Active Caching of On-Line-Analytical-Processing Queries in WWW Proxies

Thanasis Loukopoulos, Panos Kalnis, Ishfaq Ahmad and Dimitris Papadias

Department of Computer Science
The Hong Kong University of Science and Technology, Hong Kong
e-mail: {luke, kalnis, iahmad, dimitris}@cs.ust.hk

Abstract

The Internet is offering more than just regular Web pages to the users. Decision makers can now issue analytical, as opposed to transactional, queries that involve massive data (such as, aggregations of millions of rows in a relational database) in order to identify useful trends and patterns. Such queries are referred to as On-Line-Analytical-Processing (OLAP) queries. Typically, pages carrying query results do not exhibit temporal locality and, therefore, are not considered for caching at WWW proxies. In OLAP processing, this becomes a major hurdle as the cost of such queries is much higher than traditional transactional queries. This paper proposes a systematic technique to reduce the response time for OLAP queries originating from geographically distributed private LANs and issued through the Web towards the central data warehouse (DW) of an enterprise. An active caching scheme is proposed that enables the LAN proxies to cache some parts of the data, together with the semantics of the DW, in order to process queries and construct the resulting pages. OLAP queries arriving at the proxy are either satisfied locally or from the DW, depending on the relative access costs. We formulate a cost model for characterizing the latencies of these queries, taking into consideration normal Web access as well as analytical processing. We propose a cache admittance and replacement algorithm that outperforms a widely accepted caching algorithm.

1 Introduction

Caching has emerged as a primary technique for coping with high latency experienced by end-users in the WWW. There are four major locations where caching is performed: a) proxy at the front-end of a server farm [5]; b) network cache at the end-points of the backbone network [10]; c) LAN proxy [1]; d) browser. Although caching at these locations has been shown to significantly reduce the Web traffic [2], dynamically generated pages are not cacheable. Dynamic pages typically consist of a static part and a dynamic part (for example, query results from a database with a Web server linked to it).

On the other hand, the need for decision support systems has become of paramount importance in today's business, leading many enterprises to build decision support databases called data warehouses (DWs) [11]. Decision makers issue analytical (as opposed to transactional) queries that typically involve aggregations of millions of rows in order to identify interesting trends. Such queries are often referred to as OLAP (On-Line-Analytical-Processing). Users perceive the data of the DW as cells in a multidimensional data cube [12]. Fetching from the DW's sources the parts of the cube needed by queries and performing aggregations over them is an extremely time consuming task. A common technique to accelerate such queries is to precalculate and store some results. Such stored fragments are essentially parts of views in relational database terms; we will refer to their storage as materialization/caching of OLAP views. Most of the past work on view selection for materialization is limited to the central server.

The Web provides to geographically distributed clients, an easy method to access a central DW. An example is that of non-professional international investors who trade stocks in stock markets around the world. Since their queries are usually ad-hoc and driven by previous results (roll-up, drill-down), it is not possible for the data owner to provide a set of predefined query templates. Potential applications are not limited to the financial sector. Meteorological and environmental databases or other scientific information sources also have similar requirements. The problem was presented in [14] where the authors proposed a dedicated infrastructure of DBMSs that acts as a proxy server for OLAP data.

In this paper we deal with the problem of caching OLAP queries posed by ad-hoc, geographically spanned users, through their web browsers. Unlike the previous approach, however, we employ the existing proxy infrastructure and propose a method of caching both Web pages and OLAP query results in common proxy servers. Web pages carrying OLAP query results, hence abbreviated as WOQPs (Web OLAP query pages), are essentially dynamic pages, and are normally marked as uncacheable. This is not because their content changes frequently but due to the ad-hoc nature of OLAP queries (as it is unlikely that exactly the same query may be issued in the near future). Therefore, unless the caching entity is enhanced with query processing capabilities it is impossible to use a cached WOQP in order to answer future queries inquiring a subset of the cached results. We propose an active caching framework that enables the proxies to answer queries using the views cached locally and construct the WOQPs needed to present the results in the users' browsers. For tackling cache replacement issues we develop an analytical cost model and propose strategies that are experimentally proven to lead to high quality solutions. Although active caching has been employed before in answering transactional queries [19], to the best of our knowledge this is the first time that OLAP data are considered. The special case of OLAP involves unique challenges (for instance the results may vary in size by many orders of magnitude) and provides new opportunities for optimization (*e.g.*, the interdependencies of the views in a lattice).

The caching entities are assumed to be world spanned departmental LAN proxies of the company. This assumption is made for the purpose of illustration since a significant portion of the Web traffic in such environments is expected to be directed towards the central DW. Our work is applicable to the other caching points of the network, provided that significant amount of traffic towards the DW passes through them. The motivating principle is that caching/replication under restricted environments [18], [21] can considerably decrease the response time perceived by end-users.

The rest of the paper is organized as follows. Section 2

0190-3918/01 $10.00 © 2001 IEEE

provides an overview of OLAP queries and illustrates the lattice notion to describe OLAP views. Section 3 presents the proposed framework for caching OLAP queries in departmental LAN proxies. Section 4 deals with the caching and replacement strategies for OLAP views. Section 5 illustrates the simulation results. Finally, Section 6 discusses the related work and includes some summarizing remarks.

2 Overview of OLAP queries

DWs are collections of historical, summarized and consolidated data, originating from several different databases (sources). Analysts and knowledge workers issue analytical (OLAP) queries over very large datasets, often millions of rows. DW's contents are multidimensional and the typical OLAP queries consist of *group_by* and *aggregate* operations along one or more dimensions. Figure 1 depicts an example of a 2-D space with the dimensions being the customer's name and the product id.

