WPI-CS-TR-02-07 Feb 2002

Prefetching For Visual Data Exploration

by

Punit R. Doshi

Elke A. Rundensteiner

Matthew O. Ward

Daniel Stroe

Prefetching For Visual Data Exploration [*]

Punit R. Doshi, Elke A. Rundensteiner, Matthew O. Ward and Daniel Stroe

Department of Computer Science
Worcester Polytechnic Institute
Worcester, MA 01609–2280
{punitd|rundenst|matt}@cs.wpi.edu

Abstract

More and more modern computer applications from business decision support to scientific data analysis utilize visualization techniques to support exploratory activities for large datasets. Various tools have been proposed in the past decade that help users better interpret data using such display techniques. However, such exploratory visualization tools do not scale well when applied to huge datasets. Various features provided by database management systems must be applied to such applications to scale them for huge datasets. To improve the performance of such visualization systems, caching the data at client side is necessary. We exploit semantic caching [24, 8] for the advantages it offers over the traditional caching systems. To further improve the performance, we propose to exploit characteristics of visualization environments to prefetch the data for the visualization tools. We have incorporated these features into XmdvTool [47, 13, 14, 50], a freeware visual tool for multivariate exploration. We also compare an array of different prefetching strategies to determine their relative effectiveness for both synthetic user traces and real users of our system. Our results show that significant improvement can be achieved for visualization applications by caching and prefetching the data on the client-side.

Keywords: Semantic caching, Prefetching, Large-scale multivariate data visualization, Exploratory data analysis, Hierarchical data exploration

[*]This work is supported in part by several grants from NSF, namely, NSF grant IIS-9732897, NSF CISE Instrumentation grant IRIS 97-29878, and NSF grant IIS-0119276.

1 Introduction

Whether the domain is stock data, scientific data, or the distribution of sales, visualization is becoming an increasingly popular technique for data exploration. Humans can often easily detect patterns, trends as well as outliers in the underlying data when presented with visual depictions of the data [38], which may be more difficult to identify with automatic techniques. The exploration of large information spaces remains a challenging task especially with the growth of the World Wide Web and other huge repositories of information. Visualization plays an important role in aiding users to find their way through such large data sets. By presenting information visually and allowing dynamic user interaction through direct manipulation paradigms, it is possible to traverse larger information spaces in a shorter time [38]. In both academia and industry alike, significant effort has thus been spent on developing effective methods to display and visually explore information [1, 40, 39, 37, 32, 16, 36, 20, 30].

Most visualization techniques nowadays still execute on data that is first fetched from the file system into main memory. However, as data is being generated at an ever increasing rate and typical sizes of datasets become larger in the order of giga-bytes, current datasets can no longer be held entirely in main memory, rendering many current visualization tools useless.

We thus must scale current visual tools to work with large data sets. Issues related to efficient storage and retrieval of data, while often ignored in the context of visualization applications, are critical for the success of modern exploration tools. In such interactive environments the user must get the response to his or her navigation requests with little or no time lag. Furthermore, users wish to interactively explore the data using visual navigation tools. Each small movement in the user's navigation tool may mean executing a completely new query to retrieve the selected data, potentially resulting in a high data access rate.

In this work, we propose to exploit characteristics of visualization environments to improve the performance of visualization tools. An appropriate memory organization is also one critical component in interactive applications, since it influences the performance of the subsequent operations of maintaining the cache. In this vein, we apply semantic caching techniques [24, 8] for the advantages it offers, namely flexibility of grouping of data in the cache to be adjusted to the needs of the current query and saving lookup overhead in the cache due to the compact query-based organization of the cache content.

In addition, to further improve the performance of our system, we have designed several methods for prefetching data tuned to visualization applications. These strategies, working hand in hand with the query-based caching scheme, exploit the characteristics of the visualization environment such as the incremental refinement nature of queries formulated via a visual query tool in order to optimize the contents of the cache. We support features necessary for visualization, by making our prefetching solutions speculative and non-pure.

The proposed caching and prefetching techniques have been incorporated into XmdvTool [47, 13, 14, 50], a freeware visual tool for multivariate exploration. We have also ran experiments to evaluate the performance of the prefetching strategies both with various synthetic user traces as well as with real users of our system. Results show that the proposed strategies indeed improve the performance of the system, making visual exploration of large-scale data practical.

In summary, our main contribution consists of developing a set of techniques that together can be applied to interactive visualization tools in order to enable users to explore large datasets. We have designed a high-level cache policy that reduces the latency of the system by incrementally loading and maintaining data in the memory buffer. When the system is idle, a prefetcher will bring data into the cache that is likely to be used next. Our experiments confirm the important role of prefetching in visualization applications and demonstrate that the benefit of using prefetching significantly exceeds the result gained by using caching only.

Section 2 discusses the characteristics of the interactive visualization environments. Section 3 explains our approach to semantic caching, while Section 4 introduces our prefetching strategies. In Section 5 we discuss the XmdvTool visualization system, while Section 6 focuses on the experimental evaluation of our prefetching strategies. Section 7 presents the work related to our research and finally we state conclusions and some areas for future work in Section 8.

2 Multivariate Data Visualization

Here we briefly introduce XmdvTool, an exploration tool for multivariate data, which represents the driving force motivating the need for this work as well as representing the testbed into which we incorporate and then evaluate our solutions. XmdvTool is a visualization package developed at WPI [50] designed for the exploration of multivariate data which provides four distinct visualization techniques and support for clustering and analyzing hierarchies [47, 13, 14]. The tool provides four distinct visualization techniques (parallel coordinates, scatterplot matrices, glyphs, and dimensional stacking) with interactive selections and linked views, as depicted in Figure 1.

The main idea of XmdvTool is to help users to understand the data by first clustering the data points in the navigation space based on a distance function, and then to associate aggregate information with the resulting clusters [47, 31, 34, 12]. The clustering process generates a hierarchy in which different levels conceptually represent different degrees of abstraction of the data. Brushing then consists of setting the selection parameters and in specifying the desired *level of detail* at which data is displayed.

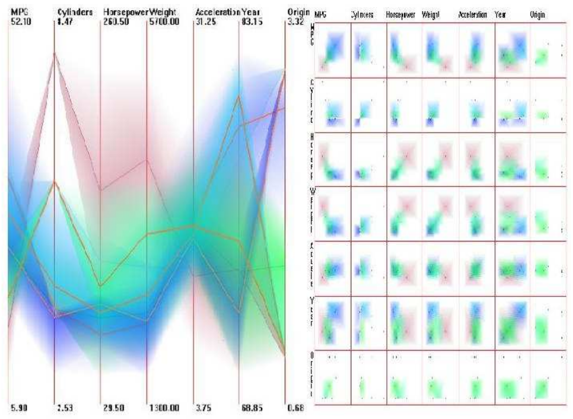


Figure 1: Various displays in XmdvTool.

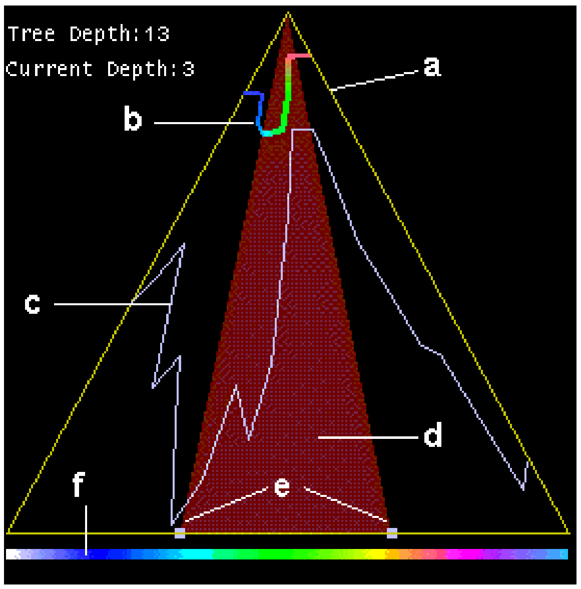


Figure 2: Structure-based brush.

In order to improve the support for visual navigation across large datasets we have designed the structure-based brush [11] tool shown in Figure 2 that supports hierarchical exploration of data for all four multi-resolution visualization techniques. The data can then be explored by interactively selecting and displaying points at different levels of detail of the cluster hierarchy. We term this exploration process *hierarchy navigation*. The user navigation operations expressed by our brushing tool interface (Figure 2) get translated into queries to the database. Brushing tool marked '*e*' is used to select cluster(s) to be displayed, while tool marked '*b*' is used to select the level of detail for the selected cluster(s). A user can select data by moving the brush marked as '*e*' horizontally or moving the brush marked as '*b*' vertically, as shown in Figure 2.

