Hybrid Approach for the Maintenance of Materialized Webviews

Houssem Saadi
Faculté des sciences de Tunis 2092 Manar Tunisia, saadi.houssem@gmail.com

Ali Ben Ammar
Institut Supérieur d’Informatique et de Gestion de Kairouan 3100 Kairouan. Tunisia, ali.benammar@isd.rnu.tn

Abdelaziz Abdellatif
Faculté des sciences de Tunis 2092 Manar Tunisia, abdelaziz.abdellatif@fst.rnu.tn

Follow this and additional works at: http://aisel.aisnet.org/amcis2010

Recommended Citation
http://aisel.aisnet.org/amcis2010/3

This material is brought to you by the Americas Conference on Information Systems (AMCIS) at AIS Electronic Library (AISeL). It has been accepted for inclusion in AMCIS 2010 Proceedings by an authorized administrator of AIS Electronic Library (AISeL). For more information, please contact elibrary@aisnet.org.
Hybrid Approach for the Maintenance of Materialized Webviews

Houssem Saadi  
Faculté des sciences de Tunis 2092 Manar Tunisia  
saadi.houssem@gmail.com

Ali Ben Ammar  
Institut Supérieur d’Informatique et de Gestion de Kairouan 3100 Kairouan. Tunisia  
ali.benammar@isd.rnu.tn

Abdelaziz Abdellatif  
Faculté des sciences de Tunis 2092 Manar Tunisia  
abdelaziz.abdellatif@fst.rnu.tn

ABSTRACT
In this paper, we have developed an approach to specify when updating materialized webviews. A webview is a web page that is automatically constructed from a structured database. We have introduced a new update policy called “early on-demand” update which is based on the user preferences. Then we have combines this policy with the on-demand one to update the materialized webviews. Our experiments showed that the proposed hybrid approach guarantees strong consistency of data and allows reducing latency of updating webviews. In addition, they prove that our solution decreases the server overload (access+ update cost) significantly better that the on-demand and the immediate policies.

Keywords (Required)  
Webview Maintenance, Update Policy, Immediate, On-demand, Early On-demand, Data-Intensive Web Site.

INTRODUCTION
Materialized webviews are efficiently used for accelerating data access in DIWS (Data Intensive Web Sites) (Labrinidis and Roussopoulos, 2001; Labrinidis and Roussopoulos, 1999; Labrinidis and Roussopoulos, 2000; Bouras and Konidaris, 2002). The term webview means web page that are automatically constructed from base data using a program or a DBMS query. The materialization technique consists in computing webviews and storing them. Having a webview materialized can potentially give significantly lower query response times. In the other hand, any change on the data source will have to be reflected on the materialized webviews. The first issue is thus to select an appropriate set of materialized webviews that minimizes query response time and data maintenance cost. The second issue is to use an appropriate maintenance approach that reduces the maintenance cost of materialized webviews.

In general, a maintenance approach specifies how (manner) or when (period) to update the materialized webviews. The first task specifies how to propagate to the materialized webview any changes made to the sources over which the webview is defined. In this case, the two possibilities are:

• The propagation of the changes incrementally, i.e., to compute the changes needed in the materialized webview and to effect only those changes;
• The re-computation of the entire webview every time a change has been made that might affect it.

The second task specifies when to react to the update event of source data. According to (Engström, Chakravarthy, Lings, 2002), the most known policies are:

• Immediate: updates to the source databases are directly reflected on the materialized webview. This policy guarantee a strong consistency, i.e., the webview will eventually reflect the final state of its sources.
• Periodic: the maintenance of the materialized webview is performed on a regular basis (once a second, once an hour, etc.). A webview may become inconsistent with its base data, representing a snapshot of the sources.
On-demand: the maintenance is performed when the materialized view is queried. This policy results in latency of going to the DBMS.

In this paper, we introduce the “early on-demand” policy which means applying the on-demand policy not when the webview is queried but when the webview is considered to be probably visited the next short period. We use the user preferences to select the set of webviews which will be probably used for the next period. Then, we update the selected materialized webviews. We call this update policy “early on-demand”. However, if the user queries a no selected webview, this later will be updated on-demand. Our goal is to reduce both the maintenance frequency of materialized webviews and the latency of going to the DBMS. Our contributions are as follows:

- The use of user preferences to specify the webviews that should be early updated;
- Combine the “on-demand” and the “early on-demand” policies for the maintenance of materialized webviews.

The remainder of the paper is organized as follows. In section two we give an overview of related works. In section three we present the state of art of the materialized webview maintenance. In section four, we present our approach for the maintenance of materialized webviews. In addition, we describe the results of our experimentations. Finally, we conclude this work in section five.

