Policies for Caching OLAP Queries in Internet Proxies

Thanasis Loukopoulos and Ishfaq Ahmad, *Senior Member*, *IEEE*

Abstract—The Internet now offers more than just simple information 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 often referred to as *On-Line-Analytical Processing* (OLAP). Typically, pages carrying query results do not exhibit temporal locality and, therefore, are not considered for caching at Internet proxies. In OLAP processing, this is a major problem as the cost of these queries is significantly larger than that of the transactional queries. This paper proposes a technique to reduce the response time for OLAP queries originating from geographically distributed private LANs and issued through the Web toward a central data warehouse (DW) of an enterprise. An *active caching* scheme is introduced 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 respective latencies, taking into consideration the combined effects of both common Web access and query processing. We propose a cache admittance and replacement algorithm that operates on a hybrid Web-OLAP input, outperforming both pure-Web and pure-OLAP caching schemes.

Index Terms—Distributed systems, data communication aspects, Internet applications databases, Web caching, OLAP.

1 Introduction

Caching has emerged as a primary technique for coping with high latencies experienced by the Internet users. There are four major locations where caching is performed:

1. proxy at the front-end of a server farm [7],
2. network cache at the end-points of the backbone network [13],
3. LAN proxy [1], [39], and
4. browser.

Although caching at these locations has been shown to significantly reduce Web traffic [3], dynamically generated pages are not cacheable. Dynamic pages typically consist of a static part and a dynamic one (for example, query results).

On the other hand, the need for decision support systems has become of paramount importance in today's business, leading many enterprises to building decision support databases called data warehouses (DWs) [14]. 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 [15]. Fetching from the DW 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 and hence 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.

In this paper, we address the problem of caching OLAP queries posed by ad hoc, geographically spanned users, through their Web browsers. Unlike previous approaches, e.g., [18], [20], we employ the existing proxy infrastructure and propose a method of caching both Web pages and OLAP query results in common proxy servers. Our work is applicable to other caching points, provided that significant traffic towards the DW passes through them (e.g., edge servers of a Content Distribution Network [22]). Some preliminary results were presented in [25].

Web pages carrying OLAP query results, 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 (as is the case for instance with sport pages where continuous updates occur in the server), but is rather due to the fact that it is unlikely that successive queries bear the same results. 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. The proposed active caching framework 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 empirically proven to lead to high quality solutions. Although active caching has been employed before in answering transactional queries [26], to the best of our

- *T. Loukopoulos is with the Department of Computer and Communication Engineering, University of Thessaly, 37 Glavani—28th October str., Deligiorgi Bld., 38221 Volos, Greece. E-mail: luke@ inf.uth.gr.*
- *I. Ahmad is with the University of Texas at Arlington, Box 19015, CSE, UTA, Arlington, TX 76019. E-mail: iahmad@cse.uta.edu.*

Manuscript received 13 Sept. 2004; revised 4 July 2005; accepted 8 Sept. 2005; published online 24 Aug. 2006.
Recommended for acceptance by J. Fortes.
For information on obtaining reprints of this article, please send e-mail to: tpds@computer.org, and reference IEEECS Log Number TPDS-0231-0904.

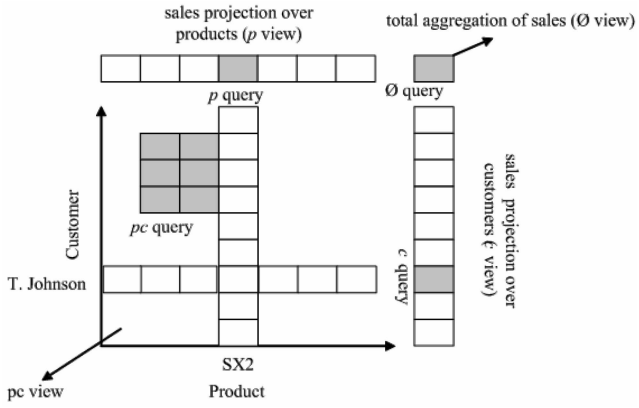


Fig. 1. An example of OLAP queries in 2D space.

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 optimizations (e.g., the interdependencies of the views in a lattice).

The rest of the paper is organized as follows: Section 2 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 discusses the simulation results, while Section 6 presents the related work. Finally, Section 7 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 data sets, 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. Fig. 1 depicts an example of a 2D space with the dimensions being the customer's name and the product id.

The value at each cell in the 2D 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 Fig. 1. It could also be a *group_by* query for two products and three 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 Fig. 1, we might consider for materialization the results of the four 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

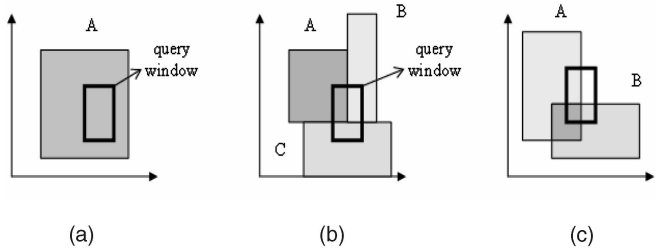


Fig. 2. Using materialized views to answer queries. (a) Query answered by one view, (b) query answered by combining three views, and (c) query cannot be answered by any view combination.

view, or through a combination of two or more views as shown in Fig. 2.

In our 2D 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 Figs. 2b and 2c. 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 [12], [15] and not fragments of them. In this paper, we follow this approach. For instance, in the example of Fig. 1, the only candidates for materialization are the p, c, $\oslash$ 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 is 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 Fig. 3. The three 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 (Fig. 3a), e.g., c can be derived from pct 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 [15]. Fig. 3a 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 Fig. 3b, 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 $\oslash$ view is to calculate it from p as compared to pc and pct.