While exploring the data, a user may navigate by sliding the extents of '*e*' to select a particular cluster in the tree hierarchy, or by moving vertically the level brush '*b*' to display data at a different level of detail. Thus the queries passed to the database are contiguous rather than ad-hoc, since the visual interface provides limited and controlled means of expressing navigational requests from the users. Such contiguity of user queries allows us to cache the queries in main memory, as there is a high probability of a partial query result from a prior query still being relevant (and thus in our cache) for the new user request.

Lastly note that users' exploratory movements are somewhat more predictable when they explore the data using such visualization tools, as such explorations are different from say, random accesses via an ad-hoc SQL query based interface. This gives us the hint that the use of prefetching in such applications may be a suitable mechanism for improving the performance of visualization applications.

Since the user will be examining the visual displays for interesting patterns in the data, there typically would be delays between two user operations. These delays between user operations allow us to prefetch the highly probable data into the main memory before the user explicitly specifies her next request.

3 Semantic Caching for Visualization Applications

Semantic caching is a popular caching strategy proposed in recent years [24, 8] for providing efficient support for access to data. In contrast to traditional caching schemes [10, 25], it caches query descriptors rather than pages of data or individual objects. It provides the following benefits over the traditional syntactic caching approaches:

- adjusts grouping of queries to the requirements of the incoming query so that no irrelevant data is cached along with the relevant ones, thus reducing overhead in managing the cache,
- minimizes the cost of cache lookup due to the compact representation of the cache content based on semantic query descriptors,
- adapts dynamically to the patterns of user queries rather than just caching static clusters of tuples.

A semantic caching scheme must typically handle the following three tasks. First, it has to be able to decide whether the answer for a query resides in the cache or not by comparing the incoming user query with the cached query descriptors, called query containment. Second, the partial answer available in the cache must be extracted by formulating appropriate probe queries. Third, it must determine any remaining query that needs to be passed to the server to fetch the remaining data. Although concrete steps towards finding common techniques that work in the general case have been taken [15], the task of resolving these issues is still generally performed on an application-by-application basis.

3.1 Containment Issues

Semantic caching schemes exploit the capability of semantic descriptors to describe the content of the cache in a concise form. This allows for a fast look-up, since only a few set-based operations are performed instead of separately checking the containment for all individual objects contained in the cache against the user query. However, these approaches then must assume either explicitly or implicitly the following. First, that it is always possible to test whether the answer for the query can be found in the cache (i). Second, that it is possible to extract the answer from the cache (ii). And third, that it is possible to compute the difference (at the query level) between the requested query and the cache predicates (iii). These assumptions define the area of applicability of semantic caching, since they are not always true (see for instance [15]). In our visual application, we are able to map our recursive hierarchical queries into range queries and then can address these three questions.

3.2 Operational Model of Semantic Caching

An operational model for our semantic caching is presented in Figure 3. In this model, the client processes a stream of queries $Q_1, \ldots, Q_N$. Let $C_i = C_1^i \cup \ldots \cup C_k^i$ denote the cache content in terms of queries at the time query Q_i is issued, $Q_0 = \emptyset$, and $O(C_i) = O(C_1^i), \ldots, O(C_n^i)$ denote the objects that correspond to the cache descriptor C_i. The cache descriptors C_j^i are queries, but could in principle be any form of set descriptors. Objects have individual replacement values, in our case determined by some probabilistic function.

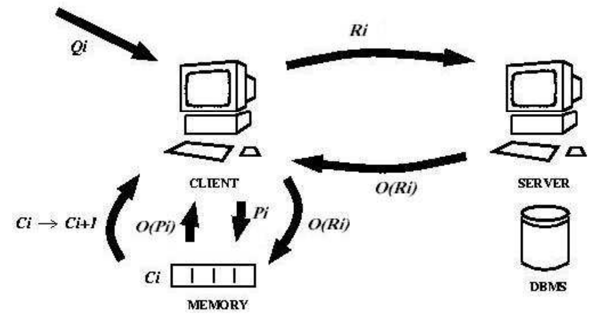


Figure 3: Operational Model for Semantic Caching.

Processing query Q_i involves the following steps:

- Compute the probe query $P_i = Q_i \cap C_i = (Q_i \cap C_1^i) \cup \ldots \cup (Q_i \cap C_k^i)$.
- Extract from cache objects $O(P_i) = O(P_1^i), \ldots, O(P_m^i)$ that correspond to the constraint formula P_i, i.e., answer Q_i partially from the set of tuples that satisfy P_i.
- Compute the remainder query $R_i = (Q_i - C_1^i) \cup \ldots \cup (Q_i - C_k^i)$.

- Fetch the tuples $O(R_i)$ that satisfy the constraint formula R_i from the server from C_i to C_{i+1}. Update C_i to reflect the changes; this may result in unifying (merging) descriptors.
- If the cache does not have enough free space, discard objects O^i_j in the decreasing order of their replacement value until enough space is free. Adapt C_i to reflect the changes; this may result in fragmenting (splitting) descriptors.
- Update the replacement values of all objects O_i based on R_i, P_i and the replacement policy.

3.3 Replacement Issues

The first step in implementing a replacement policy is to provide an estimation strategy able to measure the likelihood that an object will be needed in the near future. The estimation strategy, also called a *predictor*, is usually based on heuristics, probabilistic models, or some recorded statistics. In our case we use a probability function. The probability function also defines a partition on the set of objects.

The main task of a cache replacement policy is to find the entries in the buffer that have the lowest probability of being used and to remove them when more room is needed. This operation needs to be efficient, since it occurs frequently. When new objects are brought in they have to comply with the internal cache organization.

When a request is issued by the GUI, a containment test is performed. The system first checks whether the requested data resides entirely in memory or not. In case it doesn't, a *compensation* query has to be sent to the *loader*, an agent that fetches the data from the persistent storage.

In conclusion, the buffer access operations can be summarized as:

A: **Remove old objects.** Get the objects with the lowest probability that reside in the buffer (and further remove them one at a time when more room in the buffer is needed).

B: **Retrieve new objects.** Place an object from the database *cursor* into the memory buffer (and rehash the buffer entry).

C: **Display active set.** Get those objects from the buffer that form the active set (and send them to the graphical interface to have them displayed).

D: **Recompute probabilities.** Recompute the probabilities of the objects in the buffer once the active window gets changed (to ensure accurate predictions in the future).

E: **Test containment.** Test whether the new active set fully resides in the buffer and get the missing objects (if any) from the database (when a new request is issued).

Unlike semantic caching, we have made the caching system flexible enough to replace objects in the cache rather than queries in order to ensure that the cache is full almost all the time. This also adds an additional overhead of keeping probability values for every object rather than individual query.

4 Prefetching Strategies

To further reduce system latency, we use a speculative prefetcher that brings data into memory when the system is idle. The prefetcher is based on the property of exploratory systems that queries remain "local", i.e., given the set of currently selected objects we have a small number of choices for which objects can be selected next. The property therefore provides "implicit hints" to the system. Additional hints can be extracted from the data set characteristics, its usage over time and the user's exploration patterns as well. In what follows, we discuss different prefetching strategies and how we exploits the characteristics of visualization tasks.

4.1 Characteristics of our Prefetcher

In visualization applications, users spend a significant amount of time interpreting the graphical presentation of the selected data, and the processor and I/O system are typically idle during that period. If the computer can predict what data the user will request next, it can start fetching that data into the cache (if not already there) *before* the user asks for it. Thus, when the user requests that data later, he or she perceives a faster response time.

In some interactive database applications, there is sufficient time between user requests for such prefetching, and therefore the amount of data that can be prefetched is limited only by the cache size. This situation is referred to as *pure prefetching* and constitutes an important theoretical model in analyzing the benefit of prefetching. In our target visualization application and many others however, prefetching requests are often interrupted by further user requests, resulting in less data being prefetched at a time. In this case of *non-pure prefetching*, we also need to consider issues of cache replacement. We thus convert pure prefetching strategies into practical non-pure ones by combining them with cache replacement strategies. In [5] for instance, a pure prefetcher is used with the least recently used (LRU) cache replacement strategy, and a significant reduction in the page fault rate was shown. A multi-threaded implementation of a non-pure prefetcher is reported in

[43]. There, the latency of the disk operations is improved when using threads.

Visualization applications require prefetching strategies to be speculative, non-pure, and adaptive as explained below. Prefetching must be *speculative (on-line)* as decisions must be based strictly on the history. Without *apriori* knowledge or statistics of the user request patterns, as is the case of most interactive applications [7], prefetching must be speculative. An important requirement of speculative prefetching is that the time spent on making prefetching decisions must be minimal. Prefetching must be adaptive since the prefetching policy has to change due to run-time events. As the exploration goals and thus the access behaviour of a user may vary over time, changing when to issue prefetching requests or the amount of data to be prefetched may influence the performance of prefetching.

