniques. View materialization is an important issue if we want to improve the efficiency of many applications like OLAP, Database and Web applications. In this paper, we propose a novel approach for selecting webviews to be materialized in order to optimize the response time of web queries. The selection of webviews to be materialized was mainly based on the estimation of metrics requiring hard collects of multiple statistics [9] that’s why, we believe on a solution based on mining an interesting set of webviews to be materialized from realistic data: Web log files. Thus, Web log files will be parsed, analyzed and treated to give a set of webviews, based on frequent closed itemsets.


General Terms: Design, Performance.

Keywords: Materialized webviews, Web log files, Webview Selection, Web Usage Mining, Closed frequent itemsets.

1. Introduction

Most commercial web sites, such as portal sites, search engines, e-commerce sites apply a systematic technique to generate web sites from databases. These web sites are referred to as systematic web sites [6]. Web pages in these sites are generated dynamically by predefined templates to include personalization and advertising features. In order to reduce the overhead for generating dynamic data, it is often feasible to generate data corresponding to a dynamic object once, store the object in a cache, and subsequently serve requests to the object from cache instead of invoking the server program again. A personalized Web page contains content specific to a client, such as the client’s name. Such a Web page could not be used for another client. Therefore, caching the page is of limited utility since only a single client can use it. In fact, each client would need a different version of the page. Moreover, different fragments in a page can have different update frequency, so that, caching the entire page require when one update is necessary, the recomputation of the whole page even if some parts of this page did not change. For these reasons, we are interested in this thesis on the materialization of some “special components” of web pages referenced as Webviews and not on the caching of entire web pages. Similarly to database views, webviews can be classified into two categories: virtual or materialized. Virtual webviews are computed dynamically on-demand from databases, whereas materialized webviews are precomputed and their results are reused when a page containing them is required by an user. For the first category, the cost to compute the webview increases the response time because the web server needs additional time to compute the webviews. Whereas, in the second category (materialized webviews), every update operation of the database needs to update the webview, which increases the serverload. Webviews are usually generated by ‘wrapping’ database query results (i.e. database views) with HTML formatting commands or XML semantic tags. These views are organized in order to rapidly answer various types of web queries [9]. But, materializing all the webviews may degrade Quality of the Data (QoD) which is the average freshness of the served Web pages. In fact, the decision to materialize a webview or not can be relatively similar to the view selection problem encountered in datawarehouse: selecting which views to materialize. It is clear that webview materialization improves Quality of Service (QoS) which is the average response time for the user requests, but it poses new challenges that the most important are: (i) how to select a set of webviews to be materialized; (ii) what knowledge is used to achieve this materialization. In this paper, we propose to use knowledge about user’s behaviors and historical of navigation in order to select a good set of webviews to be materialized. This choice is motivated by the way that traditional approaches are based on the estimation of two defined metrics, namely the performance and the freshness of the data [9]. However, we found that other important re-
requirements, like user navigational patterns and user’s preferences are completely omitted by traditional approaches. For this purpose, we consider that user’s behaviors can improve drastically the performance and the quality of a webview materialization process. The knowledge about user’s behaviors can be extracted from Web log files that save information about navigational patterns of web users. To extract this information and to maintain them using one of the following three policies: [3, 10, 13] and especially association rules mining [11] to generate a good set of webviews to be materialized.

The rest of this paper is organized as follow. Section 2 reviews related works about webviews materialization problem. Section 3 describes our proposed approach. Finally, Section 4 concludes the paper and highlights some future works.