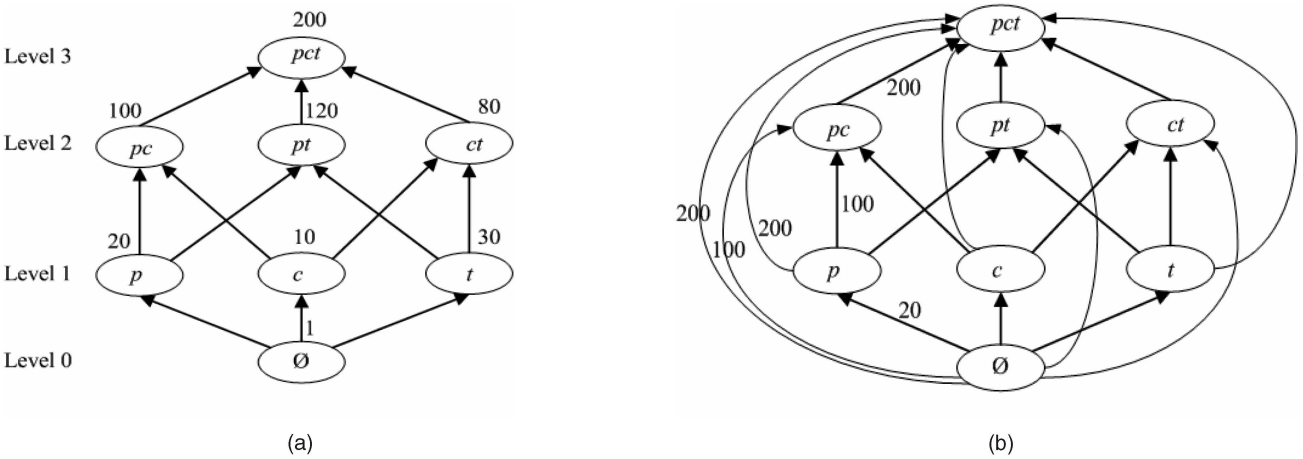


Fig. 3. Lattice and expanded lattice diagrams for $< p, c, t >$ dimensions with associated query and view computing costs. (a) Query costs associated with each node and (b) costs for computing views associated with each edge.

Answering an OLAP query of the form:

```
SELECT <grouping predicates> AGG (predicate)
FROM <data list>
WHERE <selection predicates>
GROUP BY <grouping predicates>
```

involves the following steps: 1) the query dimensions are defined as the union of the selection and grouping predicates, 2) the corresponding to the dimensions view is located, and 3) in case the view is not cached, we check whether any of its ancestors are present and select the one with the minimum cost to answer the query.

Since recomputing views from the raw data is an expensive procedure, it is common practice that the central DW always keeps the topmost view materialized, in order to be able to handle all OLAP queries [20]. We follow the same policy in the central DW but not in the proxy, 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, putting 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 2000 and 2002 will not be answered, although the proxy might have cached a WOQP referring to the volumes sold between 1999 and 2002. 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 [9], [17] 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. Therefore, we need to develop a new cache replacement policy that can take 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 [10] 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 (Fig. 3a) 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. Afterward, it runs the cache applet, which is responsible for deciding 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 cannot 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) [17] algorithm to take into account query processing latencies. The resulting algorithm is referred to as VHOW (Virtual Hybrid OLAP Web). Similar enhancements are applicable to most proxy cache replacement algorithms proposed in the literature. Table 1 summarizes the notation used.

4.1 The VHOW Algorithm

Let W_i denote the $i\text{th}$ 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 VHOW algorithm computes a benefit value $B(W_i)$ for each page using the following formula:

$$B(W_i) = f(W_i)M(W_i)/s(W_i), \tag{1}$$

where $M(W_i)$ 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:

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

where j denotes the $j\text{th}$ reference to W_i, t is the elapsed number of requests between the $j+1\text{th}$ and $j\text{th}$ access, and T is a constant controlling the rate of decay. The intuition behind (2) is to reduce past access importance. In our experiments f_1 was set to $1/2$ and T to $1/5\text{th}$ of the total number of requests. VHOW 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. Thus, objects that were frequently accessed in the past, but account for no recent hits are forced out of the cache, whereas, if eviction was only based on the benefit values (and not on the cumulative benefit) they would have stayed for a larger time period. Below is the basic description of VHOW 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 = $\min\{H(W_k) : W_k \text{ are cached}\}$
      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 $M(W_i)$ various functions can be chosen. For instance, by selecting $M(W_i) = 1 \forall W_i$, 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 Cost Model

Let V be the set of views in an $r\text{-dimensional}$ 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, $Q_i = \oslash$. Let $V^{(P)}$ denote the set of views currently

TABLE 1
Notation Used in the Paper

Symbol	Meaning
Q_i	The ith query
W_i	The ith page
$V^{(P)}$	Set of views cached at proxy P
$V^{(S)}$	Set of views materialized at the central server S
$V_i^{(P)}$	The view of $V^{(P)}$ that can answer Q_i with minimum cost
$V_i^{(S)}$	The view of $V^{(S)}$ that can answer Q_i with minimum 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 Q_i using $V_i^{(S)}$ view
$C(V_i^{(P)})$	Cost for answering Q_i using $V_i^{(P)}$ view
N_i	Network latency for sending W_i to the proxy
$F_i^{(S)}$	Cost to construct W_i at the central server
$F_i^{(P)}$	Cost to construct W_i at the proxy
$f(W_i)$	Access 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
$\bar{s}(V_j)$	Average size of queries for V_j
$M(W_i)$	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 $(V^{(P)}, V^{(S)})$

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 $V_i^{(P)}$, a similar 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 $V^{(P)}$, $V_i^{(P)} = \oslash$ 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 $Q_i = \oslash$, $C(V_i^{(S)}) = C(V_i^{(P)}) = 0$. Let $L(P \rightarrow S)$ be the cost (in terms of latency) for establishing a connection between the proxy and the server, and $T(S \rightarrow P)$ 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: $N_i = L(P \rightarrow S) + s(W_i)/T(S \rightarrow P)$, where $s(W_i) = s(w) + s(Q_i)$, with $s(w)$ denoting 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 query results) at the central server and the proxy by $F_i^{(S)}$ and $F_i^{(P)}$, respectively. In case $Q_i = \oslash$, $F_i^{(S)} = F_i^{(P)} = 0$. The total cost $M(W_i)$ of a cache miss for W_i in terms of latency is given by:

$$M(W_i) = C(V_i^{(S)}) + F_i^{(S)} + N_i. \quad (3)$$

Notice that, in case W_i comes from normal Web traffic (3) is reduced to:

$$M(W_i) = N_i \quad (Q_i = \oslash). \quad (4)$$

Equations (3) and (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 (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 V_j view in the cache, 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 (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 end up with the following equation:

$$A_i(V^{(P)}, V^{(S)}) = \min\left\{ \begin{array}{c} C(V_i^{(P)}) + F_i^{(P)}, \\ C(V_i^{(S)}) + F_i^{(S)} + N_i \end{array} \right\}. \quad (5)$$

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 (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 $C(V_i^{(S)})$ requires each node of the cached lattice diagram to maintain two fields. The first one (*materialized* field) denotes whether the view is materialized at the central site or not, while the second (*cached* field) shows if it is cached at the proxy. Unless the central site follows a static view selection policy, we need a consistency mechanism in order to keep the *materialized* field up to date.