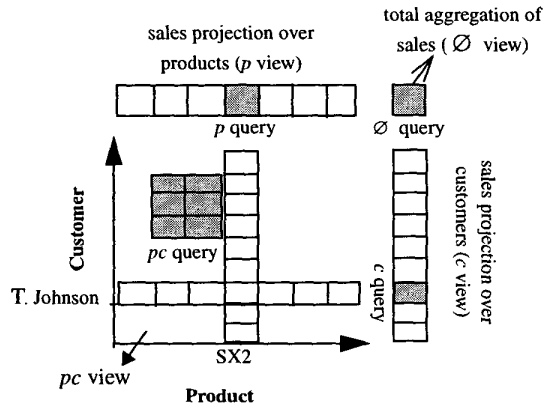


Figure 1: An example of OLAP queries in 2-D space

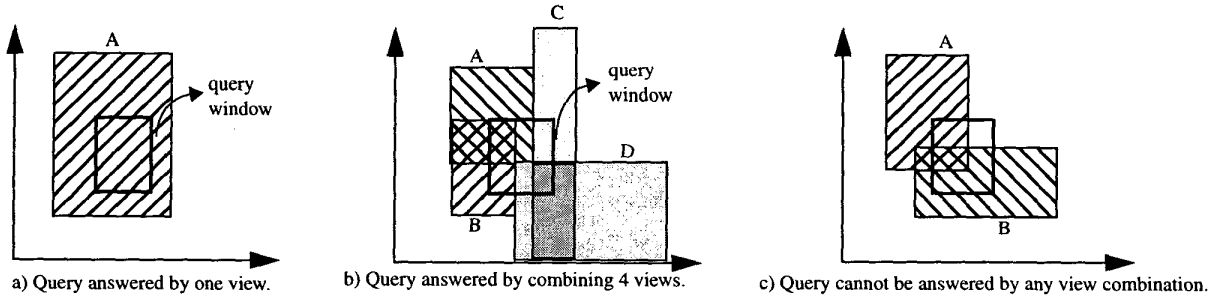
The value of each cell in the 2-D grid gives the volume of sales for a specific *<product_id, customer_id>* pair. An OLAP query could, for example, ask for the total volume of sales for the product SX2 or the customer T. Johnson, shown as shaded cells in Figure 1. It could also be a *group_by query* for 2 products and 3 customers as shown in the shaded rectangle, or an aggregation of total sales. A view is a derived relation, which is defined in terms of base relations and is normally recomputed each time it is referenced. A materialized view is a view that is computed once and then stored in the database. In the example of Figure 1 we might consider for materialization the results of the 4 described queries. The advantage of having some views materialized is that future queries can be answered with little processing and disk I/O latency. Moreover, queries asking for a subset of the materialized data may be answered by accessing one view, or through a combination of two or more views as shown in Figure 2.

In our 2-D example any rectangle in the plain can be a potentially materialized view. Due to the fact that OLAP queries are ad-hoc, stored fragments will most likely be able to only partially answer future queries, in which case we need to combine the results obtained by querying multiple stored fragments as shown in Figure 2(b, c). This approach though can be time consuming since all possible combinations of fragments may have to be considered for answering a query. Therefore, it is sound practice to consider whole views as the only candidates for materialized views [9], [12] and not fragments of them. In this paper we follow this approach. For instance in the example of Figure 1, the only candidates for materialization are the *p*, *c*, ∅ views, together with the whole plain (*pc* view). It is easy to see that under this strategy the total number of candidate views for materialization are 2^r where r is the number of dimensions.

Views have computational interdependencies, which can be explored in order to answer queries. A common way to represent such dependencies is the lattice notation. Skipping the formal definitions we illustrate the notion through the example of Figure 3. The 3 dimensions account for *<product, customer, time>*. A node in the lattice accounts for a specific view and a directed edge connecting two nodes shows the computational dependency between the specific pair of views, *i.e.*, the pointed view can compute the other, *e.g.*, *pc* can compute *p*. Only dependencies between views differing 1 level are shown in the lattice diagram (Figure 3(a)), *e.g.*, *c* can be derived from pcs but there is no direct edge connecting the two views.

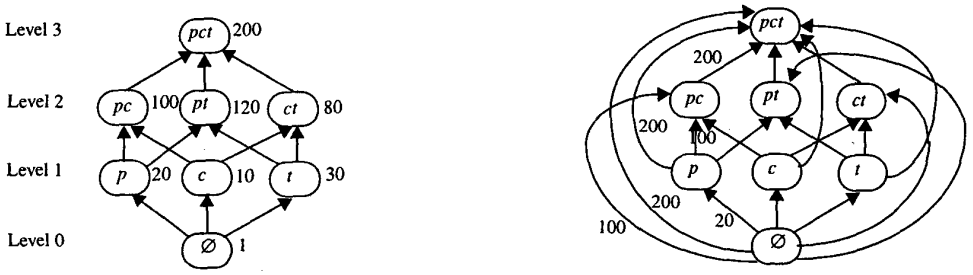
A query is answered by different views at different costs. A widely used assumption in the OLAP literature is that the cost for querying a view is proportional to the view size [12]. Figure 3(a) shows the associated query costs for a 3D lattice. We should notice that the costs increase as we move from a lower level to a higher level in the lattice. This is reasonable since higher views are normally larger. In Figure 3(b) we expand the lattice adding all the edges in the transitive closure and for each edge we attach the cost of computing the lower view, using the upper one. Again we should notice the relation of the computational cost to the view size, *e.g.*, deriving *p* view from *pct* incurs higher cost than computing *p* from *pc*, while the cheapest way to materialize ∅ view is to calculate it from *p* as compared to *pc* and *pct*.

Since recomputing views from the raw data is an expensive procedure, it is common practice that the central



a) Query answered by one view. b) Query answered by combining 4 views. c) Query cannot be answered by any view combination.

Figure 2: Using materialized views to answer queries



a) Query costs associated with each node

b) Costs for computing views associated with each edge

Figure 3: Lattice and expanded lattice diagrams for <p, c, t> dimensions with associated query and view computing costs.

DW always keeps the topmost view materialized, in order to be able to handle all OLAP queries [15]. We follow the same policy in the central DW but not in the proxies, since the size of the topmost view may be prohibitively large.