2. RELATED WORKS

Materialization was mainly studied in OLAP systems since a datawarehouse could be seen as a set of materialized views defined over remote source relations. The reader can refer to [14] which proposes a general framework for the view selection problem in datawarehouse. In the web context, Works of Challenger et al. [1, 2] deal with fragments or webviews to propose a novel approach for creating and maintaining efficiently dynamic web content. The authors in [5] gives a solution that saves on the DBMS the results of parameterized SQL computation, under the form of relational tables called cache functions, and reuse them for subsequent requests. This solution can be interesting especially when the SQL execution time cost is very high. In [4] the authors deduce that there is no miraculous method in improving server performance especially in data-intensive Web sites, thus they propose an hybrid method based on a three level architecture of caching namely: SQL caching, XML representation caching and html files caching based on granularity and frequency of update of data. The selection of webviews to be materialized, is mainly treated by Labrinidis et al. [7, 8, 9]. Labrinidis et al. [9] extended the typical three-tier architecture of modern web servers by adding an Asynchronous Cache Module between the Application Server and the Database Server. This component plays an important role to intercept requests that require dynamically generated content, to refresh them if update operations are executed on demand and never cached. Intercepted queries against virtual webviews are forwarded to the database server, whereas database updates do not affect them. This policy corresponds to a server based on an update mechanism of consistency.

• Virtual webviews: In this policy, webviews are always executed on demand and never cached. Intercepted queries against virtual webviews are forwarded to the database server, whereas database updates do not affect them. This policy corresponds to a server based on an update mechanism of consistency.

• No-materialized (cached) webviews: webviews are cached in an asynchronous cache, but if an update invalidates a cached webview, this latter will be recomputed on the next access request. This policy corresponds to a server based on an invalidation mechanism.

• Materialized webviews: This kind of webviews are cached and continuously maintained in the presence of update operations.

Thus, the main difference between materialized and no-materialized webviews, as mentioned by Labrinidis et al. [9], is the decoupling of serving access requests from updating webviews. With materialization, updates are not in the critical path of serving user requests. Without materialization, updates must be taken care of while serving user requests because the system must refresh a stale webview before responding to the user. Thus, the choice of webviews to materialize has an important impact on performance and data freshness. Because as we mentioned before, materializing all webviews will give high performance, but lead to low quality of data (i.e. views will be served very fast, but can be stale) and by keeping all webviews non materialized the quality of data will be high, but the performance will be low. The problem of selection of webviews is defined by Labrinidis et al. [9] as follows: In the presence of continuous access and update streams, dynamically select which web views to materialize, so that the overall system performance is maximized while the freshness of the served data is maintained at an acceptable level. The authors proposed a solution based on collecting multiple statistics and on the estimation of two metrics, namely QoS and QoD.

3. PROPOSED APPROACH

Selecting which views to materialize is an important problem for managers of web sites because materializing a good set of webviews allows reutilization and thus improves the performance of sites. As a solution, we propose a new approach that suggests to use the access patterns of users in order to select the set of webviews to be materialized. In fact, the access patterns of users are an important and useful knowledge that can lead to find the best webviews to be materialized. Components of our proposed approach are described in the next subsections.

3.1 Architecture

Starting from the architecture proposed by Labrinidis et al. [9], we propose to add a new component to this architecture. This component, that we call Pre-fetching engine, is responsible for analyzing web log files and the graph that depicts the association between webviews and web pages. The main objective of this component is to find a so-called good set of webviews to materialize and to communicate them to the asynchronous cache module to readjust and to update types of webviews. Thus, we obtain the architecture given by Figure 1.

3.2 Proposed data components and their representations

In this subsection we describe the components of the proposed solution and we give the means that we will use to represent these components.

3.2.1 Data components

In our approach, we consider three types of data components: webviews, web pages and web log files.

• webviews: Labrinidis et al. [8] defined webviews as HTML or XML fragments. webviews are simply parts of a web page. They are usually generated by "wrapping" Database query results (views) with HTML for-
matting commands or XML semantic tags. Webviews can be formed from any type of database query. Challenger et al. [1] deals with fragments that typically represent parts of web pages that change together. As far as we are concerned, a webview is an HTML or XML fragment, which is a part for at least one web page, formed from any type of database query or from another webview. Figure 2 gives a simple of posted webviews and web pages where $P_1, P_2, P_3, P_4$ are web pages, and $C_1, C_2, C_3, C_4$ are webviews contained in web pages.