The cache applet is responsible for defining which view can answer a query with the minimum cost. In order to avoid traversing the lattice upon every query arrival, each node stores two additional fields. The first (*local_answering_view*), shows the cached view that can answer the queries related to the node at a minimum cost, while the second (*remote_answering_view*) keeps the id of the minimum cost answering view at the DW. This information can be maintained efficiently when a new view is added or deleted from the cache.

Benefit calculation requires further discussion. Whenever a new query Q_i corresponding to the V_i view arrives, the benefit values of V_i and all its ancestors in the lattice, must be updated (notice that the benefit of successor views in the lattice do not alter since V_i is not computable by them). Straightforward calculation of (6) requires $O(|V|^2)$ time in the worst-case ($|V|$ is the number of views in the lattice). However, we can incrementally compute the benefits in $O(|V|)$ worst-case time, by noticing that only the coefficient of V_i changes in (6). Finally, estimation of the network latency parameters can be done by keeping statistics of past downloads and predicting future latency, in a way similar to how RTT (Round-Trip-Time) is estimated by the TCP protocol [35].

4.4 Cache Admittance of Views in VHOW

Web caching algorithms consider for caching all arriving objects, stemming from the fact that Web traffic exhibits temporal locality [6]. 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 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 (6) and, consequently, its cumulative benefit value $H(V_j)$ is defined as in Section 4.1. In case 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 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. Fig. 4 shows a description in pseudocode of the complete VHOW caching algorithm.

Deleting objects from the cache in order to fit a new view deserves further attention. The problem can be formulated as: Given a set of n objects, each of benefit b_i and size s_i, find a subset D such as: $\sum_{i\in D} b_i \leq B$ and $\sum_{i\in D} s_i \geq S$, with B, S, b_i, s_i integers. Notice that by interchanging the roles of benefit and size we end with the (0, 1) Knapsack problem [29], the decision problem of which is known to be NP-complete. (0, 1) Knapsack can be solved to optimality using dynamic programming [29] however, the method incurs unacceptable (for caching purposes) running time. Therefore, we followed a heuristic approach. We start by adding in a candidate list D (evict_list in the pseudocode) the objects O_j of minimum cumulative benefit $H(O_j)$, until: 1) $\sum_{O_j\in D} H(O_j) \leq H(V_i^{(S)})$ and $\sum_{O_j\in D} s(O_j) \geq s(V_i^{(S)})$ or, 2) $\sum_{O_j\in D} H(O_j) \geq H(V_i^{(S)})$. In the first case, the view is admitted after deleting the objects in the candidate list, while in the second case we check whether the last added object in D, let O_k, satisfies $H(O_k) \leq H(V_i^{(S)})$ and $s(O_k) \geq s(V_i^{(S)})$, in which case O_k is replaced with $V_i^{(S)}$, otherwise $V_i^{(S)}$ is not admitted.

5 Experimental Evaluation

Two series of experiments were conducted. The first aimed at investigating the throughput of a hybrid Web-OLAP proxy, while the second used simulation in order to determine the potential benefits in query cost terms.

5.1 System Throughput

Before proceeding with identifying the potential gains in query cost terms, we investigated whether augmenting a proxy with query answering capabilities has an adverse effect on the throughput of the rest of HTTP requests. For

```
L=0
IF ( W_i requested && Q_i =Ø)   /*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_x), H(V_m) : W_k, V_m are cached}
         evict obj= (W_x : H(W_x) == L) || (V_m : H(V_m) == L)
      store W_i
      H(W_i) = L + B(W_i)
ELSE IF ( W_i requested && Q_i ≠Ø)    /*OLAP query*/
  /*Check if Q_i can be answered from the proxy at a lower cost*/
   IF (( V_i^(P) ≠Ø) && (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)) = B(V_i^(P))   /*Initialize cumulative benefit*/
   ELSE /*query can not be answered by the proxy or it is more
costly to do so*/
      Send Q_i to the central site

ON RECEIVING the id of an answering view V_i^(S) from the server
temp=available space
cum_benefit=0
evict_list=Ø
WHILE (temp< s(V_i^(S)) && cum_benefit≤ H(V_i^(S)) && objects
remain) DO
   L=min{ H(W_x), H(V_m) : W_x, V_m are cached && ∉ evict_list}
   addin evict_list obj= (W_x : H(W_x) == L) || (V_m : H(V_m) == L)
   temp+= s(obj)
   cum_benefit+= H(obj)
IF (cum_benefit< H(V_i^(S)) && temp≥ s(V_i^(S)) )
   evict from cache the objects in evict_list
   fetch and store V_i^(S)
   H(V_i^(S)) = L + B(V_i^(S))
ELSE
   O_k is the kth and last object added in evict list
   IF ( H(O_k) ≤ H(V_i^(S)) && s(O_k) ≥ s(V_i^(S)) )
      evict O_k and fetch V_i^(S)
      H(V_i^(S)) = L + B(V_i^(S))
   ELSE do not accept V_i^(S)
```

Fig. 4. Pseudocode of VHOW algorithm.

this reason, we built a limited HTTP server to act as a Web proxy. The Web proxy had a pool of 10,000 files of 35Kbytes each. Answering OLAP queries was the task of another server process, while requests were generated by a client process. We conducted the experiments using two distinct machines (one hosting the two server processes and the other the client process), connected over an Ethernet LAN. OLAP queries were modeled as aggregations over a certain number of rows. Every experiment involved a total of 10,000 requests sent by the client. We measured the average completion time for nonquery requests and compared it to the respective time if no queries existed in the system, i.e.,

$$ThroughputChange = \frac{TimeWithQueries - TimeNoQueries}{TimeNoQueries}.$$

Fig. 5 presents the results for two distinct request arrival rates (0.5 and 5 req/sec) as the number of rows each query aggregates increases. The OLAP traffic percentage was fixed to 10 percent. Two observations can be drawn from the figure. First, the throughput of the system on nonquery requests is significantly reduced only when the queries operate on relatively large views (e.g., 10 million rows). Second, the higher the overall request arrival rate, the higher the negative effects. However, in practice, it is quite unlikely that all queries will refer to large views. Nevertheless, if we are to prevent an even temporary throughput drop (presumably due to a burst on highly computation demanding queries) the proxy must avoid caching very large views. Recall that the algorithm amortizes the view benefit to its size, thus favoring small views for caching purposes. Fig. 6 illustrates how increasing the OLAP ratio affects the completion time of the remaining HTTP requests. The figure shows an almost linear increase of the completion time, while the negative effects are again more obvious for larger request arrival rates.