A well studied problem in the database community is the view selection under storage and update constraints (see Section 6), which can be defined as: given the query frequencies and the view sizes, select the set of views to materialize so as to minimize the total query cost under storage capacity constraints and with respect to an update window. The problem is solved with static centralized solutions that are inefficient in the Web environment. Our approach is fundamentally different since we consider a distributed environment where OLAP views are cached together with normal Web pages.

3 System Model

We consider an environment consisting of an enterprise with a central DW located at its headquarters and multiple regional departments having their own LANs. Each LAN is assumed to be connected to the Internet through a proxy server. Clients from the regional departments access the Web site of the company and issue OLAP queries as well as other Web traffic. The Web server of the company forwards the queries to the DW, fetches the results, creates the relevant WOQP and sends it back. In general, a WOQP has a static part possibly consisting of many files (*e.g.*, HTML document, gif images), and a dynamic part consisting of the query results. Throughout the paper we treat the static files as one composite object and assume that all WOQPs have the same static part. This is done without loss of generality, since extending the framework to account for different static parts is straightforward.

3.1 Limitations of Existing Caching Schemes

A brute force approach for caching WOQPs at a client proxy is to treat them as static HTML documents, and give them an appropriate TTL (time-to-live) value. The main drawback of this strategy is that the proxy will be able to satisfy a query only if it had been submitted in the past in its exact form. For instance, a user request for the projection at each year of the volume of products sold between 1996-1998 will not be answered, even though the proxy might have cached a WOQP referring to the volumes sold between 1995-1998. Treating WOQPs as normal Web pages will also affect the overall system performance when it comes to cache replacement decisions. The majority of replacement algorithms proposed in the literature [6], [13] assume that only network latency determines cache miss cost. This is not sufficient in our environment, since the processing time for answering an OLAP query at the server side is another significant factor. Thus, there is a need for a new cache replacement policy that takes into account both delays.

3.2 The Proposed Caching Policy

Our aim is to allow WOQP construction at the proxy using locally cached views. Active caching [7] was proposed in order to allow front-end network proxies to dynamically generate pages. A cache applet is kept together with the static part of the page and in the presence of a request the applet fetches the dynamic data from the original site and combines them with the cached static part to create the HTML document. The main benefit of this approach is that Web page construction is done close to the client and network latencies are avoided. We implement a similar scheme as follows:

The first time an OLAP query arrives at the central site, it triggers a number of different files to be sent to the client proxy:

- The WOQP answering the query;
- The static part of the WOQP;
- A cache applet;
- The view lattice diagram together with the associated query costs (Figure 3(a)) and a flag indicating whether the view is materialized at the server or not;
- The id of the view used by the server to answer the query.

The proxy forwards the WOQP to the end-user without caching it and caches the applet, the lattice diagram and the static part of the WOQP. Afterwards, it runs the cache applet which is responsible to decide whether to fetch the answering view from the server or not. Subsequent queries are intercepted and the cache applet is invoked to handle them. The applet checks whether the currently cached views can answer the query at a cost lower than sending the request to the server and selects the minimum cost cached view to do so. Then, it combines the query results with the static part of the WOQP to create the answering page. In case the views currently cached in the proxy can not answer the query, or answering the query from the proxy is more costly than doing so from the server, the request is forwarded to the Web server. The Web server responds with the WOQP carrying the results, together with the id of the view used to satisfy the query. The WOQP is forwarded to the client without being cached and subsequently the applet decides whether to download the answering view or not. The alternative of sending only the query results to the proxy and constructing the WOQP there is not considered in this paper, although the model can encapsulate this case as well. We found that unless the results are very small (not common in OLAP) the

additional overhead of going through two connections to reach the client instead of one nullifies any traffic gains. Moreover, it is reasonable to assume that WOQP construction in the proxy is more expensive than in the Web server (when the later operates under normal workload) and, therefore, it should only happen when query results are computable from the locally cached views which is more beneficial than redirecting the request to the Web server. If the storage left in the cache is not sufficient to store a newly arrived object (view or Web page), the proxy decides which objects to remove from the cache. In order to do so, it asks the cache applet for the benefit values of the cached views. The cache applet, the lattice diagram and the static part of the WOQPs are never considered in the cache replacement phase for possible eviction. They are deleted from the cache only when the traffic towards the central DW falls below a threshold specified by an administrating entity.

4 Caching Views

Deriving an analytical cost model in order to decide whether to fetch a view or not is necessary. Furthermore, a suitable cache replacement strategy must be developed that takes into account both the nature of the normal Web traffic and the additional characteristics of OLAP queries. We tackle both problems by enhancing the GDSP (Popularity-Aware Greedy-Dual-Size) [13] algorithm to take into account query processing latencies. The resulting algorithm is referred to as VEGDSP (View Enhanced GDSP). Similar enhancements are applicable to most proxy cache replacement algorithms proposed in the literature. Table 1 summarizes the notation used.

Table 1: Notation used in the paper

Symbol	Meaning
$V^{(p)}$	Set of views cached at P
$V^{(s)}$	Set of views materialized at S
$V_i^{(p)}$	The view of $V^{(p)}$ that can answer Q_i with min. cost
$V_i^{(s)}$	The view of $V^{(s)}$ that can answer Q_i with min. cost
V_i^{all}	The view that can answer Q_i with minimum cost if all views were materialized
$C(V_i^{(s)})$	Cost for answering query using $V_i^{(s)}$ view
$C(V_i^{(p)})$	Cost for answering query using $V_i^{(p)}$ view
N_i	Network latency for sending W_i to the proxy
$f(W_i)$	Frequency of W_i
$f(V_j)$	Frequency of V_j
$s(W_i)$	Size of W_i
$s(Q_i)$	Size of Q_i
$s(V_j)$	Size of V_j
$s(\overline{V_j})$	Average size of queries for V_j
$MC(W_i)$	Cache miss cost for W_i
$B(W_i)$	Benefit for W_i
$B(V_j)$	Benefit for V_j
$H(W_i)$	Cumulative benefit for W_i
$H(V_j)$	Cumulative benefit for V_j
$A_i(V^{(p)}, V^{(s)})$	Cost for answering Q_i at the system ($V^{(p)}$, $V^{(s)}$)
$A_{V_j}(V^{(p)}, V^{(s)})$	Cost for answering the average size query of V_j at the system