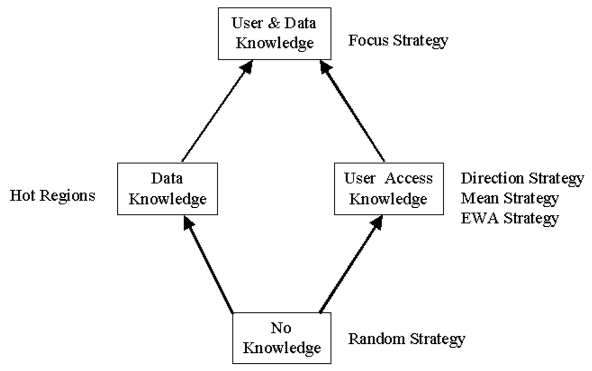


Figure 4: Hierarchy of Prefetching Strategies.

We designed and implemented several speculative, non-pure strategies for prefetching, as described below in order to perform comparative evaluation. As shown in Figure 4, our approach is to generate a hierarchy of prefetching strategies, based on different prefetching hints. We designed five prefetching strategies: *random* (S1), *direction* (S2), *focus* (S3), *mean* (S4), and *exponential weight average* (S5). In experiments, we also considered the case of not prefetching, the case referred to as S0.

4.2 Random Strategy

As shown in Figure 5, strategy S1 (random) is based on randomly choosing the direction in which to prefetch next. This strategy is appropriate when the predictor either cannot extract prefetching hints or provides hints with a low confidence measure. This is utilized initially when no other knowledge is available.

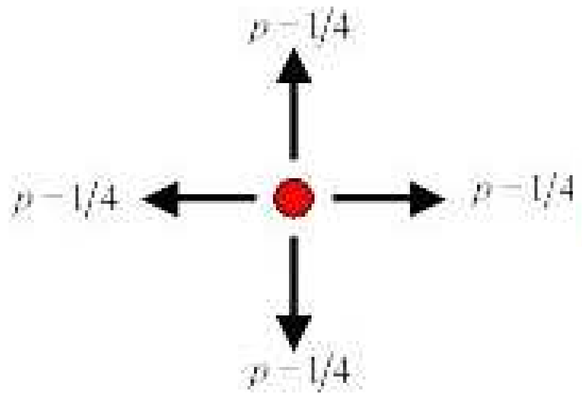


Figure 5: Random Strategy.

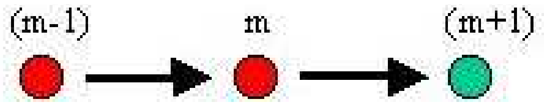


Figure 6: Direction Strategy.

4.3 Direction Strategy

Strategy S2 (direction) implies that the most likely direction of the next operation can be determined. Based on user's past explorations, the predictor would assign probabilities to all the four directions. The prefetching strategy (S2) then implies to "prefetch data in this given direction". The hypothesis that the next direction can be determined is not arbitrary. It is intuitive, for instance, that the user will continue to use the same manipulation tool for a while before changing to another one and (in our system, each manipulation tool happens to precisely control one direction only). As depicted in Figure 6, if $(m-1)$ and m are the last two locations navigated by the user, then the *direction* strategy may predict $(m+1)$ as the next location to be visited by the user.

4.4 Focus Strategy

Strategy S3 (focus) uses information about the most probable next direction (by keeping track of user's previous movement) as well as hints about regions of high interest in the data space as identified due to prior navigations of this same data by other users. This strategy will continue to prefetch data in the given direction using the above mentioned heuristics in Section 4.3. However, when a hot region is encountered the prefetcher adapts from the default *direction* prefetching and instead adapts prefetching in that new direction. The reason is that the user will likely stop there to explore or at least spend more time in that region.

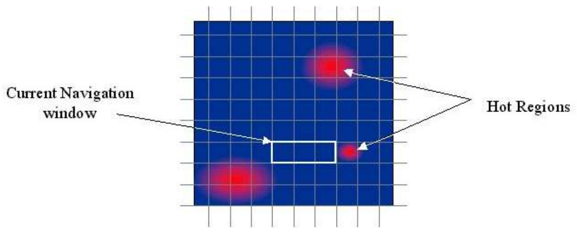


Figure 7: Hot Regions (Focus Points).

4.5 Vector Strategies

All the previous strategies do not take advantage of the history of past user explorations. The next two strategies are vector strategies that look at past user explorations.

In this model, we use a three-dimensional vector to indicate the movement of the users, one for the start of brush, one for the width of brush and the last one for the level of detail. To enable prefetching, we maintain user trace for each user, containing the set of historical movement vectors, $m_1, m_2, ..., m_{n-1}$. Each vector is calculated from the corresponding viewer's location and orientation, containing a move direction and a move distance. We predict the $n + 1$st movement vector m_{n+1} and prefetch objects that would be required if the user goes that way. This work is similar to a vector model of prefetching objects in distributed virtual environments introduced by Chim et al [3]. Basically, it looks at each movement of the viewer as a vector and computes the average of the previous movement vectors to predict the next movement. They propose three different methods to predict the next location of the viewer, called the *mean*, *window* and *exponential weighted average* methods.

We utilize two different schemes to predict the next location of the viewer: mean (S4) and exponential weighted average (S5), as depicted in Figures 8 and 9. We are embedding the *window* strategy into the previous strategies S4 and S5 by considering past user operations equal to the window size. Our experiments discussed in Section 6 have shown that large window size results in wrong data being prefetched.

In the mean scheme, the next movement vector is predicted to be the average of the previous n movement vectors. The intuitive meaning of the mean scheme depicted in Figure 8 is that we predict the $(n + 1)$st movement vector by averaging the previous n (in this example, $n = 3$) movement vectors. The magnitude of the movement is determined by the average of the magnitudes of the previous movements. Let us denote the movement vector in the nth step by m_n and the predicted movement vector for the next step by m_{n+1}.

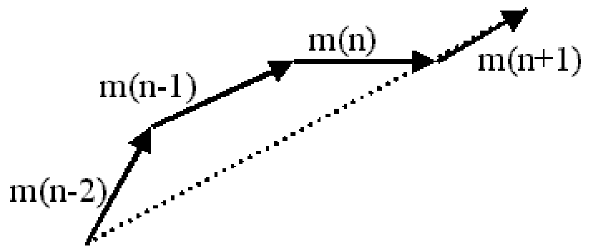


Figure 8: Mean Strategy.

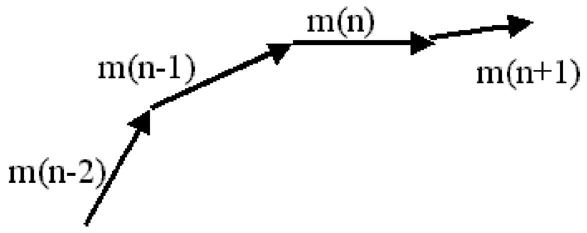


Figure 9: EWA Strategy.

The predicted vector will then be:

$$m_{n+1} = \sum_{i=1}^{n} m_i/n \tag{1}$$

To adapt quickly to changes in viewer's moving patterns, our second scheme assigns a weight to each previous movement vector m_i so that recent vectors have higher weights and the weights tail off as the vectors become aged. A parameter in the scheme is the exponentially decreasing weight, α. The most recent vector will receive a weight of 1; the previous vector will receive a weight of α; the next previous one will receive a weight of α^2, and so on. A high α will give similar weights to all the movements and predict future movements as a function of many movements, including the aged ones. By contrast using a low α, aged movements will fall off quickly and the prediction is biased towards contributions from recent movements. The predicted vector is:

$$m_{n+1} = \sum_{i=1}^{n} \alpha^{n-i} m_i/S_n \tag{2}$$

$$S_n = \sum_{i=1}^{n} \alpha^{n-i} \tag{3}$$

It can be shown that for both the mean and EWA strategies, the size of the history vectors (window) must be small. Larger values of the window tend to lower the valid data being fetched.

In Section 6 we perform experimental studies comparing these strategies. The results confirm our general assertion that prefetching is more efficient the more information we have available. Thus, changing the prefetching strategy adaptively as more patterns are discovered is likely to improve the overall system performance.

5 A Visualization Application Case Study: Applying Caching and Prefetching Strategies to XmdvTool

Given that one general algorithm for testing the containment or for extracting the answers from a cache do not exist (since the problems are undecidable), implementation of a semantic cache therefore remains a challenge for most applications. In what follows we present an implementation of our semantic caching scheme in XmdvTool. We first describe the visualization environment, then outline the characteristics of the queries and objects that we deal with, discuss the replacement function and show why it is non-fragmenting. Finally, we introduce the XmdvTool cache data structures main memory operations.