In the final experiment, we tested whether fetching views has an adverse effect on the system throughput. Fig. 7 shows the results as the view size increases for two different frequencies of view transfers. The results show that the system performance remains almost unaffected, although there is a small throughput decrease especially toward the end of the figure. Note that the results were obtained over a LAN. In practice, we expect that whenever fetching large views over the Internet is required, it will lead to a further performance decrease. This adds another factor for which caching large views must be avoided.

Overall, we conclude from the first series of experiments that a hybrid Web-OLAP proxy will likely exhibit a small

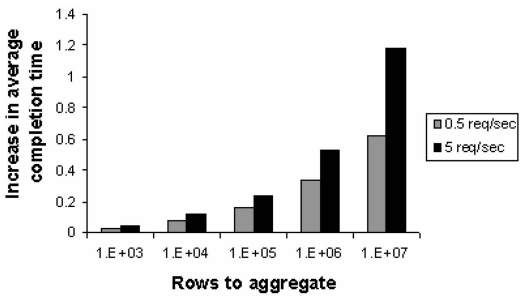


Fig. 5. Effects of query computation cost (Web:OLAP ratio = 90:10).

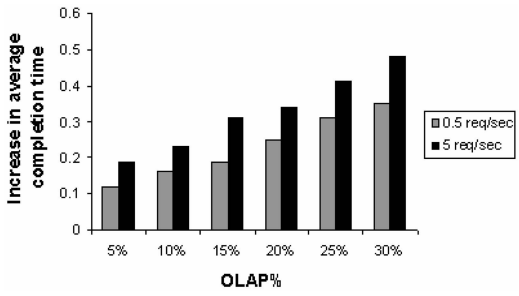


Fig. 6. Effects of OLAP request percentage (rows to aggregate = 100,000).

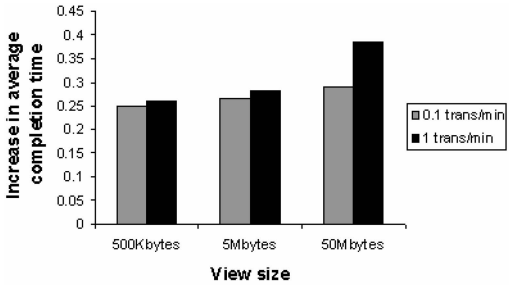


Fig. 7. Effects of view fetching (Web:OLAP ratio = 90:10, arrival rate = 5 req/sec, rows to aggregate = 100,000).

performance drop on the service of Web related traffic as compared to its pure Web counterpart. However, the negative effects can be restricted by avoiding loading the proxy with large views.

5.2 Simulation Workload

Here, we present the second series of experiments based on simulation runs. Three scenarios are considered for comparison: 1) a proxy that caches only normal Web pages using the GDSP algorithm, 2) a proxy that caches only OLAP views using the benefit function of (6) for replacement, and 3) a proxy that implements VHOW. We measure the performance of the alternatives, both in terms of Hit Ratio (HR) and in terms of Cost Saving (CS) defined as: $(WCost - PCost)/WCost$, where $Wcost$ is the cost occurred when no proxy is available and $Pcost$ the cost of each of the proxy implementations.

In order to simulate the environment, we generated representative workloads for both OLAP queries and Web requests. For the OLAP queries, we employed the data set from the APB benchmark [30]. The size of the entire cube was around 3.5GB of tuples (each tuple being roughly 20 bytes). For the Web traffic we used a synthetic workload with the page popularity following a Zipf distribution with 0.6 parameter and the size following a heavy tail one. The average page size was 35K and this is also the value we used for the static part of WOQP. We used a total of 2 million distinct Web pages, thus, equalizing the potential traffic from OLAP and Web.

OLAP queries were generated using: 1) a uniform distribution, i.e., the probability of a query to refer to a node in the lattice was equal for all nodes, and 2) the 80-20 rule, i.e., 20 percent of the lattice nodes (chosen as the ones at the lowest lattice levels) accounted for 80 percent of the queries (the remaining queries were distributed randomly). The first method represents the situation that is likely to appear when multiple users with diverse interests rest behind the same proxy, while the later depicts the case when users share common preferences. The final input stream consisted of 10 million requests and was created by first generating the pure Web traffic according to the page popularities and randomly combining it with the OLAP queries.

Since the views materialized at the DW server affect the query costs and the caching decisions, we decided to employ the VHOW at the server side, too. Furthermore, we allowed the server to cache only 10 percent of the data-cube (total size of views). Note that the only factor that burdens the materialization of all the data-cube is the storage capacity, i.e., we do not take into account update constraints.

The query cost of a view (Fig. 3a) was assumed to be the same (proportional to its size), regardless of whether it was cached at the proxy or at the DW server. Strictly speaking, this implies equal processing capabilities for the DW server and the proxies, which is not expected to be the case in practice. However, we were interested to test the scenario whereby if the same answering view is cached at both the proxy and the DW server, then the query is directed to the proxy. This is a reasonable policy to follow whenever offloading the DW server is of primary concern since it only directs queries to the DW in case 1) the proxy is unable to answer, or 2) the DW has a lower level (in the lattice) cached view that can answer the query.

5.3 Simulation Results

Intuitively, caching OLAP data into the proxy server pays off when there is a substantial amount of OLAP requests. In the first set of experiments, the goal is to identify the ratio of OLAP to common Web requests above which, VHOW is beneficial. Fig. 8 and Fig. 9 compare the alternatives of caching only normal Web pages (GDSP), OLAP views (OLAP), and both using VHOW. The cache size is fixed to be 1 percent of the data-cube, which also amounts to roughly 1 percent 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 0 percent to 100 percent. Figs. 8a and 8b show the hit ratio achieved by the algorithms when OLAP queries follow a uniform distribution and the 80-20 rule, respectively.