4.1 The VEGDSP Algorithm

Let W_i denote the ith Web page (either normal page, or WOQP), assuming a total ordering of them, $s(W_i)$ its size and $f(W_i)$ its access frequency. The basic form of VEGDSP algorithm computes a benefit value $B(W_i)$ for each page using the following formula:

$$B(W_i) = \frac{f(W_i) \cdot MC(W_i)}{s(W_i)} \quad (1)$$

where $MC(W_i)$ stands for the cost of fetching W_i from the server in case of a cache miss. In other words $B(W_i)$ represents the per byte cost saved as a result of all accesses to W_i during a certain time period. The access frequency of W_i is computed as follows:

$$f_{j+1}(W_i) = f_j(W_i) \cdot 2^{-t/T} + 1 \quad (2)$$

where j denotes the jth reference to W_i, t is the elapsed time between $j+1th$ and jth access and T is a constant controlling the rate of decay. The intuition behind Equation 2 is to reduce past access importance. In our experiments f_1 was set to $1/2$ and T to the total number of requests. VEGDSP inherits a dynamic aging mechanism from GDSP, in order to avoid cache pollution by previously popular objects. Each time a page is requested, its cumulative benefit value $H(W_i)$ is computed by summing its benefit $B(W_i)$ with the cumulative benefit L of the last object evicted from cache. Below is the basic description of VEGDSP in pseudocode:

```
L=0
IF (W_i requested)
    IF (W_i is cached)
        H(W_i) = L + B(W_i)
    ELSE WHILE (available space<s(W_i)) DO
        L = min{H(W_k)|W_k is cached}
        Evict from cache W_x: H(W_x)==L
    Store W_i
    H(W_i) = L + B(W_i)
```

In order to compute the cost $MC(W_i)$ various functions can be chosen. For instance, by selecting $MC(W_i) = 1 \ \forall W_i$ the algorithm behaves like LFU. A more suitable metric is the latency for fetching an object from the server. Most of research papers compute this latency as the summation of the time required to setup a connection and the actual transfer time. This is clearly not appropriate in case of OLAP queries since the miss penalty depends also on the query processing time at the central site, which in terms depends on which views are already materialized in the server. In the sequel we provide a cost model to compute the miss and benefit costs for caching views in the proxy.

4.2 The Cost Model

Let V be the set of views in an r-dimensional datacube ($|V| = 2^r$). A page W_i that arrives at the proxy is the answer for a unique query Q_i. In case W_i refers to normal Web traffic, $Q_i = \varnothing$. Let $V^{(p)}$ denote the set of views currently cached at the proxy and $V^{(s)}$ the ones materialized at the server. Furthermore, let $V_i^{(s)}$ be the view among the set $V^{(s)}$ that can answer Q_i with minimum cost and $V_i^{(p)}$, such a view among set $V^{(p)}$. Hence, we refer to the corresponding query costs as $C(V_i^{(s)})$ and $C(V_i^{(p)})$. Moreover, let V_i^{all} be the view that would answer Q_i with the minimum cost if all views were materialized (either at the proxy or at the server). In case Q_i can not be answered by $V^{(p)}$, $V_i^{(p)} = \varnothing$ and $C(V_i^{(p)}) = \infty$. We should notice that Q_i can always be satisfied by $V^{(s)}$ since the topmost view is always materialized at the central server. Moreover, if $Q_i = \varnothing$, then $C(V_i^{(s)}) = C(V_i^{(p)}) = 0$. Let $Cn(P \rightarrow S)$ be the cost (in terms of latency) for establishing a connection between the proxy and the server, and $Tr(S \rightarrow P)$ be the

average transfer rate at which the server sends data to the proxy. The network latency N_i, exhibited when fetching W_i from the central server is given by:

$$N_i = Cn(P \rightarrow S) + \frac{s(W_i)}{Tr(S \rightarrow P)}$$

with $s(W_i) = s(w) + s(Q_i)$, where $s(w)$ denotes the size of the static part of the page and $s(Q_i)$ the size of the query results.

Finally, we denote the time required to construct W_i having obtained the results of Q_i by $F_i^{(s)}$ and $F_i^{(p)}$, depending whether the construction occurs at the server site or at the proxy. In case $Q_i = \varnothing$, $F_i^{(s)} = F_i^{(p)} = 0$. The total cost $MC(W_i)$ of a cache miss for W_i in terms of latency is given by:

$$MC(W_i) = C(V_i^{(s)}) + F_i^{(s)} + N_i \quad (3)$$

We should notice that in case W_i comes from normal Web traffic Equation 3 is reduced to:

$$MC(W_i) = N_i \qquad (Q_i = \varnothing) \quad (4)$$

Equations 3, 4 define the miss cost for a WOQP and a normal Web page, respectively. The benefit and cumulative benefit values can then be derived using Equation 1. Under our scheme we do not consider caching WOQPs due to the ad-hoc nature of OLAP queries.

