Scalable Access within the Context of Digital Libraries*

X. Cheng R. Dolin M. Neary S. Prabhakar K. V. Ravi Kanth D. Wu
D. Agrawal A. El Abbadi M. Freeston A. Singh T. Smith J. Su
Alexandria Digital Library Project
Department of Computer Science
University of California
Santa Barbara, CA 93106

Abstract

This paper presents a summary of some of the work-in-progress within the Alexandria Digital Library Project. In particular, we present scalable methods of locating information at different levels within a distributed digital library environment. Starting at the high level, we show how queries can be routed to appropriate information sources. At a given source, efficient query processing is supported by using materialized views and multidimensional index structures. Finally, we propose solutions to the problem of storage and retrieval of large objects on secondary and tertiary storage.

1 Introduction

Library services provide efficient access to information implicit in collections of information-bearing objects such as text documents and maps. Traditional libraries, based on information in non-digital form, are characterized by a limited set of services that provide relatively coarse-grained access to the information in controlled collections of items. Digital libraries, based on information in digital form, are able to generalize these services by providing fine-grained access to information contained in their controlled collections. Some important issues facing digital libraries include decisions involving the set of services to provide and the design and implementation of those services that will scale with the size of the collection and the number of users. For these services to scale appropriately, it is critical to identify potential bottlenecks and to incorporate mechanisms for resolving them.

In this paper, we identify three primary scalability issues related to providing information access within the context of a digital library, and show how the Alexandria Digital Library (ADL) Project [1] addresses them. First, the data itself is distributed on a scale that was not possible earlier. Instead of a single or a handful of databases or information sources, we now have information sources that are scattered throughout the Internet. We analyze various existing and proposed systems and then outline *Pharos*, an approach we are investigating within the context of ADL for handling a large number of information sources. Second, the dimensionality of the data itself is large. Instead of a one dimensional information space with totally ordered items, we are now confronted with objects in large n-dimensional information spaces. Traditional techniques that work well in one dimension do not scale well. For efficiency, we use both materialized views as well as various techniques based on multidimensional index structures. Materialized views may be used to fine-tune performance to the particular data and query patterns of a given source. We approach the problem of multidimensional data in several ways. In the short-term, we parallelize current multidimensional index structures to improve their performance and explore various transformations to reduce the dimensionality of image data. In the long-term, we are developing efficient multidimensional index structures to deal with spatial data. Finally, the size of the individual objects themselves is large. Instead of small records, we are now dealing with gigabyte multimedia objects including books, images and videotapes. Our proposed storage mechanism for Alexandria involves performing wavelet transforms of (some) images and storing them as wavelet coefficients that can be used to construct images at various resolutions. We have explored various data placement techniques for storing wavelet coefficients on multiple I/O devices as well as the storage of large data objects on tertiary storage.

We describe the organization of the paper using Figure 1, which highlights the main components ad-

*This research was partially supported by NSF/NASA-/DARPA under grant number IRI94-11330.

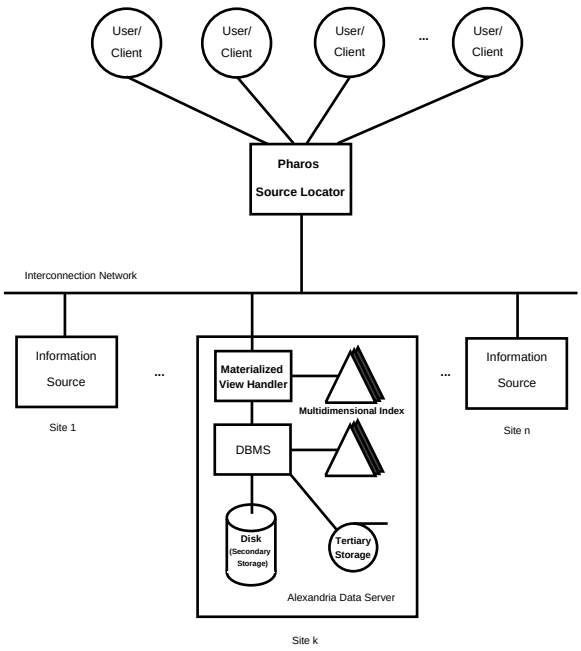

Figure 1: The components of a digital library.

dressed in this paper. The mapping of a user query to appropriate information sources is handled by the *Pharos* component described in Section 2. Section 3 shows how materialized views in the context of digital library metadata can be used to improve query response times dramatically. Multidimensional indexing in the context of spatial data as well as image data is discussed in Section 4. Storage and retrieval of large objects on both secondary and tertiary storage is described in Section 5. We conclude with a summary of our work in Section 6.

2 Locating Sources

Finding appropriate sources of information on the Internet increases in difficulty as the volume of available data grows and the numbers of Internet users and information providers increase. An information source is any site on the network that provides electronic documents such as articles, books, images, files, etc. Prior to digital networks, the number of potential information sources was relatively limited, mainly by locality; a user could discover what was available by browsing individual sources. On the Internet, information resides in numerous heterogeneous systems; each such distributed system may provide many information sources.