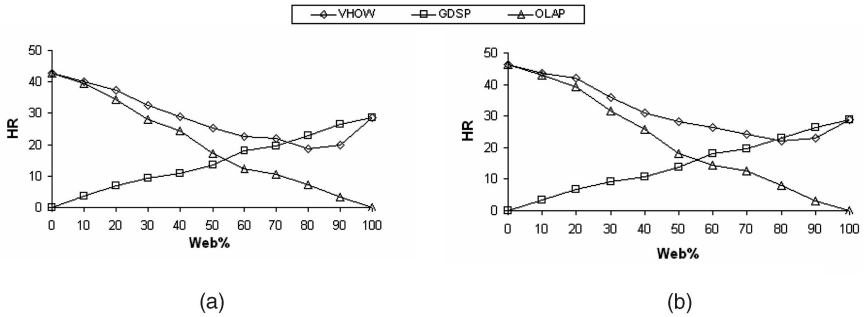


Fig. 8. Hit ratio versus percentage of Web requests (1 percent cache). (a) Uniform queries and (b) 80-20 queries.

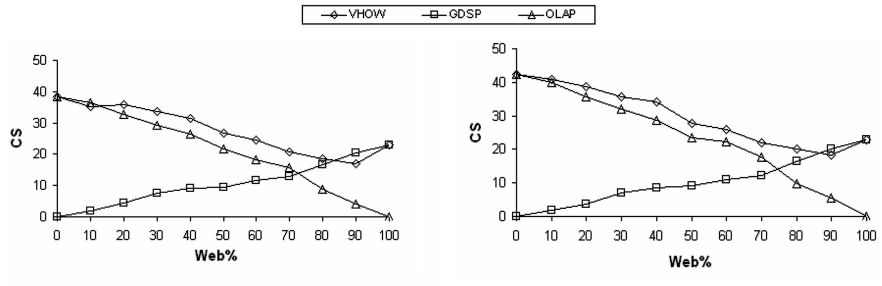


Fig. 9. Cost saving versus percentage of Web requests (1 percent cache). (a) Uniform queries and (b) 80-20 queries.

As expected, the performance of GDSP increases to the percentage of Web requests, while OLAP follows an inverse trend. The maximum hit ratio achieved by OLAP is higher by about 13 percent compared to the one of GDSP. This is because view interdependencies enable the proxy to answer queries even if it does not have the referred view cached (by deriving it from a higher level view as explained in Section 2). VHOW combines the merits of both GDSP and OLAP and, not surprisingly, outperforms for the biggest part the alternatives. Note that when the Web percentage is 80 percent or higher (Fig. 8a) VHOW achieves slightly worse hit ratio compared to GDSP. This is because OLAP views are usually larger in size compared to Web pages and, therefore, caching them, results in evicting many pages. With only 20 percent of the requests being OLAP oriented, the initial overhead of storing OLAP views is not amortized (in hit ratio terms). When OLAP queries follow the 80-20 rule, the impact of the initial overhead for caching OLAP views is smoothened as it can be shown in Fig. 8b. The same figure also shows that when queries are more "concentrated," OLAP and, subsequently, VHOW achieve higher hit ratios compared to the uniform case.

Fig. 9 shows the performance of the algorithms in terms of cost saving. The trends of the plots are the same as in Fig. 8. Observe the cost difference between OLAP queries and Web page accesses (GDSP and OLAP plots cross each other at around 70 percent in Fig. 9a). This results in VHOW being only marginally worse to GDSP when the Web traffic accounts for 90 percent of the total and significantly outperforming both alternatives in the rest of the cases. As in the hit ratio case, the 80-20 rule results in increased cost savings for VHOW (Fig. 9b). Fig. 10 shows the performance of the different policies when the cache size is 10 percent. With the cache size being substantial, VHOW totally outperforms the other two strategies since enough space is provided to cache both the most beneficial views and the most popular pages. The plots for the hit ratio and the 80-20 rule follow similar patterns and are not included. In order to gain further insight on the behavior of VHOW, we measured the average storage percentage allocated to OLAP views throughout a simulation run.

Fig. 11 presents the results for the two query generation strategies when the cache size is fixed to 10 percent. The first observation is that uniform queries lead to higher storage allocation for OLAP data, compared to queries following the 80-20 rule. This can be explained since when queries are evenly distributed to the lattice nodes, a larger number of views becomes beneficial for caching purposes, while on the other hand the 80-20 rule implies that caching 20 percent of the views is sufficient. From Fig. 11, we also observe that the storage space allocated to OLAP is disproportionably larger than its traffic when the majority of the requests are from Web (70 percent and more). The reason being the view sizes are much larger compared to pages. This explains the slight performance advantage of GDSP over VHOW for very high values of Web traffic.

In the general case, the experiments show that there exists a threshold on the percentage of OLAP traffic, above which caching OLAP data provides substantial benefits. The value of this threshold depends on both the query distribution and the cache size. For a small cache size of 1 percent, it was found that when OLAP traffic exceeds 10-20 percent, caching OLAP views becomes beneficial. When the cache size is 10 percent, the performance of VHOW is marginally worse compared to GDSP in the case where the Web traffic accounts for more than 94 percent of the total (not shown in Fig. 7). The results are encouraging since in a

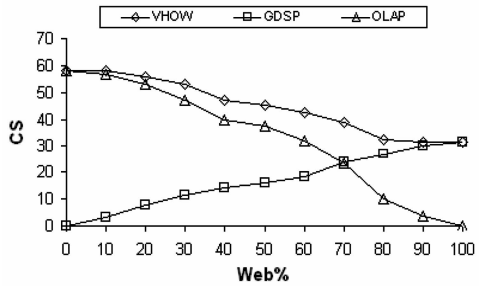


Fig. 10. Cost saving versus percentage of Web requests (uniform queries, 10 percent cache).

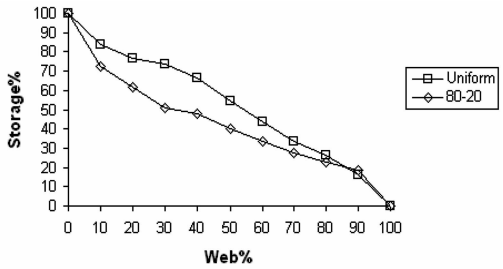


Fig. 11. Storage percentage allocated to OLAP versus percentage of Web requests (10 percent cache).

