Forms of Data Materialization in Data-Intensive Web Sites

Ali Ben Ammar[j], Abdelaziz Abdellatif[jj], and Henda Ben Ghezala[jjj]

[j]Institut Supérieur de Documentation, Manouba, 2010, Tunisia
[jj]Faculté des Sciences de Tunis, Tunis, 2092, Tunisia
[jjj]Ecole Nationale des Sciences de l'Informatique, Manouba, 2010, Tunisia

Summary
Data-Intensive Web Sites (DIWS) use a large volume of data which are stored in several databases. They serve to answer dynamic queries. That is the queries for which the response may be changed from a period of time to another or from a user to another. To carry out these queries, the required data must be generated each time from the sources. In general, the sources are distributed and heterogeneous databases. So, such queries are characterized by a highly response time that increase the server load. In this context, several tools and approaches are proposed to improve the server response time. The data materialization, which has been applied in databases and data warehouse, is a good solution to answer speedily queries in DIWS. In this paper, we will present the different forms of data materialization in DIWS. That is the data materialization in structured format, in semi-structured format, or in these two formats. The structured format is called views and the semi-structured one is called webview. A webview is a HTML document whose content is dynamically extracted from structured databases. We will compare the profits produced by the materialization of data in these three forms. Our experiments demonstrate that the best profit is produced by the combination of the structured and the semi-structured formats and that the materialization profit depends on the number of accessed sources.

Key words:
Web data forms, Materialization, Data-Intensive Web Sites, Views, and Webviews.

1. Introduction

The data on the DIWS are stored in structured databases. They are extracted and transformed into web pages to satisfy some dynamic queries. E-commerce, voyages, stock exchange, and meteorology sites are examples of DIWS. They need the following tasks to carry out user's queries: data extraction, data integration when the sources are heterogeneous, and web page construction. So, for a DIWS with a high number of sources, the data access will be more complex. It makes the web server more loaded and in need for more resources. Consequently, several techniques have been developed to meet the demand for faster and more efficient access to the DIWS. The majority of these techniques have been applied in other domains like databases and operating systems. Among them a main role is acquired by the replication, refreshing, caching, and materialization of data. The last technique consists of storing some query results to avoid their repetitive research. The stored results are those rarely updated and frequently asked. The view materialization has been used efficiently in data warehouse because of the high volume of data and the complex queries [8, 9, 11]. On the web, this technique is used to improve the server load by storing data in two formats: (i) views of structured data on the source databases or (ii) HTML documents on the server [1, 2, 3, 4, 5, 6]. The purpose is to avoid generating and transforming data when there is no update on the sources. This will improve query response time and provide good data quality that is data freshness. In this paper, we will compare three materialization profits that may be provided for a DIWS. The first corresponds to the materialization of data using the structured format that is using views. The second corresponds to the materialization of data using the semi-structured format that is using HTML documents called webviews. The Third correspond to the materialization of data using the two formats that is views on the sources and webviews on the server. Our experiments show that the latest profit is the best.
The data materialization is a good solution for improving DIWS server load. This is because of the high number of sources that may be accessed by a DIWS. The profit of such solution will be demonstrated in our experiments. So, our contributions in this paper are twice: (i) compare the different forms of data materialization, and (ii) prove the need for the materialization as a solution to improve DIWS response time.
This paper is organized as follow: section 2 presents the related works. Section 3 describes in more detail the data materialization on the web. Section 4 we present our solution and experiments. Section 5 is the conclusion.

2. Related Works

Recently, there has been a lot of interest on how to improve web server response times. Several approaches and tools have been developed for this reason. The web caching, the data replication and the processing distribution are the techniques that interest the researchers. The data materialization on the web is addressed by a few works [1, 2, 3, 4, 5, 6, 7]. The majority of these works concern the materialized webview selection. Only in [5], the authors discuss the data materialization forms. They compare two data materialization forms: the materialization of views and the materialization of webviews. We are not aware of any work addressing the combination of these two forms of data materialization. That is considering the view materialization on the source databases and the webview materialization on the web server.