Concerning views we can compute directly the benefit $B(V_j)$ of keeping in cache V_j view, by taking the difference in total cost for answering the queries before and after a possible eviction of V_j from the cache. Let $f(V_j)$ denote the access frequency of V_j. Since there are no direct hits for views we use the following alternative to compute $f(V_j)$. Whenever a query Q_i arrives, the cache applet adapts the frequency of V_i^{all} using Equation 2.†

Let $A_i(V^{(p)}, V^{(s)})$ denote the cost for satisfying Q_i in the whole system (both proxy and server). Q_i can be answered either by $V^{(p)}$ or by $V^{(s)}$, depending on the relative cost difference. Thus, we reach the following equation:

$$A_i(V^{(p)}, V^{(s)}) = min\left\{\begin{matrix} C(V_i^{(p)}) + F_i^{(p)}, \\ C(V_i^{(s)}) + F_i^{(s)} + N_i \end{matrix}\right\} \quad (5)$$

Let $s(V_j)$ be the size of view V_j and $s(\overline{V_j})$ be the average query size for queries with $V_i^{all} = V_j$. Since all queries satisfied by the same view incur the same processing cost (proportional to the view size), the benefit value of V_j can be computed as follows:

$$B(V_j) = \frac{\sum_{\forall V_k} f(V_k)[A_{V_k}(V^{(p)} - \{V_j\}, V^{(s)}) - A_{V_k}(V^{(p)}, V^{(s)})]}{s(V_j)} \quad (6)$$

where $A_{V_k}(V^{(p)}, V^{(s)})$ stands for the cost of answering at the system a query Q_i: $V_i^{all} = V_k$ && $s(Q_i) = s(\overline{V_k})$.

4.3 Deriving the Parameters

Here we provide details on how to compute the parameters of Equations 5, 6. $C(V_i^{(s)})$ and $C(V_i^{(p)})$ are computed by finding at the lattice diagram the query costs of the corresponding $V_i^{(s)}$ and $V_i^{(p)}$ views as described in Section 2. Computing $C(V_i^{(s)})$ is feasible since in each node of the cached lattice there is a tag denoting whether the view is materialized at the central site or not (*m_at_cs* tag). A second tag (*c_at_p*) shows if it is cached at the proxy. The cache applet is responsible for searching the lattice and defining the query cost values‡. It is also responsible for updating the *c_at_p* tag whenever a new view is stored or deleted. Unless the central site follows a static view selection policy, we need to employ a consistency mechanism in order to keep the *m_at_cs* tag up to date. Periodically, the proxy sends a GET IF MODIFIED SINCE request to the central site and gets an updated version of the lattice if needed. Naturally, this means that the cache applet might use a stale lattice copy at its query processing decisions, but the performance impact of that is expected to be marginal.

Calculating $s(\overline{V_k})$ values is done by having the proxy keeping track of the query result sizes exhibited locally and having the central server informing the applet of the result sizes for queries satisfied by him. An assumption made in our experiments was that the proxy and the central server have equal processing capabilities. This, in terms, implies that $F_i^{(s)} = F_i^{(p)}$ and the query cost for a view (Figure 3(a)) is the same, regardless whether it is cached at the proxy or at the central site. Estimation of the network latency parameters, can be done by keeping statistics of past downloads in a per server basis and predict the latency exhibited in the future, in a way similar to how RTT (Round-Trip-Time) is estimated in the TCP [25].

4.4 Cache Admittance of Views in VEGDSP

Web caching algorithms consider for caching all arriving objects. This stems from the fact that Web traffic exhibits temporal locality [4]. However, when views come in question such approach is inadequate since their size can be large, resulting in many objects being evicted from the cache in order to free space. To avoid this we decided to follow an alternative policy.

When a view V_j is considered for caching at the proxy, its benefit value $B(V_j)$ is calculated using Equation 6 and consequently its cumulative benefit value $H(V_j)$ is defined as in Section 4.1. In case that there is not enough storage space left to cache V_j, instead of evicting immediately the object with the least cumulative benefit which might still not free enough space, we calculate the minimum possible aggregated cumulative benefit of a set of objects that if deleted from the cache, enough space would be freed. V_j is cached only if $H(V_j)$ is greater than this aggregated value. Figure 4 provides a description in pseudocode of the complete VEGDSP caching algorithm.

5 Experimental Evaluation

In this section we present the simulation results. There are two scenarios considered for comparison. First, a proxy that caches only normal Web pages using the GDSP algorithm. Second, a proxy that implements VEGDSP. We measured the performance of the two alternatives in terms of Cost Saving (CS). CS is defined as:

$$\frac{WCost - PCost}{WCost}$$

where *WCost* is the cost occurred when no proxy is available and *PCost* the cost of each of the proxy implementations.

5.1 The Workload

In order to simulate our environment we generated representative workloads for both OLAP queries and Web requests. For the OLAP queries we employed datasets from the TPC-H benchmark [27] and the APB benchmark [20]. We used a subset of the TPC-H database schema consisting of 14 attributes, while for the APB dataset we used the full

†. Frequency counters are maintained in the lattice diagram.

‡. Instead of searching the lattice upon a query arrival, each node in the lattice keeps an *answering_view* field storing the minimum cost view that can answer queries referring to the node. This information can be maintained efficiently when a new view is added or deleted from the cache.

```
L=0
IF ( W_i requested && Q_i = ∅ )        /*Normal Web page*/
   IF ( W_i is cached)
        H(W_i) = L + B(W_i)
   ELSE WHILE (available space< s(W_i) ) DO
        L = min{H(W_k), H(V_m)|W_k, V_m are cached}
        Evict object K = ((W_x : H(W_x) == L) || (V_n : H(V_n) == L))
   Store W_i
   H(W_i) = L + B(W_i)
ELSE IF ( W_i requested && Q_i ≠ ∅ ) /*OLAP query*/
   IF (( V_i^(p) ≠ ∅ ) && (A_i(V^(p), V^(s)) == C(V_i^(p)) + F_i^(p) ))
        Find query results
        Construct and send back W_i
        H(V_i^(p)) = L + B(V_i^(p))
   ELSE                                /* Q_i can not be answered by the proxy
        Send Q_i to the central site     or it is more expensive*/
ON ARRIVAL of view_id V_i^(s) from the server
   temp = available space
   cum_benefit = 0
   evict_list = ∅
   WHILE (temp< s(V_i^(s)) ) DO  /*Calculate the aggregation of cumulative
                                   benefits for the least costly set of objects
                                   need to be evicted to free-up space*/
        L = min{H(W_k), H(V_m)|W_k, V_m are cached}
        ADD IN evict_list object K = ((W_x : H(W_x) == L) ||
                                      (V_n : H(V_n) == L))
        temp += s(K)
        cum_benefit += H(K)
    IF (cum_benefit < H(V_i^(s)) )     /*Fetch the answering view from
        Evict objects in the evict_list    the server if beneficial*/
        Fetch and store V_i^(s)        /*Store V_i^(s) with H(V_i^(s))
        H(V_i^(s)) = L + B(V_i^(s))        cumulative benefit*/
```