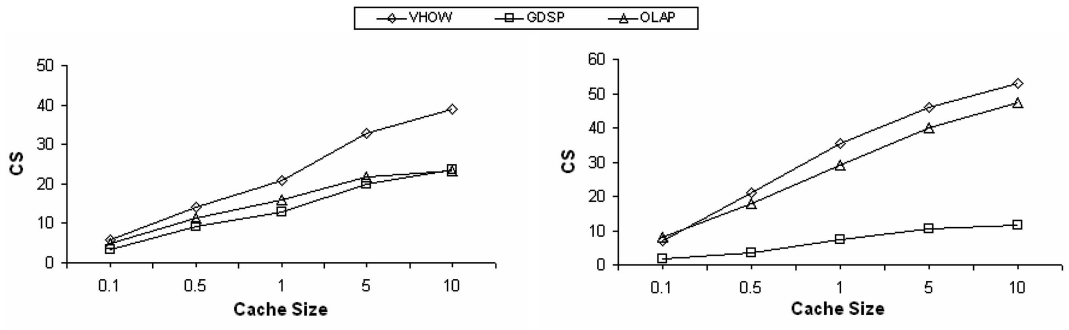


Fig. 12. Cost saving versus cache size (uniform queries). (a) Web:OLAP ratio 70:30 and (b) Web:OLAP ratio 30:70.

decision making environment such OLAP traffic levels can be easily reached. Notice that we did not record any case in our simulations where OLAP outperformed VHOW, probably due to the fact that Web pages follow a Zipf distribution and, therefore, even when traffic is OLAP predominant, caching the most popular Web pages is still advantageous.

In the second set of experiments, we tested the performance of VHOW when cache size varies between 0.1 percent and 10 percent of the total data-cube size. The network transfer rate is again fixed to 32KBps and the percentage of OLAP requests is 30 percent (Fig. 12a) and 70 percent (Fig. 12b). The performance for all algorithms increases to the available cache size. We observe that VHOW follows the same trend as GDSP when 70 percent of the load comes from the Web and the same trend as OLAP when 70 percent is for queries, while maintaining a clear lead against both OLAP and GDSP. Another observation is that the rate of performance increase for the algorithms tends to drop (increasing the cache size from 1 percent to 10 percent accounts for an increase in performance that is comparable to the one achieved by moving from 0.1 percent to 1 percent).

In the final experiment, we investigate the performance of the algorithms as a function of the transfer rates between the proxy and the central DW. The ratio of Web requests was fixed to 70 percent (solid lines) and 30 percent (dashed lines), queries followed a uniform distribution, and the network transfer rate varied from 32 KBps to 4 MBps. Fig. 13 a presents the results for 1 percent cache size, while Fig. 13b for 10 percent. 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 data-cube 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 VHOW. Since this behavior is due to OLAP traffic, the performance drop is more prominent in the 30 percent Web case (dashed lines), while GDSP's performance remains almost constant.

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

6 Related Work

A relevant previous work is on the view selection problem [15] 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 [12]. The

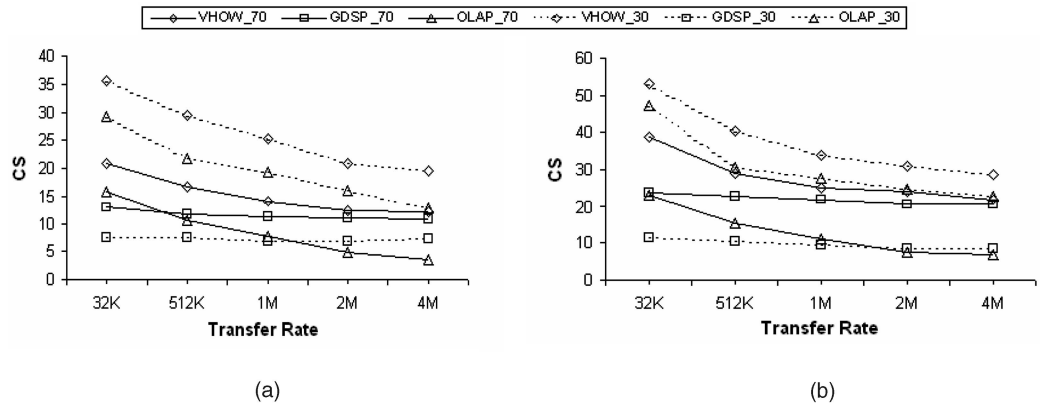


Fig. 13. Cost saving versus network transfer rate for different Web:OLAP ratios (uniform queries). (a) 1 percent cache and (b) 10 percent cache.

approximation algorithm achieves in the worst-case solution quality within 63 percent of the optimal. In [5], the search space of the problem is reduced by a heuristic that excludes views irrelevant to the most frequent queries. Another method for view selection, which is based on sorting [34], has smaller computational overhead than the method proposed in [15], while ensuring the same lower worst case bound provided that the view sizes satisfy certain conditions. In [36], 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 [20], the authors propose a method to dynamically materialize and maintain fragments of OLAP views with respect to both space and time constraints in a DW. In [21], 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 the fly. In [20] and [21], the authors propose caching algorithms that consider views as the only objects to be cached. Thus, they can suffer from cache pollution (i.e., previously popular documents fill the cache), if applied directly to a Web environment. The work in [21] assumes also transactional queries further differentiating it from our work. A normalized cost caching and admission algorithm for DW is presented in [32]. The same authors proposed similar caching algorithms for Web proxies in [33], without considering OLAP queries. Related also is the view-based query containment problem, i.e., defining which views can answer a query. Theoretical results for various problem formulations are presented in [8]. The authors in [2] tackle the problem from a template-based query caching perspective, while [4] and [28] consider the case of materialized XPath views. Here, once the query dimensions are defined, selecting an appropriate view to answer the query involves a search in the lattice (since only whole dimensions are considered for materialization).