2.1 Network Models

In [8], we consider three general classes of network models that encompass the discovery and querying of information sources. The first class of models uses no intermediate servers; the only communication is directly between query generators and information sources. Extreme cases within this class require either that the query generator communicate with every source directly for each query, or else that each query generator locally store a description of every source with enough detail that the best sources can be found without having to acquire additional metadata. The next class places between the query generators and the sources, a single intermediary that handles both query and metadata traffic. A simple model in this class, the Brute-Force (BF) Model, is the system used by most, if not all, current WWW index sites, (e.g. Lycos, AltaVista, etc.). In this model an intermediary extracts metadata by gathering all documents available at all sources. Queries are sent to the intermediary, which returns a result set to the query generator; queries are not propagated to the sources. A more sophisticated approach is the STARTS Model [13], built on a *gGlOSS* [14] framework. Here, each source extracts its own metadata; the intermediary then gathers this metadata rather than extracting its own. When an intermediary receives a query, it analyzes its pre-collected metadata and chooses a set of relevant sources. The query is then propagated to these sources in a standard format and they return the results, also in a standard format, back to the intermediary, which passes the merged results back to the query generator.

The third and final class allows multiple, communicating, intermediaries. The corresponding metadata for this class of models is potentially more complex and carefully distributed. We describe a design called Pharos, shown in Figure 2, which utilizes hierarchical metadata; this metadata is organized in a tree-like structure. Information at a parent node in the tree contains some form of summary of the information of all its children, though possibly with less detail. Metadata is extracted by the source, but here the metadata is composed of a small amount of non-hierarchical information and a relatively large amount of hierarchical information. The metadata is then split into two levels: the *high-level* metadata consists of the non-hierarchical information and the top portion of the hierarchical metadata, and the *mid-level* metadata consists of the lower portion of the hierarchical metadata. There are two types of intermediaries:

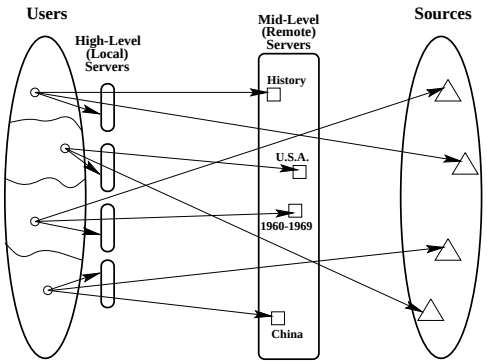

Figure 2: Pharos Model Query

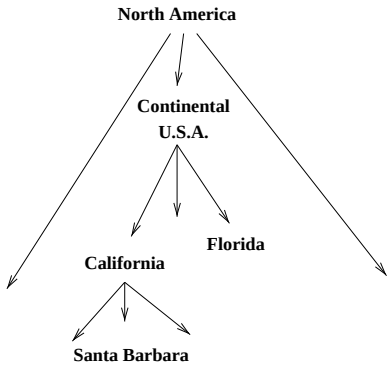

Figure 3: Information Hierarchy

high-level servers and mid-level servers.[1] These levels correspond to the levels in the metadata. A source propagates its metadata as needed to the relevant intermediaries. When a query is generated, there is an iterative process of metadata retrieval. The query is first matched against the local high-level metadata and the sources are filtered to a greatly reduced subset (from, say, $\sim 10^6$ potential sources available on the network to, say, $\sim 10^3$ relevant sources for the specific query). The query generator then requests more metadata about this subset from the relevant topic-based mid-level servers, situated remotely on the network. These servers supply more detailed information about this filtered list of sources, thus further reducing the number of potentially relevant sources (say from $\sim 10^3$ to $\sim 10^1$). These sources are then queried directly. Pharos is presented in more detail below.

The analysis of these models in [8] indicates that Pharos scales well by comparison. The BF Model requires that all intermediate servers pull over the actual documents of all sources, causing problems as the number of sources and documents per source increase. The scalability of Pharos is in part due to the fact that intermediate servers pull over only metadata, not documents, and that this metadata is classification-based, hence the metadata size is independent of a collection's size. Furthermore, Pharos partitions its metadata between many coordinated intermediate servers, and therefore much less duplicate information is distributed compared to the other models.

2.2 Pharos Architecture

The metadata in Pharos is designed to support a multi-level information system and matches the query structure as well as the retrieval system. Therefore, metadata is grouped according to its informational relationship and designed around a hierarchical structure which lends itself to progressively refined queries. Pharos employs information classifications that fit common query methods, such as a subject-based hierarchy, a geographical hierarchy, and a temporal hierarchy. Once these hierarchies have been defined, we include the top portions in the high-level metadata, and lower portions in the mid-level metadata.