- Web pages: Challenger et al. [2] defined a web page by a complete entity which may be served to the client.
- Web log files: these files store the behavior of users when he access to the web. These files record in a chronological order the different access of users to the web. Every line corresponds to an user access to a URL. The usual formats of these files are the Common Log Format (CLF), and the extended version of CLF (ECLF). A line in the ECLF format contains: the IP address or the user host name, the user id, the user login (if applicable), the date and the time of request, the operation type, the request resource name, the request status, the request page size, the user agent (browser and operating system), and, the referred URL requested by the user.

### 3.2.2 Representation of data components

To represent the data components we will use a graphical representation to highlight the relation between webviews and web pages, and a formal representation for web log files.

Remembering that the association between data components is described as follows: A web page $P_k$ is considered to be derived from webview $W_j$, if $P_k$ contains $W_j$.

For that, we will use Object Derivation Graph (ODG) [1], which is a directed acyclic graph, where nodes correspond to webviews or web pages. This graph has mainly two types of edges. The first one is an inclusion edge indicating that an object embeds a webview. The second one is a link edge, indicating that an object contains a hypertext link to another object.

![Figure 1: Proposed architecture](image)

To illustrate the use of this graph, the ODG corresponding to Figure 2 is given by Figure 3.

For representing web log files we use the formal representation of this kind of files defined by Tanasa in [13]. This formal representation will be used in the next sections.

### 3.3 Selection of webviews to materialize

We define the problem of selecting webviews as follows: Using knowledge about users access pattern, find webviews to be materialized so that the satisfaction of users is optimal.

To achieve this, we will use WUM techniques. Our aim is to define a good set of webviews to materialize to be near the optimal. In the sequel section, we present an overview about WUM techniques.

#### 3.3.1 WUM overview

The term WUM was introduced by Cooley et al. [11] when a first attempt of taxonomy of web mining was done. In particular they defined web mining as the discovery and analysis of useful information from World Wide Web of large Web data repositories in order to produce results that can be used in analysis. WUM is mainly based on parsing Web server log files. Regarding the services offered by WUM techniques [10], we keep improving the server performance by caching dynamic web pages. But, the main problem of using caching techniques for dynamic web contents is the coupling of serving access requests and the handling of updates. Hence, an update operation invalidates a cached object and needs to recompute it on the next access request. That’s why, view materialization can solve this problem since it de-
Figure 3: Example of Object Derivation Graph

couples the serving access request from the handling of the updates [9].

Cooley and Mobasher in [3] divide the WUM process in three main steps: preprocessing, pattern discovery and pattern analysis. Details about this process can be founded in Srivastava et al. [12].

3.3.2 Data preparation step

In this subsection we briefly describe the data preparation step and we introduce the integration of webviews into transactions to propose a novel data preparation step adapted to the problem of the selection of webviews to materialize. Figure 4 gives a description of this proposed step.

Figure 4: Data preparation step for the problem of selection of webviews to materialize