In [26], active caching is employed to store database results in proxies, but only transactional (i.e., non-OLAP) workloads are considered. The same authors in [27] use a template-based approach to tackle the case of queries with embedded calls to user defined functions. The problem of caching OLAP results in a distributed environment is studied in [18]. The same authors in [19] consider the use of a P2P network for answering OLAP queries. Our work differs from both [18] and [19] since we study OLAP query caching under a Web context. Although in this paper we used the Web proxy paradigm to illustrate our framework, it is straightforward that our work is applicable in other cases too, e.g., at a CDN (Content Distribution Network) server should a CDN subscriber hosting a DW, deems necessary to offload its servers by shifting part of the query processing overhead to the CDN.

Various Web proxy caching algorithms exist in the Internet literature [3], [9], [17], [24], [33], [38]. Our approach is applicable in conjunction with these algorithms. Our aim is not to propose a new Web caching algorithm, but rather to provide a framework for caching OLAP views as well as illustrating the problems that rise and coping with them. Related to Web caching is the replica placement problem [16], [23], [31], where the aim is to allocate objects to proxies depending on past access frequencies. Although such problem formulations bear the merits of optimizing a global performance parameter (compared to the individual performance of each proxy), they often lead to static, centralized solutions, and are not widely deployed on the Internet (as opposed to caching proxies).

7 Conclusions

In this paper, we considered the problem of minimizing the cost of online analytical processing queries issued through the Internet. Instead of depending on dedicated machines with full DBMS capabilities, we consider the existing Web proxy architecture. 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 (VHOW) is developed that judiciously treats OLAP views and Web pages while taking into account the different costs involved in each case. Results from the simulation studies confirm the efficacy of the proposed framework, even when the ratio of OLAP queries to normal Web traffic is moderate. Naturally, deploying a distributed database system with sufficient replication capabilities would yield substantial performance benefits in such an environment. We consider this work complimentary to this option; for instance, when it is more cost effective due to lack of funds to use the already existing Web equipment.

Strategies that refresh parts of the cached views and invalidate others will most likely lead to increased performance and are part of our future work. Moreover, we can take advantage of the ICP (Internet Cache Protocol) [37] and the proxy hierarchies as described in [11] 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 him, instead of sending it to the central site. Research in both directions can be extended using the proposed framework, cost model, and caching algorithm.

Acknowledgments

The authors wish to thank anyonymous reviewers for their constructive suggestions.

References

[1] M. Abrams, C. Standridge, G. Abdulla, S. Williams, and E. Fox, "Caching Proxies: Limitations and Potentials," *Proc. Fourth Int'l World Wide Web Conf.: The Web Revolution*, pp. 119-133, Dec. 1995.

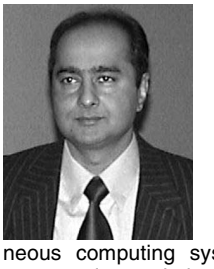
[2] K. Amiri, S. Park, R. Tewari, and S. Padmanabhan, "Scalable Template-Based Query Containment Checking for Web Semantic Caches," *Proc. 19th IEEE Int'l Conf. Data Eng. (ICDE '03)*, pp. 493-504, 2003.

[3] M. Arlitt, L. Cherkasova, J. Dilley, R. Friedrich, and T. Jin, "Evaluating Content Management Techniques for Web Proxy Caches," *Proc. ACM SIGMETRICS Performance Evaluation Rev.*, vol. 27, no. 4, pp. 3-11, Mar. 2000.

[4] A. Balmin, F. Ozcan, K. Beyer, R. Cochrane, and H. Pirahesh, "A Framework for Using Materialized XPath Views in XML Query Processing," *Proc. 30th Int'l Conf. Very Large DataBases (VLDB '04)*, pp. 60-71, 2004.