RELATED WORK

The problem of maintenance has been widely addressed for materialized views in data warehouses environment (Engström, Chakravarthy, Lings, 2003(a); Engström, Chakravarthy, Lings, 2002; Engström, 2002; Engström, Chakravarthy, Lings 2003 (b)). The mains ideas of these works are interested in the specification of the appropriate maintenance policies. However there are few works addressing the update of materialized webviews on the web. In (Qu, Labrinidis, 2006), the authors address the problem of prioritizing the scheduling of updates over queries. They use a twolevel scheduling scheme that dynamically allocates CPU resources to updates and queries according to user preferences. In(Zhang, Qin, 2005), the authors propose the State Transfer Graph (STG), a general paradigm to describe the behaviors of the webviews. A state describes a specific status of a webview, while a state transfer graph is an abstraction model to describe how a webview moves among these states. Therefore, a concrete maintenance approach can be drawn from a given STG. In addition, they present two particular methods named MEDI and VMF in this paper. There are three and five states for webviews in these two methods, respectively. They show that when the web changes rapidly, the MEDI approach is recommended. However, when the web keeps its changing trend and the queries are also increasing expeditely, the VMF method is recommended. In (Sanderson, Dumais, 2007), the user preferences are studied in order to cluster queries or to accelerate response time of search queries. We are not aware of any work addressing the problem of combining update policies based on user preferences.

STATE OF ART

In DIWS environment, when data on the sources change, update operations must be performed to refresh the materialized webviews. The two issues at hand are how to perform the refresh operation, and when to refresh the materialized webviews. These issues are previously (Engström, Chakravarthy, Lings, 2003(a); Engström, Chakravarthy, Lings, 2002; Engström, 2002; Engström, Chakravarthy, Lings 2003 (b)) addressed in data warehouse environment, i.e., for the maintenance of materialized views.

The solution of the issue of how to perform the refresh operation on materialized webview may be either re-computing the entire materialized webview from scratch or computing only the changes (incremental update) to the materialized webview. The simple solution is to re-compute the materialized webview but it results in a high maintenance cost.

The when to refresh issue, is closely associated with the overhead that the webview update algorithm places on the web server. In this context and according to (Engström, Chakravarthy, Lings, 2002), the essential maintenance policies are:

- Immediate maintenance: Maintenance is performed immediately when updates occur;
- Periodic: Maintenance is performed on a regular basis (once a second, once an hour, etc.)
- On-demand: Maintenance is performed when a webview is queried.

The policy selection is highly dependent on two user requirements:

- Quality of service (response time): it is the delay between submission of a query and the return of its result;
- Quality of data (staleness or freshness): The freshness level of webview data. It can be defined as the difference in time between when results are returned and the latest time when the webview reflected the source.
The periodic and on-demand policies are known as deferred maintenance. They result in lowest quality of data and worst quality of service. They are less used on the web. However, the immediate maintenance increases the server load, i.e., decreases the quality of service, because every update to the base data leads to a webview refresh but it guarantees a worst quality of data.

The majority of maintenance approaches address the problem of when to update. In (Zhang, Qin, 2005), the authors propose an approach to minimize the maintenance cost. They use the term state and the tool state transfer graph to reduce the frequency of immediate update of materialized webviews. A state describes a specific status of a webview, while a state transfer graph is an abstraction model to describe how a webview moves among these states. A webview transfers between the following states:

- **Active**: The webview is always up-to-date and can be used directly. To keep the webview always active, any update to the base data will lead to an immediate refreshing of this webview;
- **Sleeping**: The webview does not respond to any updates at all and will only be evaluated on-demand;
- **Median**: A webview in state median is up-to-date and can be used directly. Thus the query latency can be reduced partially.
- **Semi-Active**: The main difference between active and semi-active is that a less active webview usually cannot transfer to active state directly, unless it obtains enough activity.
- **Semi-Sleeping**: The main difference between sleeping and semi-sleeping is that a sleeping webview will not transfer to semi-active state directly. It must go through the Median state.

For the minimum-update approach, there are two states for each view: Active and Sleeping. A webview transfers between these two states.

Based on these states, the authors specify two particular methods named MEDI (MEDian) and VMF (View Maintenance Based on A Five-State Transfer Graph). There are three and five states for webviews in these two methods, respectively. In the MEDI approach, when a webview, in state “active”, receives an update request, it will be re-computed immediately, thereby avoiding the latency for the next query. If the next request happens to be a query, they will benefit from this pre-computation. However, the activity of this webview will decrease, hence its state transfer to “median”. In the VMF approach, each state has its attributes, and the main one is the activity. When a webview receiving a query request, its activity increases. In contrast, its activity decreases when receiving update requests. In general, the activities of state “active”, “semi-active”, “median”, “semi-sleeping” and “sleeping” decrease in turn. The first three states are up-to-date. They show that when the web changes rapidly, the MEDI approach is recommended. However, when the web keeps its changing trend and the queries are also increasing expeditely, the VMF method is recommended.