The goal of a preprocessing step is to generate an ad-hoc formatted data set from information stored in a web log file. This step facilitates the process of knowledge discovery. The result of this step is the identification of users and their transactions. The reader can refer to Tanasa [13] for details. Thus, the preprocessing step gives as result a set of n web pages $P = \{P_1, P_2, \ldots, P_n\}$, and a set of m user transactions $T = \{t_1, t_2, \ldots, t_m\}$, where each $t_i \in T$ is a subset of $P$. Conceptually speaking, we can view each transaction as a sequence of ordered pairs: $t = \langle (p_1, \omega(p_1)), (p_2, \omega(p_2)), \ldots, (p_n, \omega(p_n)) \rangle$, where $\omega(p_i)$ is the weight associated to page $p_i$ in transaction $t$. We choose to represent these weights in a binary manner, to note the existence or non-existence of a page in a transaction. For the processing step, Cooley mentioned that [12] association rules may serve as a heuristic for prefetching documents so as to reduce user latency, when we want to load a page from a remote site. By these rules, and having the current selected pages by one user, the system will recommend the page that can be solicited later having the historical of navigation of users. In our case, we want to have a set of selected webviews that are frequently solicited by multiple users. Thus, we find that we can stop at the generation of frequent itemsets. We will describe the procedure that we will adapt in their generation in the next subsection. Generally, the input for these rules can be represented by a binary relation $R_1$ defined over the couple $(T, P)$, where $T$ is the set of transactions and $P$ is the set of web pages. $(T, P) \in R_1$ if and only if the transaction $t \in T$ contains the page $p \in P$. But, in our case, the information concerning the presence or not of webviews into the transaction must appear. For that, remembering that in practice, there are thousands of web pages in a web site, with dozens of Webviews on each page. There is also a significant amount of webviews sharing a subset of web pages. Thus, to integrate webviews into transactions, we use the Object Derivation Graph described before. This graph allows to depict the second association $R_2$ between web pages and webviews, defined over the couple $(P, W)$ where $P$ is the set of web pages and $W$ is the set of webviews. $(p, w) \in R_2$ if and only if the web page $p \in P$ is composed by the webview $w \in W$. Having the two binary relations $R_1$ and $R_2$, we define a new relation $R$ that is the composition of $R_1$ and $R_2$, $R_1 : T \rightarrow P$; $R_2 : P \rightarrow W (\forall (x, y) \in T \times W)$, $(xRy) \iff \exists z(xR_1z \land zR_2y)$.

To illustrate our proposed approach, we consider four web pages associated to a web page: $P_1$, $P_2$, $P_3$ and $P_4$. After parsing the web log file we can get the transactions represented by Table 1.

Table 1: Table of transactions

<table>
<thead>
<tr>
<th></th>
<th>$P_1$</th>
<th>$P_2$</th>
<th>$P_3$</th>
<th>$P_4$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$T_1$</td>
<td>+</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>$T_2$</td>
<td></td>
<td>+</td>
<td></td>
<td></td>
</tr>
<tr>
<td>$T_3$</td>
<td>+</td>
<td></td>
<td>+</td>
<td></td>
</tr>
<tr>
<td>$T_4$</td>
<td>+</td>
<td></td>
<td></td>
<td>+</td>
</tr>
<tr>
<td>$T_5$</td>
<td></td>
<td>+</td>
<td>+</td>
<td></td>
</tr>
<tr>
<td>$T_6$</td>
<td></td>
<td></td>
<td></td>
<td>+</td>
</tr>
</tbody>
</table>

Remembering that the different webviews of these pages are depicted in Figure 2. Thus, the relation between the set of web pages and the set of webviews, deduced from the ODG graph, is represented by the binary Table 2.

Finally, the composite relation, which represents the in-
Table 2: Table of webviews and web pages

<table>
<thead>
<tr>
<th>C1</th>
<th>C2</th>
<th>C3</th>
<th>C4</th>
</tr>
</thead>
<tbody>
<tr>
<td>P1</td>
<td>+</td>
<td></td>
<td></td>
</tr>
<tr>
<td>P2</td>
<td>+</td>
<td>+</td>
<td></td>
</tr>
<tr>
<td>P3</td>
<td>+</td>
<td></td>
<td>+</td>
</tr>
<tr>
<td>P4</td>
<td>+</td>
<td>+</td>
<td>+</td>
</tr>
</tbody>
</table>

Table 3: The input for the processing step

<table>
<thead>
<tr>
<th>C1</th>
<th>C2</th>
<th>C3</th>
<th>C4</th>
</tr>
</thead>
<tbody>
<tr>
<td>T1</td>
<td>+</td>
<td>+</td>
<td></td>
</tr>
<tr>
<td>T2</td>
<td>+</td>
<td></td>
<td></td>
</tr>
<tr>
<td>T3</td>
<td>+</td>
<td>+</td>
<td>+</td>
</tr>
<tr>
<td>T4</td>
<td>+</td>
<td>+</td>
<td></td>
</tr>
<tr>
<td>T5</td>
<td>+</td>
<td>+</td>
<td>+</td>
</tr>
<tr>
<td>T6</td>
<td>+</td>
<td>+</td>
<td>+</td>
</tr>
</tbody>
</table>

put for the generation of frequent itemsets is represented in Table 3. This binary relation depicts association between transactions and webviews.