3. Data materialization on the web

Data-Intensive Web Sites (DIWS) provide access to a large number of web pages whose content is dynamically extracted from structured databases [12]. Today, they become necessary for allowing some e-commerce tasks or accessing dynamic information. Their architecture includes a database management system (DBMS) layer, a site server layer and the client. Thus, a new kind of pages, dynamically generated, and a new architecture were born. We have no more the traditional couple of a web client and a web server, but a third part is added, the application program, running on the web server to receive the request from the client, retrieve the relevant data from the database and then pack them into HTML or XML format. Newspaper sites and shopping ones are examples of such architecture. Several tools and approaches for developing such systems are presented in [12, 10]. For each kind of application, a set of web development tools are specified. The performance problem of DIWS lies in addressing the latency reduction of page produced by the site and the quality of data presented to the clients. Firstly, because returning web page may require costly interaction with the database system. So, the net effect of this situation is network congestion, high client perceived latency, web server overload and slow response times for web severs. Secondly, because the quality of data is of crucial importance, especially for applications that must always serve fresh data (e.g. providers of stock prices, sports scores).
The data materialization on the web is used as a technique to improve the server load. It consists of storing data in two formats: (i) views of structured data on the source databases or (ii) HTML documents on the server [1, 2, 3, 4, 5, 6]. The aim is to avoid generating and transforming data when there is no update on the sources.
Similarly to traditional database views, the term webviews is used on the web to mean web pages that are automatically constructed from base data using a program or a DBMS query. The materialization approach consists in computing webviews and storing them. Having a webview materialized can potentially give significantly lower query response times, provided that the update workload is not heavy. Even if the webview computation is not very expensive, by keeping it materialized we eliminate the latency of going to the DBMS every time which could lead to DBMS overloading. Having a view materialized can also give significantly lower query response times. It allows eliminating the local data extraction every time.
The webview and view materialization approach is similar to that of view materialization in a data warehouse [8, 9, 11]. The main issues of the webview and view materialization approach are: how to select dynamically the appropriate objects (webviews and views) to be materialized, how to refresh them, how to distribute their storage over several servers and how to exploit them. The corresponding tasks to these issues constitute the management of materialized data. They are known as the selection, the maintenance, the storage and the use of materialized data.

4. Solution and experiments

4.1 Solution overview

The DIWS access includes three tasks: data extraction, data integration and webview construction. Materializing views on the source databases can eliminate the first task. Materializing webviews on the web server can eliminate the three tasks. What we propose in this work is to compare the profits produced by these two materialization forms and by their combination.

4.1.1 Definitions of execution and maintenance costs

- The execution cost of a query Q:
  CE(Q)=data extraction cost+data integration cost+webview construction cost

Here we extract data from base tables. In all the cost definitions, we eliminate the data access time which is negligible for the whole execution cost.

- The maintenance cost of the webview W, corresponding to Q: *CM(W)=data extraction cost+data integration cost+webview construction cost*

We suppose that for each update we recompute the webview. That is we execute the corresponding query Q.

- The maintenance cost of a view Vi, used to answer Q:
  CM(V_i)=data extraction cost

We suppose that for each update we recompute the view.

- The execution cost of Q when W is materialized:
  CE(Q, W)=0

- The execution cost of Q when VM, the set of Vi corresponding to Q, is materialized: *CE(Q, MV)= data extraction cost+data integration cost+webview construction cost*

Here we extract data from materialized views. We suppose that for each webview, it corresponds a set of materialized views that may distributed over several sources.

From the above definitions, we can conclude that CE(Q, MV)<CE(Q). This is because many table jointures are eliminated by materializing views.

4.1.2 *Definitions of execution and maintenance costs during a time period P*

In order to observe the server load during a time period P, we need the following parameters:

- n= frequency of a query Q
- v= number of views of MV
- m_i=update frequency of the source of a view Vi
- m=update frequency of W. $m \geq m_i$ because the webview update occurs when any materialized view is updated. That is it corresponds to the update frequencies of all the materialized views. But since some views may be updated at the same time or theirs updates may refer to the same transaction, $m_i \leq m \leq \sum_{i=1}^{v} m_i$. That is an update of a webview may refer to several updates, of materialized views, that occur at the same time. In our solution, we suppose that a webview is materialized only when its update frequency is less than its access one. That is when m is less than n.

Consequently,

- The execution cost of Q during P: $CE(Q,P) = n * CE(Q)$
- The materialization cost of W during P:

$$C_{mat}(W,P) = m * CM(W) = m * CE(Q)$$

Here we suppose that for each source update we must recompute W. The materialization cost measure the server load during P when W is materialized. In this case, we will have only the update cost of W because the execution cost of Q will be null.

- The materialization cost of MV during P:

$$C_{mat}(MV,P) = (\sum_{i=1}^{v} m_i * CM(V_i)) + n * CE(Q,MV)$$

In this case, we will have the update cost of the materialized views and the execution cost of Q when MV is materialized.

- The materialization cost of W and MV during P:

$$C_{mat}(\{W,MV\},P) = (\sum_{i=1}^{v} m_i * CM(V_i)) + m * CE(Q,MV)$$

In this case, we will have the update cost of the materialized views and the update cost of W when MV is materialized. W will be updated using the materialized views.