In (Qu, Labrinidis, 2006), the authors propose a unifying framework for specifying quality of service (QoS) and quality of data (QoD) requirements, which they call Quality Contracts (QCs). Essentially, the user specifies the amount of “worth” to him/her for the query to have a certain QoS and QoD. In this way, users can specify the relative importance of QoS over QoD and can also specify the relative importance among their different queries. The system, on the other hand, can infer the relative importance of different users’ queries and allocate its resources to maximize “profits”, and as such, maximize user satisfaction. With QCs, they cast the problem of scheduling queries and updates into the problem of optimizing the total profit for the system. They present a novel two-level scheduling scheme for scheduling updates and queries under quality contracts. The basic idea behind the proposed scheme is in deciding on the allocation of resources between queries and updates using the expected “profit gain” to the system: executing queries contributes to QoS (and QoD), whereas executing updates contributes to QoD. They propose, QUTS (short for Query-Update Time-Sharing) which adapts resource allocation by monitoring the achieved user satisfaction in QoS/QoD and comparing it to the best case policy. So, their approach results in applying immediately or differed update of materialized webviews.

**HYBRID APPROACH FOR THE MAINTENANCE OF MATERIALIZED WEBVIEWS**

**Approach principle**

In this approach, we apply the on-demand policy to update the materialized webview. But, for the webviews which will be probably queried the next short period, we apply the “early on-demand” policy. The “early on-demand” policy means that the webview is updated on-demand not because it is queried but because it is estimated to be accessed in the short next period. This estimation is extracted from the navigation historic of the web site. So, when a user accesses the web site, we must select the set of materialized webviews which will be probably used the next requests and then we should update them. The
aim is to benefit from this pre-computation and avoid the latency of updating the query results at the moment of access. To perform this update, we introduce the following concepts:

- Transfer matrix (TM): a matrix used to represent the probabilities of transfer (navigation) between each couple of web pages. TM(i,j) is the percentage of access to the web page Pi which are followed by an access to the web page Pj, i.e.,

\[
TM(i,j) = \frac{\text{number of navigation from } P_i \text{ to } P_j}{1 + \text{number of access to } P_i}
\]

- The reason for the +1 in the denominator of this equation is to guarantee a valid probability value, even when there are no access operations to Pi.
- Support probability (SP): a level of probability from which a transfer is considered valid. The valid transfers result in a set of materialized webviews which will be probably used for the next requests.
- User position (UP): indicates the position of the user on the structure of the web site. It specifies the current web page.

Essentially, our system performs the following tasks to update the materialized webviews:

1. Update the transfer matrix: at the end of each user session, the system updates the transfer matrix;
2. When a user visit a web page Pi, the system affect the value Pi to UP and then selects the set of materialized webviews which will be probably accessed the next requests. Let MW be this set and W be the set of all materialized webviews. So,

\[
MW = \{w_j \in W / TM(i,j) \geq SP\}
\]
3. Apply the “early on-demand” policy to update the webviews of MV which are stale, i.e., they are not up-to-date;
4. If the next user request happens to be in \(W - MW\) apply the on-demand policy to update the queried webview.

This approach guarantees both the quality of service and the quality of data. It allows providing fresh query results with infrequent updates.

Experiments

In this section we present results from our experiments using the TPC-W benchmarking (TPC-W Benchmark Specification). The speed in which accesses can be processed, the incoming update stream and the speed in which updates can be performed are inputs to the simulator. The duration of the streams was 600 seconds or 600,000 milliseconds (our simulator’s internal clock run at milliseconds). We implemented our approach as well as the on-demand and the immediate approaches in our tests.

Figure 1 presents the results of our first test which concerns the maintenance cost of the three approaches.

![Figure 1: Maintenance cost of three update policies](image)

It is clearly that the on-demand policy provides the least maintenance cost. This is because the update operations are less frequent. For the same reason, our approach is more beneficial than the immediate policy.
Figure 2 presents the results of our second test which concerns the access cost of the three approaches.

![Access cost](image1.png)

**Figure 2: Access cost of three update policies**

The second test shows that our hybrid approach allows reducing the concurrency operations of access and maintenance. Consequently, it is more beneficial than the on-demand update policy.

Figure 3 illustrates the profit produced by our hybrid approach. It presents the results of our third test which concerns the total cost (access + update) of the three approaches.

![Total cost](image2.png)

**Figure 3: Total cost of three update policies**

In summary, our approach is more beneficial that the two well known update policies. It allows reducing update frequency and hence the maintenance cost. In addition, it allows avoiding latency of updating the queried webviews.

**CONCLUSION**

In this paper, we have proposed a hybrid approach for the maintenance of materialized webviews. This approach combines two update policies: the “on-demand” and the “early on-demand”. We have introduced the later policy which means applying the on-demand policy not when the webview is queried but when the webview will be probably visited the next short period. The choice of the appropriate policy is performed according to the user preferences, i.e., that is the frequency of navigation between web pages.

Our solution guarantees strong consistency of data and allows reducing latency of updating webview. In addition, our experiments showed that this solution decreases the server load (access+update cost) significantly better that the on demand and the immediate policies.

**REFERENCES**