5.1 Objects and Queries

Previous research [41] has shown that hierarchical exploration via our brush in XmdvTool [11] can be modeled as a two-dimensional exploration in which a selection window, called the *active window*, slides over an $n \times m$ grid of integers, called the *navigation grid*. The objects (the data points or data clusters subject to analysis) have a spatial representation that makes them selectable by the active window.

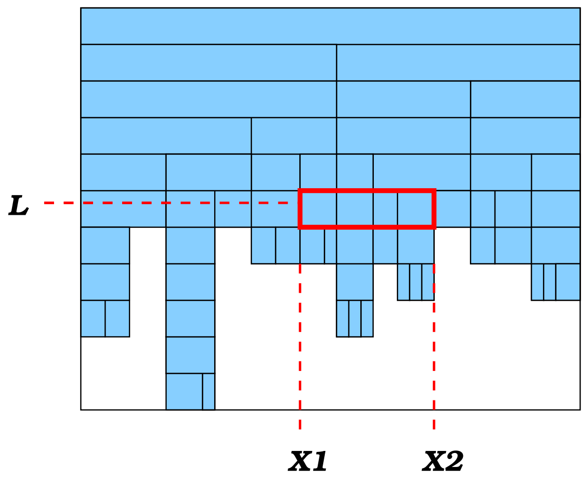
Objects are the data points or data clusters to be analyzed, to which some precomputed information is assigned in order to facilitate their visual manipulation. As shown in [41, 42], this additional information, consisting of a level value and two extents values, makes the objects behave like small rectangles $(e_1, e_2) \times L = (e_1, e_2, L)$, yet still preserving their hierarchical structure (Fig. 10).

Objects are thus similar to active windows: they are both rectangular regions of the form $(e_1, e_2) \times L$. The containment test of whether an object belongs to the active window or not reduces to an inclusion test between rectangles. Fig. 11 presents an example SQL query assuming that our $N - dimensional$ data is stored in a hierarchical table called $HIER(e_1, e_2, L, dim_1, dim_2, ..., dim_N)$, where e_1 and e_2 are the extents, L the level of detail, and $dim_1, dim_2, ..., dim_N$ the multidimensional values of the data points.

5.2 Replacement Function

The replacement values are given by a probability function that measures the likelihood that an object will belong to the active set in the near future. The function is based on a set of probability values assigned to the operations that can change an active window.

Let's consider a navigation grid $\Delta = (1..I) \times (1..K)$, where I and K are natural numbers. Each region



```
select  *
from    hier
where   e_1 >= :x_1
and     e_2 <= :x_2
and     L = :L;
```

Figure 10: Objects as rectangles in XmdvTool and an active window $A = (x_1, x_2, L)$.

Figure 11: SQL Queries in XmdvTool for active window $A = (x_1, x_2, L)$.

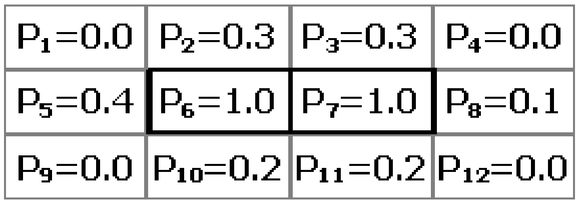
(e_i, L_k) from the navigation grid has an associated probability $\mathcal{P}(m, i, k)$ that measures the likelihood that the point will belong to the active set after the user's next m operations. Also, a probability $\mathcal{P}^*(m, i, k)$ will measure the likelihood that the point will belong to the active set at any time during the next m operations. Obviously, we have: $\mathcal{P}^*(m, i, k) = \oplus_{t=0}^{m} \mathcal{P}(t, i, k)$, where $\oplus$ is a probability sum, i.e., $p_1 \oplus p_2 = p_1 + p_2 - p_1 p_2$ (from the principle of inclusion and exclusion).

The *lookahead parameter* (LA) is the number of operations considered in advance when computing the probabilities $\mathcal{P}$ and $\mathcal{P}^*$, i.e., the parameter m from the definitions above. The LA parameter dictates how many operations the predictor will predict. In general, the bigger LA is the more speculative the system becomes and thus the more errors may occur. We used in our implementation an LA equal to 1.

In our case we have six possible operations (restricting or enlarging any of the three active window's parameters). Let us assume, for example, that we have an active window $\omega = (i_1, i_2, k)$ and from this configuration, going left with i_1 is 50% probable, going up with k is 25% probable, and so on. Then objects in (i_1, i_2, k) will have a probability of 1.0, objects in $(i_1 - 1, i_1, k)$ will have a probability of 0.50, objects in $(i_1, i_2, k - 1)$ a probability of 0.25, and so on.

5.3 The Cache Data Structure

Let us consider the navigation grid displayed in Fig. 12. We have here twelve regions of equal probability, and the active window covers the two middle ones. For simplicity we consider that only one object resides in



P_1=0.0	P_2=0.3	P_3=0.3	P_4=0.0
P_5=0.4	P_6=1.0	P_7=1.0	P_8=0.1
P_9=0.0	P_{10}=0.2	P_{11}=0.2	P_{12}=0.0

Figure 12: Navigation Grid.

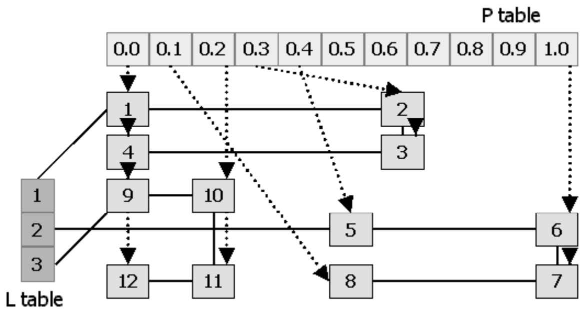


Figure 13: Cache Contents.

each region. We also number the objects from 1 to 12. The picture presents only three levels (1, 2, and 3). Assume probabilities are assigned to each region and implicitly to each object, based on a "operation-driven" probability model. Thus, objects 6 and 7 have a probability of 1.0 as they are in the active window. There is 40% chance that the window expands to the left, and so on. The corresponding cache content is shown in Figure 13. In this example a probability precision of 0.1 is assumed, and consequently 10 probability-based buckets are used.

5.4 Architecture of Visualization System

The system architecture depicted in Fig. 14 illustrates the key modules and interactions of our XmdvTool system that incorporates all the ideas described in this paper. First, as shown on the top of the figure, an off-line process transforms the hierarchical data into MinMax trees [41, 42], a precoded indexing structure allowing us to express hierarchical navigation as range queries as explained in Figure 11. The prepared data is then loaded into the database. The process implements the MinMax tree approach, the details of which are explained in [41]. Information about the database schema is used later during exploration.

When exploring, users interact via the graphical interface (GUI) shown on the right side. Details of the visual exploration interface have been given in Section 2. The visual navigation operations correspond to changes in the active set. When a change in the active set is detected, a *producer* thread is created, while the GUI itself acts as *consumer*. Basically, two threads operate concurrently on the buffer data: the consumer and the producer. The active set information from the GUI is passed to a *rewriter*. The rewriter consults the semantics of the buffer, expressed by a buffer query, and then generates a set of sub-requests to incrementally adjust the data in the buffer. Each sub-request is transformed into an SQL query by the *translator*, based on the database schema. The queries are passed to the *loader*, which fetches the necessary objects from database and places them in the buffer. From here, the *reader* reads them and sends them to

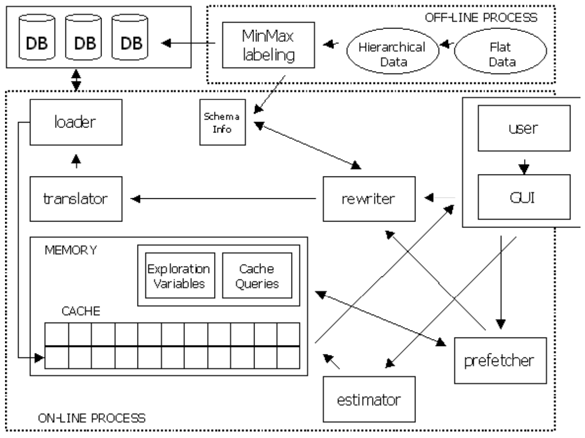


Figure 14: System architecture. Dashed-line rectangles show the separation between the on-line and the off-line computation. Solid-line rectangles represent the modules. Squares represent meta-knowledge. Solid arrows show the control flow.

the display. Once the *reader* is done with the reading of data, if time permits, the prefetcher will signal the *rewriter* to prefetch the next most probable data depending on the information that it has been gathered over time.