4.1.3 *Definitions of materialization profits during a time period P*

- The materialization profit of W during P:

$$\Pr_{mat}(W,P) = CE(Q,P) - C_{mat}(W,P) = n * CE(Q) - m * CE(Q)$$

- The materialization profit of MV during P:

$$\Pr_{mat}(MV,P) = CE(Q,P) - C_{mat}(MV,P) =$$
$$n * CE(Q) - ((\sum_{i=1}^{v} m_i * CM(V_i)) + n * CE(Q,MV))$$

- The materialization profit of MV and W during P:

$$\Pr_{mat}(\{W,MV\},P) = CE(Q,P) - C_{mat}(\{W,MV\},P)$$
$$= n * CE(Q) - ((\sum_{i=1}^{v} m_i * CM(V_i)) + m * CE(Q,MV))$$

4.2 Experiments

4.2.1 *Experiment details*

For these experiments we used a machine with AMD Athlon 64 Processor 3200+498 MHZ and 512 MB of memory. The web server used is Apache 1.3.33. The data server used is MySQL 4.1.9. We chose to work in a non distributed environment in order to eliminate the (uncontrollable) network latency. We used a voyage web site to define ours webviews and databases. This site makes access to five sources corresponding to five different air companies.

Our workload consisted of 30 webviews. The total access to the web server averaged to 5 requests per second. According to [5], this should correspond to a quite heavy load on the server, of about 0.4 million requests per day. We have considered four experiments. Each experiment was run for 10 minutes. In the first experiment all the webviews were kept virtual. In the second experiment all the webviews were materialized at the web server. In the third experiment all the views were materialized at the source databases. In the fourth experiment all the webviews and all the views were materialized respectively at the web server and the source databases. We suppose that for each webview, it correspond 5 views distributed over the accessed sources. In all the experiments the update and the access frequencies are chosen to be, respectively, 10 per webview and 100 per webview during the 10 minutes. The 10 updates are distributed over the 5 sources. That is we suppose that, during the 10 minutes, each source is averagely updated twice which correspond to the update frequency of each view.

4.2.2 Experiment results

The materialization costs of the different forms, observed during 10 minutes, are presented in the table1. The materialization cost includes the query execution cost and the update cost of the data materialization. In column, we varied the number of the accessed sources. In the first line, we measured the server workload when there is no data materialization. In the second line, we measured the server workload when we materialize only webviews. In the third line, we measured the server workload when we materialized only the views. In the fourth line, we measured the server workload when we materialize the views and the webviews. The variation of these materialization costs is represented in the fig.1.

Table 1: Materialization costs in seconds for the different materialization forms.

Materialization formats \ source number	1	2	3	4	5
No data materialization	60	130	180	200	237
Webview materialization	6	15	19	24	33
View materialization	30	60	80	100	130
View and webview materialization	2	3	5	10	12

The results presented in the table1 prove that the webview materialization is more profitable than the view materialization. This conclusion is found by the authors of [5]. Our contribution is the combination of these two formats. From table 1, we can deduce that our contribution always gives the smallest materialization cost. This is due to the fact that, with this materialization scenario, we will update only the materialized data. The extraction and the integration costs will be reduced in this scenario. The materialization costs of this scenario are less than the webview materialization ones. This is because the update frequency of some views is less than the update frequency of their corresponding webview. That is when a webview is updated, only some base views are updated. However the rest of the base views will be used to extract data rather than the base table. So, the data extraction cost will be reduced especially in the case of a high number of sources or a heavy table.

The fig. 1 shows that the materialization costs increase with the source number for the different materialization forms. The materialization costs produced by our contribution are the smallest ones for the different numbers of sources. In fig. 2, we present the variation of the materialization profits which are the differences between the server overload when there is no data materialization and the different materialization form costs. This fig. shows that (i) the best profit corresponds to our contribution and (ii) the materialization profits increase with the number of sources. The later conclusion proves the need for the materialization as a technique to improve the data access on the DIWS which are characterized by a high number of data sources.

5. Conclusion

In this paper we have shown that the data materialization is a good solution to improve the data access on the DIWS which are characterized by a high number of data sources. The combination of the view and the webview materializations is the scenario that always produces the best materialization profit. This is because the server workload will consist only of updating materialized data. That is the data extraction, data integration and webview construction costs will be reduced. In the future work, we will develop an approach to select the optimal objects (views and webviews) that should be materialized to improve the query response time. This selection must respect some resource constraints.