[5] E. Baralis, S. Paraboschi, and E. Teniente, "Materialized View Selection in a Multidimensional Database," *Proc. 23rd Int'l Conf. Very Large Data Bases (VLDB '97),* pp. 156-165, 1997.
[6] P. Barford, A. Bestavros, A. Bradley, and M. Crovella, "Changes in Web Client Access Patterns: Characteristics and Caching Implications," *World Wide Web J.,* vol. 2, nos. 1-2, pp. 15-28, 1999.
[7] A. Bestavros, "WWW Traffic Reduction and Load Balancing through Server-Based Caching," *IEEE Concurrency,* vol. 5, no. 1, pp. 56-67, Jan.-Mar. 1997.
[8] D. Calvanese, G. Giacomo, M. Lenzerini, and M. Vardi, "View-Based Query Containment," *Proc. 22nd ACM Symp. Principles of Database Systems (PODS '03),* pp. 56-67, 2003.
[9] P. Cao and S. Irani, "Cost-Aware WWW Proxy Caching Algorithms," *Proc. USENIX Symp. Internet Technology and Systems,* pp. 193-206, Dec. 1997.
[10] P. Cao, J. Zhang, and K. Beach, "Active Cache: Caching Dynamic Contents on the Web," *Proc. Middleware '98 Conf.,* pp. 373-388, Sept. 1998.
[11] A. Chankhunthod, P.B. Danzig, C. Neerdals, M.F. Schwartz, and K.J. Worrell, "A Hierarchical Internet Object Cache," *Proc. USENIX Technical Conf.,* pp. 153-163, Jan. 1996.
[12] H. Gupta and I.S. Mumick, "Selection of Views to Materialize Under a Maintenance-Time Constraint," *Proc. Int'l Conf. Database Theory (ICDT '99),* pp. 453-470, 1999.
[13] J.S. Gwertzman and M. Seltzer, "The Case for Geographical Push-Caching," *Proc. Fifth Workshop Hot Topics in Operating Systems (HotOS-V),* pp. 51-55, 1995.
[14] J. Hammer, H. Garcia-Molina, J. Widom, W. Labio, and Y. Zhuge, "The Stanford Data Warehousing Project," *IEEE Data Eng. Bull.,* vol. 18, no. 2, pp. 41-48, 1995.
[15] V. Harinarayan, A. Rajaraman, and J.D. Ullman, "Implementing Data Cubes Efficiently," *Proc. ACM SIGMOD Int'l Conf. Management of Data,* pp. 205-216, 1996.
[16] S. Jamin, C. Jin, Y. Jin, D. Riaz, Y. Shavitt, and L. Zhang, "On the Placement of Internet Instrumentation," *Proc. IEEE INFOCOM '00 Conf.,* pp. 295-304, Mar. 2000.
[17] S. Jin and A. Bestavros, "Popularity-Aware Greedy Dual-Size Web Proxy Caching Algorithms," *Proc. 20th IEEE Int'l Conf. Distributed Computing Systems (ICDCS '00),* pp. 254-261, Apr. 2000.
[18] P. Kalnis and D. Papadias, "Proxy-Sever Architectures for OLAP," *Proc. ACM SIGMOD Int'l Conf. Management of Data,* pp. 367-378, 2001.
[19] P. Kalnis, W. Siong, B. Ng, C. Ooi, D. Papadias, and K.L. Tan, "An Adaptive Peer-to-Peer Network for Distributed Caching of OLAP Results," *Proc. ACM SIGMOD Int'l Conf. Management of Data,* pp. 25-36, 2002.
[20] Y. Kotidis and N. Roussopoulos, "DynaMat: A Dynamic View Management System for Data Warehouses," *Proc. ACM SIGMOD Int'l Conf. Management of Data,* pp. 371-382, 1999.
[21] A. Labrinidis and N. Roussopoulos, "WebView Materialization," *Proc. ACM SIGMOD Int'l Conf. Management of Data,* pp. 367-378, 2000.
[22] N. Laoutaris, V. Zissimopoulos, and I. Stavrakakis, "On the Optimization of Storage Capacity Allocation for Content Distribution," *Computer Networks,* vol. 47, no. 3, pp. 409-428, 2005.
[23] B. Li, M. Golin, G. Italiano, and X. Deng, "On the Optimal Placement of Web Proxies in the Internet," *Proc. IEEE INFOCOM '99 Conf.,* pp. 1282-1290, 1999.
[24] P. Lorenzetti, L. Rizzo, and L. Vicisano, "Replacement Policies for a Proxy Cache," *IEEE/ACM Trans. Networking,* vol. 8, no. 2, pp. 158-170, Apr. 2000.
[25] T. Loukopoulos, P. Kalnis, I. Ahmad, and D. Papadias, "Active Caching of On-Line-Analytical-Processing Queries in WWW Proxies," *Proc. 30th Int'l Conf. Parallel Processing (ICPP '01),* pp. 419-426, Sept. 2001.
[26] Q. Luo, J.F. Naughton, R. Krishnamurthy, P. Cao, and Y. Li, "Active Query Caching for Database Web Servers," *Proc. Int'l Workshop Web and Databases (WebDB),* pp. 92-104, 2000.
[27] Q. Luo and W. Xue, "Template-Based Proxy Caching for Table-Valued Functions," *Proc. Ninth Int'l Conf. Database Systems for Advanced Applications (DASFAA '04),* pp. 339-351, 2004.
[28] B. Mandhani and D. Suciu, "Query Caching and View Selection for XML Databases," *Proc. 31st Int'l Conf. Very Large Databases (VLDB '05),* pp. 469-480, 2005.
[29] S. Martello and P. Toth, *Knapsack Problems: Algorithms and Computer Implementations.* John Wiley and Sons, 1990.
[30] OLAP Council, "OLAP Council APB-1 OLAP Benchmark, Release II," http://www.olapcouncil.org, 2001.
[31] L. Qiu, V. Padmanabhan, and G. Voelker, "On the Placement of Web Server Replicas," *Proc. IEEE INFOCOM '01 Conf.,* pp. 1587-1596, Apr. 2001.
[32] P. Scheuermann, J. Shim, and R. Vingralek, "WATCHMAN: A Data Warehouse Intelligent Cache Manager," *Proc. 22nd Int'l Conf. Very Large Databases (VLDB '96),* pp. 51-62, 1996.
[33] J. Shim, P. Scheuermann, and R. Vingralek, "Proxy Cache Algorithms: Design, Implementation and Performance," *IEEE Trans. Knowledge and Data Eng.,* vol. 11, no. 4, pp. 549-562, July/Aug. 1999.
[34] A. Shukla, P.M. Deshpande, and J.F. Naughton, "Materialized View Selection for Multidimensional Data Sets," *Proc. 24th Int'l Conf. Very Large Databases (VLDB '98),* pp. 488-499, 1998.
[35] W.R. Stevens, *TCP/IP Illustrated,* vol. 3. Addison-Wesley, 1996.
[36] D. Theodoratos and T.K. Sellis, "Data Warehouse Configuration," *Proc. 23rd Int'l Conf. Very Large Databases (VLDB '97),* pp. 126-135, 1997.
[37] D. Wessels and K. Claffy, "Internet Cache Protocol (ICP) Version 2," RFC2186, 1998.
[38] R. Wooster and M. Abrams, "Proxy Caching that Estimates Page Load Delays," *Proc. Sixth Int'l World Wide Web Conf.,* pp. 977-986, Apr. 1997.
[39] N.E. Young, "On-Line Caching as Cache Size Varies," *Proc. Symp. Discrete Algorithms (SODA '91),* pp. 241-250, Jan. 1997.

Thanasis Loukopoulos received a diploma in computer engineering and informatics from the University of Patras, Greece, in 1997. He was awarded the PhD degree in computer science by the Hong Kong University of Science and Technology (HKUST) in 2002. After receiving the PhD, he worked as a visiting scholar in HKUST. Currently, he is a visiting lecturer in the Department of Computer and Communication Engineering of the University of Thessaly, Greece. His research interests include data management in content distribution networks, video servers, P2P, and ad hoc networks.



Ishfaq Ahmad received the BSc degree in electrical engineering from the University of Engineering and Technology, Lahore, Pakistan, in 1985, and the MS degree in computer engineering and a PhD degree in computer science from Syracuse University, New York, in 1987 and 1992, respectively. His recent research focus has been on developing parallel programming tools, scheduling and mapping algorithms for scalable architectures, heterogeneous computing systems, distributed multimedia systems, video compression techniques, and grid computing. His research work in these areas is published in close to 200 technical papers in refereed journals and conferences. He is currently a full professor of computer science and engineering in the CSE Department of the University of Texas at Arlington (UTA). At UTA, he leads IRIS (Institute for Research In Security), a multidisciplinary research center engaged in safety and security related technologies. He is an associate editor of *Cluster Computing*, *Journal of Parallel and Distributed Computing*, *IEEE Transactions on Circuits and Systems for Video Technology*, *IEEE Concurrency*, and *IEEE Distributed Systems Online*. He is a senior member of the IEEE.