6 Experimental Evaluation

6.1 Settings

All experiments were conducted on an Alpha v4.0 878 DEC station, running Oracle 8.1.5. We used C as the host language and embedded SQL statements for accessing the data in the database. Throughout all phases of testing we used different types of datasets, both real and synthetic, with always consistent results. The eleven datasets we used have between 128 (same as the well-known IRIS benchmark) and 10 million datapoints, between 8 and 20 dimensions, and between 1,024 (2^{10}) and 65,536 (2^{16}) objects as the maximum number of points displayed at a time.

We have experimented with different randomly generated characteristics of the dataset, such as the number of focus regions, delay factor, "keep direction" factor, and some system parameters such as the prefetching strategy, hints to the query optimizer, and size of the data. In all experiments we have used navigation scripts containing between 300 and 3000 user operations. For synthetic users, these scripts were generated by a pseudo-random number generator code that mimicked the actual user traces we had observed. We confirmed the validity of the generated scripts by also experimenting with traces collected from various users of our system.

The values we measured during the experiments included *number of objects displayed* during a navigation session, *query-based hit ratio*, *objects-based hit ratio*, and *latency*. The display requests are queued and served when the system is idle. However, if two display requests come from the same GUI widget, the older one is cancelled. This behavior may result in lost information, when the display requests are very frequent and thus too close in time to one another. This is why we have considered the *number of objects* actually being displayed as a measure of the *visual quality*. The *hit ratio* is the number of objects already in the cache over the total number of objects requested from the back-end. The *latency* is the total time, expressed in milli-seconds, that the user had to wait for her requests to be served, i.e., the time required to fetch the data after the user-query has arrived. It also includes the time in which the user had to wait, but did not get the reply as the query was cancelled because of the navigation speed.

6.2 Experiments with Hot Regions

For this experiment, the user simulation files were generated varying the number of hot regions from 1 to 5. Figure 15 shows the performance of various strategies as the number of hot regions increases. The performance is measured on the Y-axis as normalized latency. Normalized latency for $Latency_n$ is given by the formula:

$$Latency_n = \frac{ActualLatency - MinLatency}{MaxLatency}$$

where $MinLatency$ and $MaxLatency$ are the minimum and maximum value of latency from the observed latency values.

Figure 16 is similar to this chart except that the performance is shown to indicate the performance in terms of percentage from zero latency. This normalized latency $Latency_n$ is given by the formula:

$$Latency_n = \frac{ActualLatency}{MaxLatency}$$

As the number of hot regions increases, most of the direction-oriented prefetching strategies deteriorate in performance as the user input tends to be a bit random in nature when the person tries to move in the direction of hot regions most of the time. The performance of the *focus* strategy is slightly lower initially because most of the time the user is not moving in the direction of the hot regions. Then the *focus* prefetching strategy improves in performance as it prefetches in the direction of those hot regions, thus prefetching the correct data a higher percentage of the time.

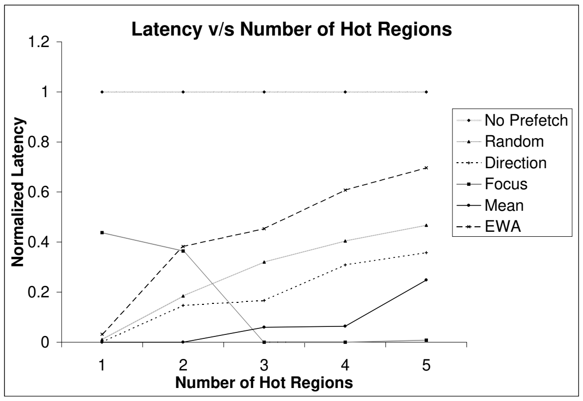


Figure 15: Performance vs. Number of Hot Regions (Normalized for Zero Latency).

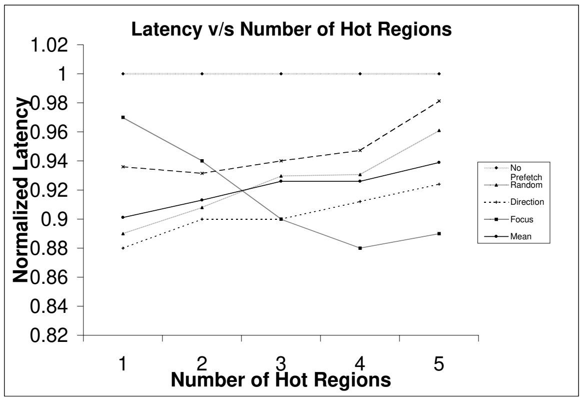


Figure 16: Performance vs. Number of Hot Regions.

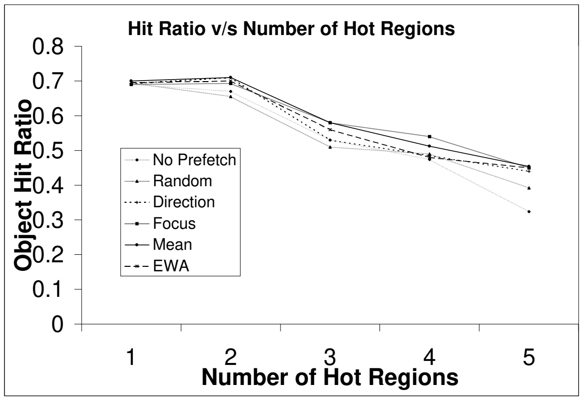


Figure 17: Hit Ratio vs. Number of Hot Regions (Normalized for Zero Latency).

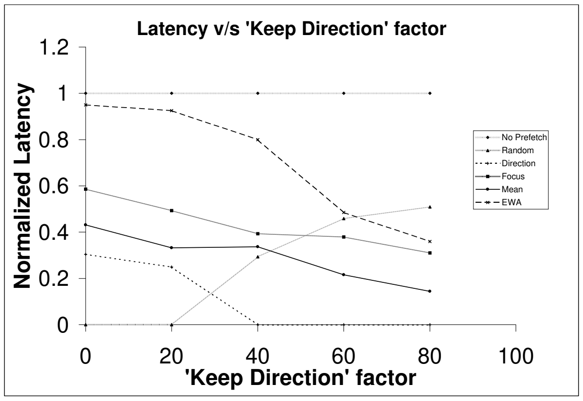


Figure 18: Performance vs. 'Keep Direction' factor.

As shown in Figure 17 as the number of hot regions increases, the object hit in the cache reduces for all the strategies. The reason for this is that the user input tends to become more random in nature as he moves towards the hot regions that are close to the current navigation window. Note that there is less effect on the *focus* prefetching strategy compared to the other strategies, as it places emphasis on prefetching in the direction of the hot regions. Also note that all the prefetching strategies have a higher hit ratio than not applying any prefetching, justifying the usefulness of prefetching for visual applications.

6.3 Experiments with User Directionality

The user simulation files were generated for varying navigational strategies, say from random direction changes to using the same direction most of the time. Figure 18 shows the variation in the performance of all the strategies (on the Y-axis) when we have navigation which is more or less directional (on X-axis). The performances of most of the strategies, such as *direction*, *focus*, *mean* and *ewa*, improve, as most of them are directional in nature. Although the *focus* strategy also improves somewhat in performance, it still does not perform better than the direction strategy as it tries to get data in the direction of the hot regions. Also note that the performance of the *random* strategy gets worse as the user becomes more directional as it does not exploit this newly gained knowledge. We utilize the *direction* strategy as the representative prefetching strategy for the rest of this section as it is an overall good performer compared to the other strategies.

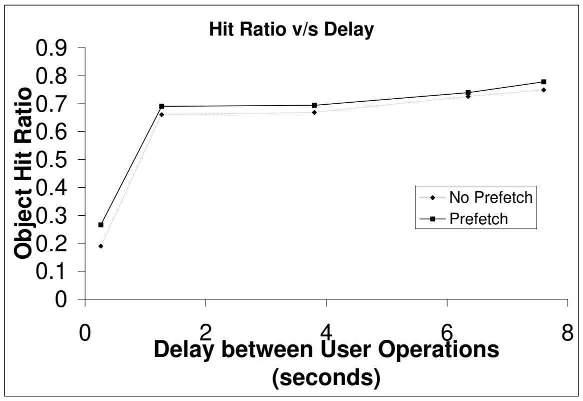


Figure 19: Hit Ratio vs. Delay.