### 3.3.3 Processing step

For the processing step adapted to the problem of selection of webviews, we rely on the generation of frequent itemsets. In our case, a frequent itemset means a set of webviews that are frequently used together in multiple transactions. To generate these sets, there exists at least three possibilities: the generation of frequent itemsets, closed frequent itemsets, or maximal itemsets. A frequent itemset is one that occurs in at least a user-specified percentage of transactions. A frequent itemset is closed if it has no superset with the same frequency [15]. A frequent itemset is called maximal if it is not a subset of any other frequent itemset. Having these three types of generated sets, and the fact that we are interested in mining webviews that are solicited frequently and that covers the preferences of all the users, the set of items (webviews) formed by the union of either frequent, closed or maximal itemsets can give a solution to the problem. Since by the given definitions of the three types of patterns we can deduce that the union set of each one of them gives the same set of items. Thus we can choose between the three possibilities.

As a solution we can keep the union of the items based on closed itemsets. We first exclude the union of items by applying the generation of frequent itemsets because in many applications (especially in dense data) with long frequent patterns enumerating all possible \(2^{m-2}\) subsets of a \(m\) length pattern \((m\) can easily be 30 or 40 or longer) is computationally unfeasible [15]. We then exclude the union of items by applying the generation of maximal because experiments show that the execution time of the existing implementations for the generation of maximal patterns is definitely higher than the execution time of the existing implementations for the generation of closed itemsets. Remembering that our objective is to reduce the initial set formed by all the webviews that constitute one web site by selecting the best set of webviews that will be materialized under a space constraint. One can think that the union gives the set of all items, and no selection on the initial set of webviews is made, but we found as the support is increased, the set formed by the union of itemsets decreases by size and than initial set of webviews is reduced by size (see Figure 5). More experiments must carried out to decide of the best support that will be given to the implementation that generates frequent closed itemsets.

Thus to illustrate this, we consider an experimental web site with 30 Web pages and 63 webviews. After parsing the web log file and applying the pretreatment associated, we generate 74 transactions by webviews. We then apply Charm algorithm that generates closed frequent itemsets [16]. The input is the transactions by the webviews and the support (in our case is equal to \(1/3\)). Then, we apply the following algorithm that generates recommendations to the asynchronous cache. The input for this algorithm is the set of frequent closed itemsets (setfreclo) generated by Charm. Maxsize and maxup are two fixed specified threshold by an user representing respectively the maximal size of the asynchronous cache and frequency of updates that must not exceeds one webview to be materialized. Freup of one webview represents the frequency of update of this webview. The result of the algorithm webviewsmat will be the set of webviews recommended by the pre-fetching engine and communicated to the asynchronous cache.

**Figure 5: Variation of the number of webviews generated by the union compared to the support**

```
Algorithm Recommendation (setfreclo, maxsize, maxup)
For each freclo of setfreclo do
  For each webview of freclo do
    If size(webview) < maxsize and
      freup(webview) < maxup then
      webviewsmat = webviewsmat \( \cup \) webview
  Return webviewsmat.
```

```
4. CONCLUSION AND FUTURE WORKS

In this paper, we proposed a new approach for selecting a set of webviews to be materialized. This approach is based on WUM techniques. We integrate webviews into the transactions in the preprocessing step. We use then the generation of closed frequent itemsets to filter the set of the most solicited webviews to be materialized under a size constraint. Implementation efforts are carried out in order to test the proposed prototype for materialization on real web site and to compare its performance with existing methods. Moreover, research efforts are undertaken to take into account the aspects of the update propagation of the materialized webviews and the algorithm of placement to adapt for the proposed prototype.

5. REFERENCES