Figure 4: Pseudocode for VEGDSP

schema for the dimensions. The size of raw data for TPC-H is 6M tuples and for APB 1.3M tuples. For the Web traffic we used a synthetic workload with Zipf distribution for the popularity of pages and a heavy tail distribution for the page size. The average page size was 50K and that was the value used also for the static part of WOQP. The total number of Web pages was 30,000 and the total number of requests 100,000. OLAP queries were generated using a uniform distribution, *i.e.*, the probability of a query to refer to a node in the lattice was equal for all nodes and were afterwards combined with the Web traffic randomly again, to form the request pattern arriving at the proxy. Query size was also selected to follow a uniform distribution to the size of the view.

Since the views materialized at the DW server affect the query costs and the caching decisions, we decided to employ the VEGDSP at the server side, too. Furthermore we allowed the server to cache only 10% of the datacube (total size of views) which corresponds for the TPC-H dataset to 100M tuples and for the APB to 5.8M tuples. We should note here that the only factor that burdens the materialization of all the datacube is the storage capacity, *i.e.*, we do not take into account update constraints. We conducted our experiments in an UltraSparc2 workstation running at 200MHz with 256 MB of main memory. The trend of the results for both the APB and the TPC-H dataset was similar. In the rest of this section we will only present the results for the APB dataset, due to lack of space.

5.2 Results

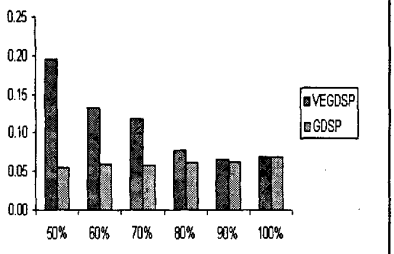
Intuitively, caching OLAP data into the proxy server pays off when there is a substantial amount of OLAP requests. In our first set of experiments, the goal is to identify the ratio of OLAP to common Web requests above which, VEGDSP is beneficial. In Figure 5(a) we compare VEGDSP and GDSP using the APB dataset. The cache size is fixed to 10% of the total size of Web pages, the network transfer rate is 32KBps and the percentage of non-OLAP (*i.e.*, Web requests) varies from 50% to 100%. The first thing to notice is that VEGDSP outperforms GDSP with the differences being more apparent when the percentage of OLAP requests is high. When the workload consists of Web requests only, VEGDSP acts exactly as GDSP. Observe that the performance of GDSP deteriorates when the OLAP requests increase. This is due to the fact that GDSP considers only Web requests, therefore the percentage of requests that are benefited drops.

Figure 5(b) shows the results when the cache size is 50% of the total size of Web pages. While the trend is the same, the difference between the two algorithms is smaller. By setting the cache to 50% we provide enough space for GDSP to achieve almost its maximum performance since most of the frequently accessed pages fit in the cache (recall that the Web requests follow a Zipf distribution). On the other hand, VEGDSP is benefited in a smaller extend by the increase to the cache size, since the OLAP requests follow a uniform distribution. Note that VEGDSP was not always better than GDSP. We recorded some cases, when the OLAP requests were around 5% of the workload, where VEGDSP was marginally worse. This was more obvious for the TPC-H dataset, since it is more skewed. The reason is that the cost overhead for transferring views from the data warehouse to the proxy is not amortized by answering a significant number of queries locally. Nevertheless, in the general case our experiments show that there is a threshold on the percentage of OLAP traffic, above which caching OLAP data provides substantial benefits. In the tested cases this threshold was around 10%, which is very promising, since in a decision-making environment this value can be easily exceeded.

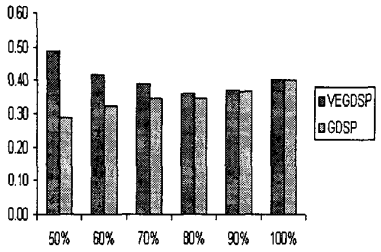
In the second set of experiments we tested the performance of VEGDSP when cache size varies between 1% and 50% of the total Web page size. The network transfer rate was again fixed to 32KBps and the percentage of OLAP requests was set to 50% (Figure 6(a)) and 30% (Figure 6(b)). The performance for both algorithms increases to the available cache size. We observe that VEGDSP follows the same trend as GDSP while clearly maintaining a lead even for the very modest cases of 1% and 5% cache sizes. Another observation is that the proportional performance difference of the algorithms shrinks as the cache size increases (noted also in Figure 5).

In the final experiment, we investigated the performance of VEGDSP as a function of the transfer rates between the proxy and the central DW. VEGDSP was executed for 50%, 70% and 90% of non-OLAP traffic and the network transfer rate varied from 32 KBps to 4 MBps. Figure 7 presents the results for cache sizes equal to 10% and 50%. We observe that CS decreases as the network transfer rate increases. Recall from Section 4 that the decision of whether to satisfy a query using the cached views at the proxy or redirecting it to the DW, depends on both the processing cost and the network cost. Since the DW materializes a substantial part of the datacube there is a high probability that the processing cost for answering a query at the DW is lower than the one at the proxy. With a higher transfer rate more queries will be redirected to the DW resulting to lowering the gains of the algorithm. Since this behavior is due to OLAP traffic, the performance drop is more prominent at the VEGDSP_50 case.