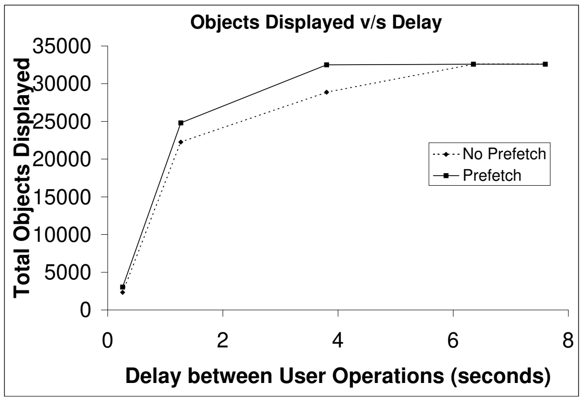


Figure 20: Objects Displayed vs. Delay.

Varying delay parameters. The user simulation files were generated by giving delay factors to the original user input file, and then we measured various output parameters, including the hit ratio, number of objects displayed and latency. The latency did not require normalization as the navigation path in the files remained the same. As Figure 19 points out, the hit ratio increases with the increase in the delay between two consecutive user operations as the prefetcher gets more time to prefetch the data. The same is true for the non-prefetching scheme as the data fetching gets aborted fewer times.

As shown in Figure 20, when the user quickly manipulates the interaction tools to navigate, many of the queries get cancelled as the user requires new data, and the old data being fetched is no longer required. As the delay between the user operations increases, the amount of data displayed increases as the cancellation of the queries reduces. This improves the performance of the prefetcher, as the gaps between the user queries increases and it gets more time to prefetch data.

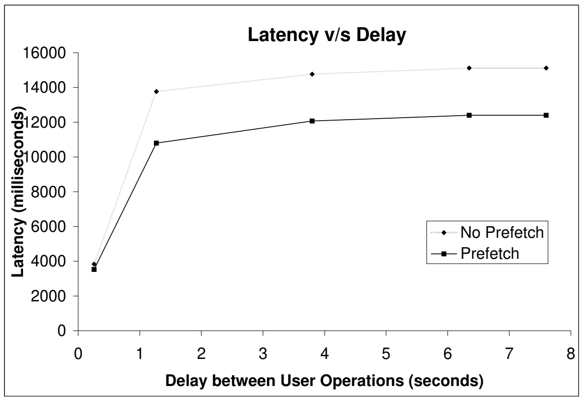


Figure 21: Latency vs. Delay.

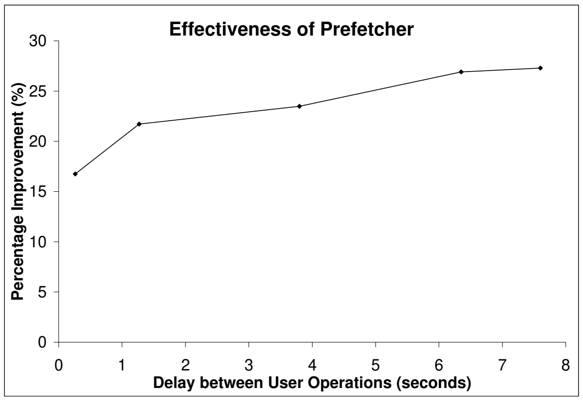


Figure 22: Effectiveness of Prefetcher vs. Delay.

In Figure 21, the X-axis shows the delay between the user operations and the Y-axis depicts the latency. We measure the time taken for the overall fetching task, which includes both the time taken for queries that were fetched from the database and the time when queries were not fetched because of the user speed. When measuring the latency, an issue of how to measure latency comes into the picture. Latency is the online waiting time measured whenever the query is answered by the backend. It does not include the time when the backend was unable to fetch the data from the table due to the cancellation of the query when the user issued a new query prior to the completion of the previous one. As seen in Figure 20, the number of objects actually being displayed initially is low, or in other words, the number of fetch requests completed satisfactorily is low. This implies that the amount of unmeasured latency is high (the amount of the remaining latency that would have been added if the query had been answered).

6.4 Effectiveness of Prefetcher

Figure 22 charts the percentage of improvement in the performance of applying the *focus* prefetching strategy compared to not applying prefetching for simulated user traces on the simulated data for different delays between the user operations. The improvement is measured as the percentage of latency reduced in comparison to when no prefetching is done. It is calculated as:

$$PercentImprove = \frac{(LatencyWithoutPrefetch - LatencyWithPrefetch)}{LatencyWithoutPrefetch} * 100$$

As can be seen, the performance is improved by 28 percent. This improvement is attained when the average speed of the user input is one single input per second. Also, the curve flattens a bit as the prefetching gets completed most of the time and is no longer preempted by user inputs. This indicates that an improvement in the performance as the delay between the user operations increases is no longer feasible beyond a certain point of time.

From the above charts, we find that the *focus* prefetching strategy is a reasonable choice for prefetching the data, as it results in good performance for a variety of user access patterns. But the focus strategy needs some pre-requisite knowledge, i.e., the system needs to analyze user traces over a given dataset to identify potential hot regions. This represents a difficulty for realizing the *focus* strategy in some applications in practice. It can be noted that the two vector strategies, *mean* and *exponential weight average*, are less effective than the *focus* and the *direction* prefetching strategies for our navigation environment. The reason for this lower performance is that they try to fetch the data according to the vector-specific direction, but our navigation tool supports only four directions. These strategies may perform better in applications where the navigation tools are more vector oriented. The *direction* prefetching strategy hence turns out to be second best contender. Given that it requires no prior knowledge and due to its simplicity, especially as its performance is nearly equal to that of *focus* prefetching strategy, we adopt the *direction* solution in our current system.

6.5 Validating our Experiments Using Real User Traces

To backup our experimental results, we also have performed a user-study where we had collected traces from a number of users of our system. These traces consisted of 30 minutes each for 20 different users. These traces were then given as input to the tool and system settings were varied. The values recorded were averaged out for the same settings respectively.

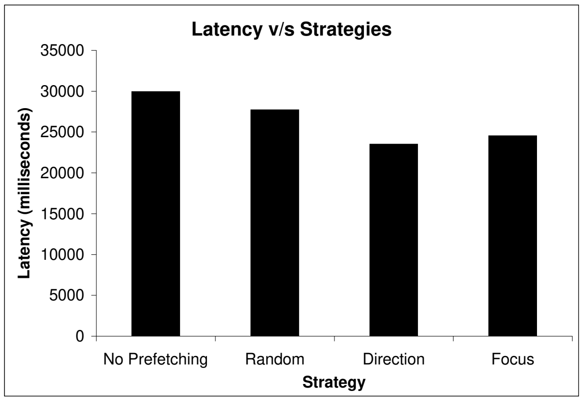


Figure 23: Average Latency vs. Strategies.

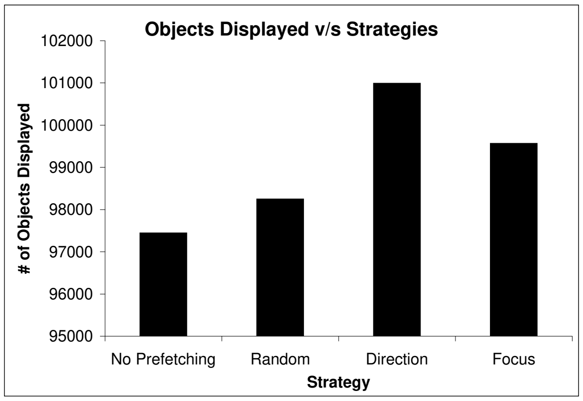


Figure 24: Average Objects Displayed vs. Strategies.

Figure 23 shows the average latency obtained by the actual user traces for different strategies. The hot regions set for the *focus* strategy were calculated based on averaging accesses from different user traces for the same dataset and selecting the most frequently accessed regions. As it can be noted, the *focus* strategy has slightly higher latency than the *direction* strategy. This is so because the hot regions for the data were set simply by averaging access time of the dataset over all the user traces and then selecting regions that had an average access ratio above a particular threshold. Not all the users have similar hot regions, leading to lower performance of the *focus* strategy. We also found that the *focus* strategy improves the system performance if the hot regions are calculated for individual user traces, i.e., the predictors are more accurate. This solution is however not realistic to apply in practice because predictors are not likely to be that accurate.

Figure 24 depicts the average number of objects displayed for different strategies, as concluded from Figure 23. Note that the average number of objects displayed (the visual quality) for the *focus* strategy is less than that of the *direction* strategy.

Figure 25 indicates the average hit ratio for the strategies, based on the previous two charts. The hit ratio for the *focus* strategy is slightly lower than that of the *direction* strategy. Overall, based on the real user traces, we thus favor the *direction* strategy for prefetching.