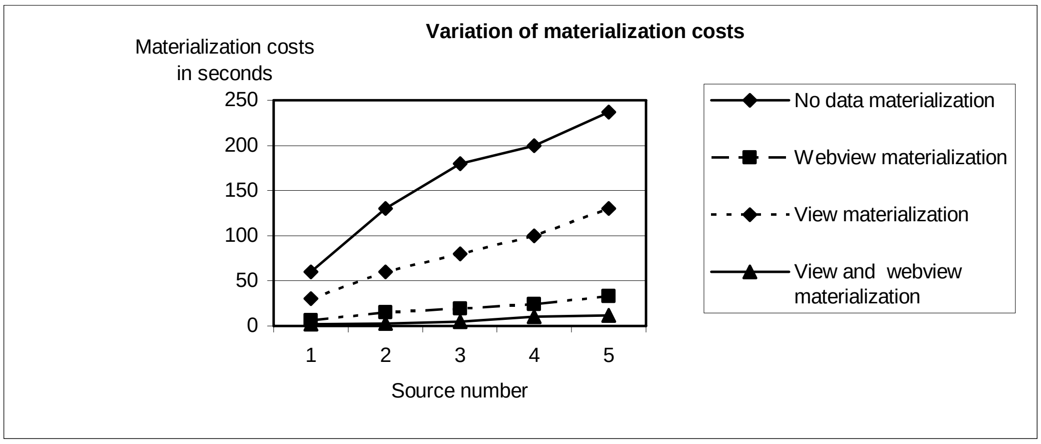

Fig. 1: Variation of materialization costs.

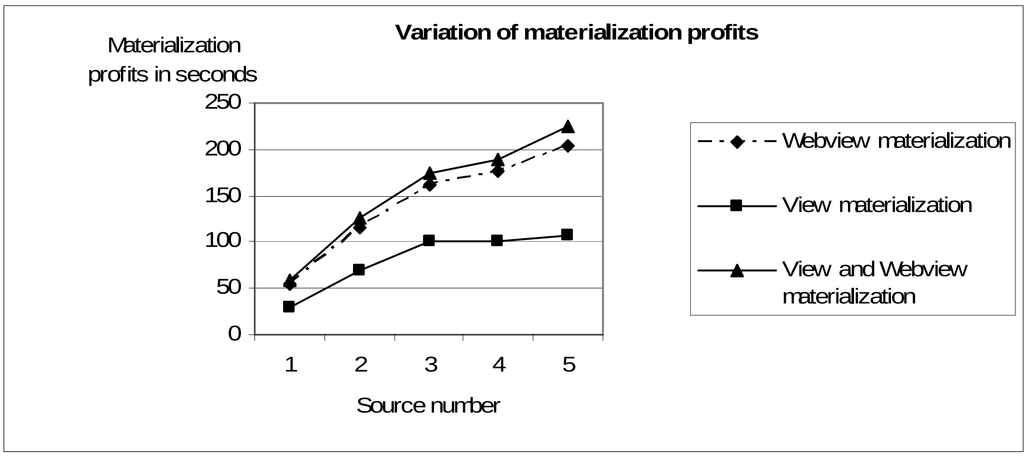

Fig. 2: Variation of materialization profits.

References

[1] A. Labrinidis, N. Roussopoulos. “Adaptive WebView Materialization”. WebDB'01: 85-90. 2001.

[2] A. Labrinidis, N. Roussopoulos. “Balancing Performance and Data Freshness in Web Database Servers”. VLDB'03: 393-404. 2003.

[3] A. Labrinidis, N. Roussopoulos. “On the Materialization of WebViews”. In ACM SIGMOD Workshop on the Web and Databases (WebDB '99): 79-84. 1999.

[4] A. Labrinidis, N. Roussopoulos. “Online View Selection for the Web”. Technical report. 2002.

[5] A. Labrinidis , N. Roussopoulos. “WebView Materialization”. Proceedings of the 2000 ACM SIGMOD International Conference on Management of Data. pp.367-378. May 15-18, 2000, Dallas, Texas, United States.

[6] B. Christos, K. Agisilaos. “Efficient Materialization Of Dynamic Web Data To Improve Web Performance”. 15th International Conference on Computer Communication (ICCC 2002) Mumbai. India, August 11-14 2002.

[7] B. Zhang, Z. Sun, and W. Jin. “Update of Materialized WebView”. IEEE International Conference on e-Business Engineering (ICEBE 2005). October 18-20, 2005, Beijing, China.

[8] H. Gupta, I. S. Mumick. “Selection of Views to Materialize Under a Maintenance-Time Constraint”. 7th International Conference on Database Theory Jerusalem, Israel, January 10-12, 1999.

[9] H. Gupta. “Selection of Views to Materialize in a Data Warehouse”. International Conference on Database Theory. Delphi, Greece, January 8-10, 1997.

[10] P. Fraternali. “Tools and Approaches for Developing Data-Intensive Web Applications: A Survey”. In ACM Computing Surveys Volume 31:3, pp 227-263. 1999.

[11] S. Iqbal, J. J. Bunn, H. B. Newman. “Distributed Heterogeneous Relational Data Warehouse In A Grid Environment”. Computing in High Energy and Nuclear Physics, 24-28 March 2003, La Jolla, California.

[12] Yagoub, D. Florescu, V. Issarny, and P. Valduriez. “Caching strategies for data intensive web sites”. In Proceedings of the VLDB 2000 Conference, pp 188-199. 2000.