The above results indicate that in the presence of OLAP queries traditional Web caching schemes can be inefficient. The proposed architecture together with the cache

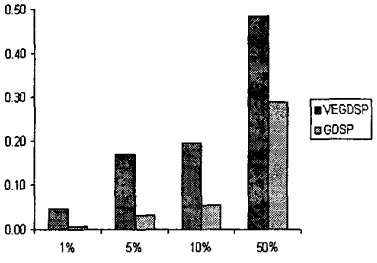


(a): 10% cache

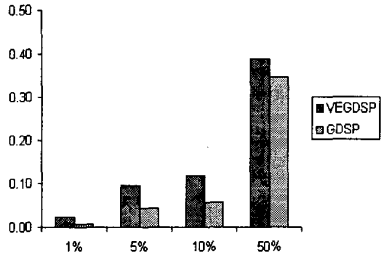


(b): 50% cache

Figure 5: CS vs. Percentage of Web Requests for different cache sizes.

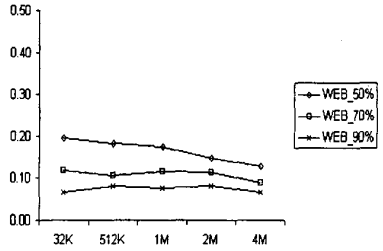


(a): (50%/50%) (OLAP/Web)

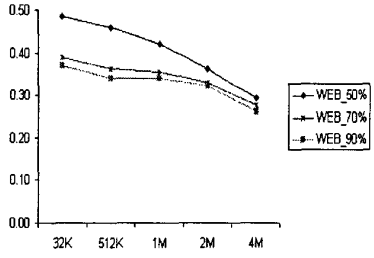


(b): (30%/70%) (OLAP/Web)

Figure 6: CS vs. Cache Size for different (OLAP/Web) ratios.



(a): 10% cache



(b): 50% cache

Figure 7: CS vs. Network Transfer Rate for different cache sizes.

algorithm (VEGDSP) can result in improving the overall system performance.

6 Related Work and Conclusions

In this paper we considered the problem of minimizing the cost of online analytical processing queries issued through the Internet. We proposed a novel scheme that allows a proxy to reply to OLAP queries without necessarily having to access the central DW. An analytical cost model is derived to quantify the actual benefits. Furthermore, a suitable cache algorithm (VEGDSP) is developed that judiciously treats OLAP views and Web pages, taking into account different costs involved in each case. Results of the simulation studies confirm the efficiency of our framework, even when the ratio of OLAP queries to normal Web traffic is moderate.

Related to this paper is previous work on the view selection problem [12] where the authors proposed a greedy algorithm that chooses a near-optimal set of views, given the storage capacity constraint and an expected query workload. View selection under update constraints was studied in [9]. The approximation algorithm achieves in the worst case solution quality within 63% of the optimal. In [3] the search space of the problem is reduced by a heuristic that excludes views irrelevant to the most frequent queries. [24] describes another method for view selection which is based on sorting and has smaller computational overhead than [12], while ensuring the same lower worst case bound provided that the view sizes satisfy certain conditions. In [26] the authors study the minimization of both query execution and view maintenance costs, under the constraint that all queries should be answered from the selected views. The above methods aim at solving a resulting optimization problem in a static and centralized manner. Even though they can be considered for implementing view selection in a central site if the query patterns do not change frequently, they are not suitable for materializing views in a dynamic environment.

In [15] a method is proposed in order to dynamically materialize and maintain fragments of OLAP views with respect to both space and time constraints in a DW, while in [16] the authors consider a Web server linked to a DBMS and tackle the problem of whether to cache views at the server, at the DBMS, or compute them on fly. [15] and [16] are simple caching algorithms that consider views as the only objects to be cached. Thus, they can suffer from what known as the cache pollution problem, *i.e.*, previously popular documents fill in the cache if applied directly to a Web environment. A normalized cost caching and admission algorithm for DW is presented in [22]. The same authors proposed similar caching algorithms for Web proxies in [23], but do not consider OLAP queries. The problem of caching OLAP in Web environments is studied in [14]. However, that approach is based on a dedicated infrastructure of DBMSs which is different from the Web proxies. In [19], active caching is employed to store database results in proxies, but only transactional (*i.e.*, non-OLAP) workloads are considered.

Various Web proxy caching algorithms exist in the WWW literature [2], [6], [13], [17], [23], [29]. Our approach is applicable in conjunction with these algorithms. Here, we use the GDSP algorithm because as established in [13] and in [2], [6], [30] (for previous versions of the algorithm), it leads to efficient solutions when Web traffic is concerned. Our aim is not to propose a new proxy caching algorithm but rather to provide a framework for caching OLAP views as well as illustrating and solving the problems that rise. We are currently applying our extensions to all the main algorithms of the WWW literature to compare their performance in our environment.

Another direction for the future work involves the development of efficient update processes. In this paper we

used an approach that invalidates all cached copies instead of updating the cached views at the proxies. Strategies that refresh parts of the cached views and invalidate others will most likely lead to better performance. Another possibility is to take advantage of the ICP (Internet Cache Protocol) [28] and the proxy hierarchies as described in [8] to further reduce the query costs. The intuition is that a proxy can fetch a view or satisfy a query by forwarding the request to a proxy located close to it, instead of sending it to the central site. Research in both directions can be based on the proposed framework, cost model and the caching algorithm.

Acknowledgments

This work was supported from the Research Grants Council of the Hong Kong SAR, grants HKUST 6090/99E and HKUST 6070/00E, as well as, HKTIIT98/99.EG02.

References

[1] M. Abrams, C. Standridge, G. Abdulla, S. Williams and E. Fox, "Caching Proxies: Limitations and Potentials," *in Proc. of the 4th Int. World Wide Web Conf'95: The Web Revolution*, Boston MA, Dec. 11-14, 1995.

[2] M. Arlitt, L. Cherkasova, J. Dilley, R. Friedrich and T. Jin, "Evaluating content management techniques for Web proxy caches," *in Proc. of the ACM SIGMETRICS Performance Evaluation Review*, Vol. 27(4), pp. 3-11, March, 2000.

[3] E. Baralis, S. Paraboschi, E. Teniente, "Materialized view selection in a multidimensional database," *in Proc. of the 23rd Int. Conf. on Very Large DataBases (VLDB'97)*, pp. 156-165, 1997.