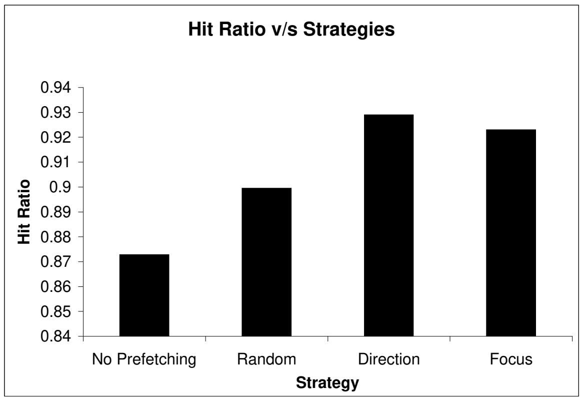


Figure 25: Average Hit Ratio vs. Strategies.

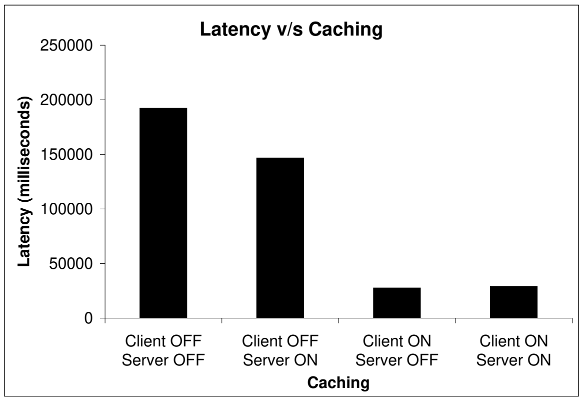


Figure 26: Latency vs. Caching.

6.6 Experiments with Caching

Figure 26 shows the improvement in the performance of the system when the caching is turned ON and OFF on both the server and client side. Since Oracle does not provide too fine support to directly control the cache, we vary the degree in which tuples are cached by providing the Oracle optimizer with cache hints. The cache hints that we use are "cache" and "nocache". A "cache" object will persist in the database buffer as long as possible. A "nocache" object will not be loaded into the buffer. However, nothing prevents the optimizer to read a "nocache" object from the cache, if the object is there. Thus, the results of these experiments have a high degree of deviation. Note that latency reduces by 85 percent just by caching at the client side. This is logical as the hit ratio when client-side caching is turned ON is around 85 percent. Also note that latency increases slightly with server-side caching turned ON as the server caches the old data that the client may already have cached, but not the new data that client may need. However, the caching at the server when the client doesn't have a cache improves the performance a bit more, because the server caches some of the old objects required by the client.

7 Related Work

7.1 Visual Exploration Systems

Much work has been done in recent years on visual interaction tools, including [45, 19, 9, 18, 23, 21, 17]. Integrated visualization-database systems such as Tioga [40], IDEA [37] and DEVise [32] represent the work most close related to ours in terms of developing tools for visual data exploration support. The specific approaches taken are however different. Tioga [40, 2] implements a multiple browser architecture for a *recipe*, a visual query. The system is able to cache the computed data; however, the problem of translating front-end operations into database queries is not present since database queries are explicitly specified by the graphical interface. Also, they do not cache the queries in their system. VIDA [49] is a visualization tool (an extension of DataSplash [2]) that is developed in an attempt to solve the cluttering problem by providing *goal-directed zooming*. Infostill [4] is a data analysis application that focuses on assisting users on all stages of data analysis. It does not take client-side caching into consideration for improving the performance of the tool. IDEA [37] is an integrated set of tools that supports interactive data analysis and exploration. This tool focuses on multiple display views like XmdvTool, but on-line query translation and memory management are not addressed in that work. In DEVise [32], a set of query and visualization primitives to support data analysis is provided. The number of primitives supported is relatively large. However, caching data is done at the database level using default mechanisms only; special memory management techniques as in our work are not studied. Unlike prior work, we aim to focus on the interactions between the two areas: Visualization and Databases.

7.2 Caching

Semantic caching is used for client-side caching and replacement in a client-server database system. It is aimed in large at providing support for navigational access to data (like visualization applications). We have implemented a caching structure inspired by [24, 8] as it provides efficient support for access to data. In particular, we have developed a hash-based look-up structure that allows replacement at the object-granularity level. Though caching is necessary for visualization applications and necessarily a prerequisite for support of prefetching, our research concentrates on the trade-offs between different prefetching strategies.

7.3 Prefetching

Most of the recent work on prefetching can broadly be classified into one of the three classes: web prefetching [44, 46, 27, 33, 29], prefetching for memory caches by operating systems [26, 35, 22], and I/O prefetching [6, 28, 48]. From our literature search, no work has been done to date focussed on prefetching for visualization applications, although there are some similarities with the work done in these fields. Web prefetching typically uses the idle time when the user is thinking what to do. We utilize the same principle. Similarly, the works done in I/O prefetching use the I/O idle time to prefetch the data in the memory. The prefetching techniques described in web prefetching schemes in the literature [44, 46, 27, 33, 29] typically prefetch the most frequently visited page by the user. This is similar to our *focus* strategy, in the sense of associating usage values with the object space instead of focussing on user trace analysis. Hence our idea could be extended to web prefetching as well. *Mean* and *exponential weight average* strategies have been inspired from [3].

8 Conclusions

With the increasing amount of data being accumulated nowadays, the need for visually exploring large datasets becomes more and more important. A viable way to achieve scalability in visualization is to integrate visualization applications with database management systems. A good memory management strategy should be employed in order to reduce the overhead of I/O intensive database accesses and thus make the use of the persistent storage transparent to end-users. This paper presents a solution to this problem by:

1. Semantic caching - Caching queries rather than individual objects or pages of tuples.

2. Prefetching - Use of prefetching to further improve the performance of the caching system.

The approach is being used in coupling XmdvTool, a visualization application for interactive exploration of multivariate data, with an Oracle8i database management system.

Experiments for caching and prefetching have shown that caching at the client-side improves the performance of visual environments to a great extent. To further reduce the response time in the system, we have designed a speculative non-pure prefetcher that brings data into memory when the system is idle. We have introduced a family of five prefetching techniques based on our hypotheses about the characteristics of different user navigation styles. Our evaluation have shown that *direction* strategy outperforms the other strategies in terms of various parameters. Navigation patterns simulated data specificity as well as user specificity. These simulated user sessions were confirmed to be similar to that of actual users by the user study.

Directions for further research include using data mining tools for analysis of user traces. If we can confidently predict navigation patterns, then the task of extracting the exact parameters from a real navigation script becomes a tractable statistical problem. In the future, we plan to also consider dynamic changes in the data as well as rapid changes of the interaction tools.

References

[1] C. Ahlberg and B. Shneiderman. Alphaslider: A compact and rapid selector. *Proceedings of the ACM Conference on Human Factors in Computing Systems (CHI'94), p. 365-371*, 1994.

[2] A. Aiken, J. Chen, M. Lin, and M. Spalding. The Tioga-2 database visualization environment. *Lecture Notes in Computer Science*, 1183:181–??, 1996.

[3] J. Chim, M. Green, R. Lau, H. V. Leong, and A. Si. On caching and prefetching of virtual objects in distributed virtual environments. In *Proceedings of the sixth ACM international conference on Multimedia, 1998*, pages 171–180, Sept. 1998.

[4] K. Cox, S. Hibino, L. Hong, A. Mockus, and G. Wills. Infostill:a task-oriented framework for analyzing data through information visualization. In *IEEE Information Visualization Symposium 1999, Late Breaking Hot Topics*, pages 19–22. ACM Press, Jan. 9–11 1999.

[5] K. M. Curewitz, P. Krishnan, and J. S. Vitter. Practical prefetching via data compression. In *Proc of the 1993 ACM SIGMOD Intl Conf on Management of Data, Washington, D.C., May 26-28, 1993*, pages 257–266. ACM Press, 1993.

[6] K. M. Curewitz, P. Krishnan, and J. S. Vitter. Practical prefetching via data compression. pages 257–266, 1993.

[7] F. D., L. A., S. D., and Y. K. Optimization of run-time management of data intensive web-sites. In *Proceedings of the 25th International Conference on Very Large Data Bases (VLDB '99)*, pages 627–638, San Francisco, Sept. 1999. Morgan Kaufmann.

[8] S. Dar, M. J. Franklin, B. T. Jónsson, D. Srivastava, and M. Tan. Semantic data caching and replacement. In *Proc of 22th Intl Conf on Very Large Data Bases, Sept 3-6, 1996, Mumbai (Bombay), India*, pages 330–341. Morgan Kaufmann, 1996.

[9] M. Derthick, J. Harrison, A. Moore, and S. Roth. Efficient multi-object dynamic query histograms. *Proc. of Information Visualization*, pages 58–64, Oct. 1999.