[4] P. Barford, A. Bestavros, A. Bradley and M. Crovella, "Changes in Web client access patterns: characteristics and caching implications," *WWW Journal Special Issue on Characterization and Performance Evaluation*, Vol. 2, pp. 15-28, 1999.

[5] A. Bestavros, "WWW Traffic Reduction and Load Balancing through Server-based Caching," *IEEE Concurrency: Special Issue on Parallel and Distributed Technology*, Vol.5, pp. 56-67, Jan.-Mar. 1997.

[6] P. Cao and S. Irani, "Cost-aware WWW proxy caching algorithms," *in Proc. of the USENIX Symposium on Internet Technology and Systems*, pp. 193-206, Dec. 1997.

[7] P. Cao, J. Zhang and K. Beach, "Active Cache: Caching Dynamic Contents on the Web," *in Proc. of Middleware'98 Conference*, 1998.

[8] A. Chankhunthod, P.B. Danzig, C. Neerdals, M.F. Schwartz and K.J. Worrell, "A Hierarchical Internet Object Cache," *in Proc. of the USENIX Technical Conference*, San Diego, CA, Jan. 1996.

[9] H. Gupta, I.S. Mumick, "Selection of Views to Materialize Under a Maintenance-Time Constraint," *Int. Conf. on Database Theory (ICDT'99)*, pp. 453-470, 1999.

[10] J.S. Gwertzman and M. Seltzer, "The Case for Geographical Push-Caching," *in Proc. of the 5th Workshop on Hot Topics in Operating Systems (HotOS-V)*, IEEE Computer Society Press, Los Alamitos, Calif., 1995, pp.51-55.

[11] J. Hammer, H. Garcia-Molina, J. Widom, W. Labio and Y. Zhuge, "The Stanford Data Warehousing Project," *IEEE Data Eng. Bulletin, Special Issue on Materialized Views and Data Warehousing*, Vol. 18(2), pp. 41-48, 1995.

[12] V. Harinarayan, A. Rajaraman, and J.D. Ullman, "Implementing Data Cubes Efficiently," *in Proc. of the ACM SIGMOD Int. Conf. on Management of Data*, pp. 205-216, 1996.

[13] S. Jin and A. Bestavros, "Popularity-Aware GreedyDual-Size Web Proxy Caching Algorithms," *in Proc. of the 20th IEEE Int. Conf. on Distributed Computing Systems (ICDCS'00)*, April 2000, Taiwan, pp. 254-261.

[14] P. Kalnis, D. Papadias, "Proxy-Sever architectures for OLAP," *in Proc. of the ACM SIGMOD Int. Conf. on Management of Data*, pp. 367-378, 2001.

[15] Y. Kotidis, N. Roussopoulos, "DynaMat: A Dynamic View Management System for Data Warehouses," *in Proc. of the ACM SIGMOD Int. Conf. on Management of Data*, pp. 371-382, 1999.

[16] A. Labrinidis, N. Roussopoulos, "WebView Materialization," *in Proc. of the ACM SIGMOD Int. Conf. on Management of Data*, pp. 367-378, 2000.

[17] P. Lorenzetti, L. Rizzo and L. Vicisano, "Replacement policies for a proxy cache," Techn. Report LR-960731, Univ. di Pisa, available at: http://www.iet.unipi.it/~luigi/research.html.

[18] T. Loukopoulos and I. Ahmad, "Replicating the Contents of a WWW Multimedia Repository to Minimize Download Time," *in Proc. of the 14th Int. Parallel and Distributed Processing Symposium, (IPDPS'00)*, Cancun, Mexico, May, 2000.

[19] Q. Luo., J.F. Naughton, R. Krishnamurthy, P. Cao, Y. Li, "Active Query Caching for Database Web Servers," *in Proc. of Int. Workshop on Web and Databases (WebDB)*, pp.29-34, 2000.

[20] OLAP Council, "OLAP Council APB-1 OLAP Benchmark, Release II," http://www.olapcouncil.org.

[21] M. Rabinovich, "Issues in Web Content Replication," *in Data Engineering Bulletin*, Invited Paper, Vol.21 No.4, Dec. 1998.

[22] P. Scheuermann, J. Shim and R. Vingralek, "WATCHMAN: A Data Warehouse Intelligent Cache Manager," *in Proc. of the 22nd Int. Conf. on Very Large DataBases (VLDB'96)*, pp. 51-62, 1996.

[23] J. Shim, P. Scheuermann and R. Vingralek, "Proxy cache algorithms: Design, implementation and performance," *IEEE Trans. on Knowledge and Data Engineering*, Vol. 11(4), pp. 549-562, July/Aug., 1999.

[24] A. Shukla, P.M. Deshpande, J.F. Naughton, "Materialized View Selection for Multidimensional Datasets," *in Proc. of the 24th Int. Conf. on Very Large DataBases (VLDB'98)*, pp. 488-499, 1998.

[25] W.R. Stevens, "TCP/IP Illustrated," vol. 3, *Addison-Wesley*, 1996, Section 10.5.

[26] D. Theodoratos, T.K. Sellis, "Data Warehouse Configuration," *in Proc. of the 23rd Int. Conf. on Very Large DataBases (VLDB'97)*, pp. 126-135, 1997

[27] Transaction Processing Performance Council, "TPC Benchmark R (Decision Support), Rev. 1.0.1," http://www.tpc.org/, 1993 - 1998.

[28] D. Wessels and K. Claffy, "Internet Cache Protocol (ICP) version 2," RFC2186, 1998.

[29] R. Wooster and M. Abrams, "Proxy caching that estimates page load delays," *in Proc. of the 6th Int. World Wide Web Conf.*, Santa Clara, CA, April, 1997.

[30] N.E. Young, "On-line caching as cache size varies," *in Proc. of Symposium on Discrete Algorithms (SODA'91)*, pp. 241-250, Jan., 1997.