[10] D. DeWitt, D. Mayer, P. Futtersack, and F. Velez. A study of three alternative workstation-server architectures for object-oriented database systems. In *Proc of VLDB'90 16th Intl Conf on Very Large Data Bases, Brisbane, Australia, 1990.* Morgan Kaufmann, 1990.

[11] Y. Fua, M. Ward, and E. Rundensteiner. Structure-based brushes: A mechanism for navigating hierarchically organized data and information spaces. *IEEE Visualization and Computer Graphics, Vol. 6, No. 2, p. 150-159*, 2000.

[12] Y.-H. Fua, M. O. Ward, and E. A. Rundensteiner. Hierarchical parallel coordinates. Technical Report ??, Worcester Polytechnic Institute, Computer Science Department, 1999.

[13] Y. H. Fua, M. O. Ward, and E. A. Rundensteiner. Hierarchical parallel coordinates for exploration of large datasets. *IEEE Proc. of Visualization*, pages 43–50, Oct. 1999.

[14] Y. H. Fua, M. O. Ward, and E. A. Rundensteiner. Navigating hierarchies with structure-based brushes. *Proc. of Information Visualization*, pages 58–64, Oct. 1999.

[15] P. Godfrey and J. Gryz. Answering queries by semantic caches. In *Proc of Database and Expert Systems Applications, Florence, Italy*, pages 485–498, Sept. 1999.

[16] S. Hibino and E. A. Rundensteiner. MMVIS: A multimedia visual information seeking environment for video analysis. In *Proceedings of ACM CHI 96 Conference on Human Factors in Computing Systems*, volume 2 of *DEMONSTRATIONS: Video: Authoring and Indexing*, pages 15–16, 1996.

[17] S. Hibino and E. A. Rundensteiner. Processing incremental multidimensional range queries in a direct manipulation visual query. In *Proc of the Fourteenth Intl Conf on Data Engineering, Orlando, Florida, USA*, pages 458–465, 1998.

[18] S. Hibino and E. Rundersteiner. User interface evaluation of a direct manipulation temporal visual query language. In *Proc of The Fifth ACM Intl Multimedia Conf (MULTIMEDIA '97)*, pages 99–108, New York/Reading, Nov. 1998. ACM Press/Addison-Wesley.

[19] Y. Ioannidis. Dynamic information visualization. *SIGMOD Record (ACM Special Interest Group on Management of Data)*, 25(4):16–16, Dec. 1996.

[20] C. Jeong and A. Pang. Reconfigurable disc trees for visualizing large hierarchical information space. *Proc. of Information Visualization '98, p. 19-25*, 1998.

[21] N. Jing, Y. Huang, and E. A. Rundensteiner. Hierarchical encoded path views for path query processing: An optimal model and its performance evaluation. *IEEE Transaction on Data and Knowledge Engineering*, 10(3):409–432, May 1998.

[22] D. Joseph and D. Grunwald. Prefetching using Markov predictors. In *Proc of the 24th Annual Intl Symposium on Computer Architecture (ISCA-97)*, Computer Architecture News, pages 252–263, New York, June 1997. ACM Press.

[23] S. Kaushik and E. Rundensteiner. SVIQUEL: A spatial visual query and exploration language. *DEXA*, 1460:290–299, 1998.

[24] A. M. Keller and J. Basu. A predicate-based caching scheme for client-server database architectures. *VLDB Journal*, 5(1):35–47, 1996.

[25] A. Kemper and D. Kossmann. Dual-buffering strategies in object bases. In *Proc of VLDB'94 20th Intl Conf on Very Large Data Bases.* Morgan Kaufmann, 1994.

[26] A. Ki and A. E. Knowles. Adaptive data prefetching using cache information. In *International Conference on Supercomputing*, pages 204–212, 1997.

[27] A. Kraiss and G. Weikum. Integrated document caching and prefetching in storage hierarchies based on Markov-chain predictions. *VLDB Journal: Very Large Data Bases*, 7(3):141–162, 1998.

[28] Krishnan and Vitter. Optimal prediction for prefetching in the worst case. In *SODA: ACM-SIAM Symposium on Discrete Algorithms (A Conference on Theoretical and Experimental Analysis of Discrete Algorithms)*, 1994.

[29] T. M. Kroeger, D. D. E. Long, and J. C. Mogul. Exploring the bounds of Web latency reduction from caching and prefetching. In *Proceedings of the USENIX Symposium on Internet Technologies and Systems (ITS-97)*, pages 13–22, Berkeley, Dec. 8–11 1997. USENIX Association.

[30] Y. Leung and M. Apperley. A review and taxonomy of distortion-oriented presentation techniques. *ACM Transactions on Computer-Human Interaction Vol. 1(2), June 1994, p. 126-160*, 1994.

[31] B. Lipchak and M. Ward. Visualization of cyclic multivariate data. *Proc. of Visualization '97, Late Breaking Hot Topics, p. 61-4*, 1997.

[32] M. Livny, R. Ramakrishnan, K. S. Beyer, G. Chen, D. Donjerkovic, S. Lawande, J. Myllymaki, and R. K. Wenger. DEVise: Integrated querying and visualization of large datasets. In *Proc ACM SIGMOD Intl Conf on Management of Data, May 13-15, 1997, Tucson, Arizona, USA*, pages 301–312. ACM Press, 1997.

[33] E. Markatos and C. Chronaki. A top-10 approach to prefetching on the web. Technical Report TR96-0173, 1996.

[34] J. Y. Matthew O. Ward and E. A. Rundensteiner. Hierarchical exploration of large multivariate data sets. *Proceedings Dagstuhl '00: Scientific Visualization*, May 2001.

[35] R. H. Patterson, G. A. Gibson, E. Ginting, D. Stodolsky, and J. Zelenka. Informed prefetching and caching. In *Proceedings of the Fifteenth ACM Symposium on Operating Systems Principles*, pages 79–95, Copper Mountain, CO, 1995. ACM Press.

[36] G. Robertson, J. Mackinlay, and S. Card. Cone trees: Animated 3d visualization of hierarchical information. *Proc. of Computer-Human Interaction '91, p. 189-194*, 1991.

[37] P. G. Selfridge, D. Srivastava, and L. O. Wilson. Idea: Interactive data exploration and analysis. In *Proc of the 1996 ACM SIGMOD Intl Conf on Management of Data, Montreal, Quebec, Canada, June 4-6, 1996*, pages 24–34. ACM Press, 1996.

[38] B. Shneiderman. *Designing the User Interface: Strategies for Effective Human-Computer Interaction.* Addison-Wesley Publishing, third edition, 1997.

[39] B. Shneiderman. Tree visualization with tree-maps: A 2d space-filling approach. *ACM Transactions on Graphics, Vol. 11(1), p. 92-99*, Jan. 1992.

[40] M. Stonebraker, J. Chen, N. Nathan, C. Paxson, and J. Wu. Tioga: Providing data management support for scientific visualization applications. In *19th Intl Conf on Very Large Data Bases, 1993, Dublin, Ireland*, pages 25–38. Morgan Kaufmann, 1993.

[41] I. D. Stroe, E. A. Rundensteiner, and M. O. Ward. Minmax trees: Efficient relational operation support for hierarchy data exploration. Technical Report TR-99-37, Worcester Polytechnic Institute, Computer Science Department, 1999.

[42] I. D. Stroe, E. A. Rundensteiner, and M. O. Ward. Scalable visual hierarchy exploration. In *Database and Expert Systems Applications, Greenwich, UK*, pages 784–793, Sept. 2000.

[43] P. Sulatycke and K. Ghose. A fast multithreaded out-of-core visualization technique. In *13th Intl Parallel Processing Symposium and 10th Symposium on Parallel and Distributed Processing*, Apr. 1999.

[44] N. Swaminathan and S. Raghavan. Intelligent prefetching in www using client behavior characterization, 2000.

[45] E. Tanin, R. Beigel, and B. Shneiderman. Incremental data structures and algorithms for dynamic query interfaces. *ACM Special Interest Group on Management of Data*, 25(4), Dec. 1996.

[46] A. Venkataramani, P. Yalagandula, R. Kokku, S. Sharif, and M. Dahlin. The potential costs and benefits of long term prefetching for content distribution. *Technical Report TR-01-13, UT, Austin, 2001.*, 2001.

[47] M. Ward. Xmdvtool: Integrating multiple methods for visualizing multivariate data. *Proc. of Visualization '94, p. 326-33*, 1994.

[48] H. Wedekind and G. Zoerntlein. Prefetching in realtime database applications. In *ACM SIGMOD*, 1986.

[49] A. Woodruff and M. Stonebraker. Visual information density adjuster (VIDA). Technical Report CSD-97-968, University of California, Berkeley, Nov. 24, 1997.

[50] Xmdvtool home page. http://davis.wpi.edu/~xmdv.