An *information hierarchy* is a topic-based classification tree, as shown in Figure 3. In order to describe the collection at a source, we quantify the number of documents with content relevant to each part of an information hierarchy. For example, the subject hierarchy is modeled after the Library of Congress's LC Classification system [17], which has a fixed list of categories. Similarly, a simplistic geographical information hierarchy can be formed by tiling the Earth's surface into progressively smaller, hierarchical, longitude/latitude squares.

An information *taxonomy* is defined as a static hierarchy within which documents can be classified, usually according to their content. A taxonomy is assumed to be a tree structure such that each node in the tree has two attributes: a label and a numerical value. The label denotes a word, term, or phrase from an information hierarchy. The numerical value of each node of a taxonomy is called the *coverage* of the node. It is used to describe a document collection by quantifying the number of documents (or portions thereof) with content relevant to the meaning of the label of the node. Intuitively, a node's coverage indicates what fraction of a document collection is classifiable within a node's label. We allocate as much of a document as possible to nodes closest to the leaves of the tree.

[1] Low-level servers are considered to be the sources themselves.

Therefore, a document is classified within the most specific sub-tree of the hierarchy. The number of documents associated with a node is its coverage times the total number of documents in the collection.

Given a hierarchical metadata structure, we need to partition the information between high- and mid-level components. The main constraint in the case of high-level servers is size. The size of a high-level metadata record is dependent on the number of taxonomies used and the number of nodes that are included from each taxonomy. Although the number of taxonomies a source uses might vary, for a given taxonomy all sources must always include the same nodes in the high-level metadata. High-level metadata refers to 1) any taxonomy-independent source description, and 2) the high-level portions of all taxonomies within which a source's collection has been classified. We estimate that we can include on the order of 20 to 30 nodes from each taxonomy in our high-level metadata records. Mid-level metadata is not subject to the same space constraints as the high-level metadata for several reasons. The structure of mid-level metadata is similar to the taxonomy-dependent components of the high-level metadata, but covers nodes of greater depth in the information taxonomies.

Once the metadata at each site has been compiled, it needs to be distributed over the network according to the intended storage and retrieval architecture. As previously stated, the high-level metadata needs to be widely distributed and replicated, while the mid-level metadata is very selectively distributed. The distribution of high-level metadata is designed to utilize the distribution of USENET news via NNTP by having each source post their high-level metadata as a news article. Each source registers with each mid-level server for which the source has relevant metadata, telling the server how often to update the source's metadata. The mid-level servers in Pharos are responsible for keeping their metadata up-to-date. This approach minimizes network traffic and guarantees that the mid-level servers are ready to receive the metadata before it is sent. Harvest [3] provides a suitable transport mechanism for distributing and storing mid-level metadata in Pharos [7]. We have performed a set of simulation experiments of Pharos which indicate that a sufficient number of good sources could be located with the Pharos architecture so that the majority of relevant documents could be retrieved[7].

3 Materialized Views and Database Performance

In the previous section, we presented an overall framework and specific techniques for locating information sources. In this section, we focus on organizing and searching data collections efficiently within an individual site (source).

Digital libraries contain large collections (e.g., 5–10 million items) of complex data objects and their description (metadata). The metadata are usually more structured than the data objects themselves and are better managed by database management systems which provide good indexing and query optimization techniques. The metadata databases, however, are still very large and have complex structures (schemas). For example, the ADL schema contains over 80 tables each of which has 10 to 20 attributes and some relations have more than 5 million tuples. For such databases, the traditional database design methods, typically based on object modeling techniques or normalizing relations, have limitations. This is because they are mainly concerned with the semantic integrity of the database structure and usually do not consider the performance of query processing at all.

The databases in digital libraries are query intensive and evaluating user queries can be very inefficient even using good query optimization strategies. Consider, for example, a query including a join of two relations. If the relations have a large number of tuples, the join will be time-consuming. One solution to this problem is to precompute some queries, and then store and index the answers. These answers are called *materialized views*. At the time of processing a user query, we rewrite the query to use some of the materialized views so that some or even all of the join operations are eliminated.

There are two main issues in performance tuning using materialized views: 1) to identify a set of views to materialize and 2) to translate user queries against the original database to ones that use the materialized views. We have developed a general approach based on these steps for relational databases [6]. Briefly, we first analyze frequent user queries and identify query patterns in them. We then select materialized views based on the frequencies and complexity of query patterns. At runtime, query patterns in a user query are identified and the query is translated to an equivalent one using a translation table that contains the information about the materialized views. The resulting query involves fewer joins and hence exhibits better performance.

Selection of materialized views requires a careful analysis of user queries, their frequencies and complexity. In our approach, we only consider relational algebra queries. Intuitively, these queries correspond to simple SQL queries of the form "Select ... From

... Where", which frequently occur in database applications. In the following, we describe the design of materialized views.

A *query pattern* is the natural join of relations that are connected through join attributes. In our model query patterns involving more than one relation are candidates for materialized views. To select materialized views from query patterns, we collect user queries through operational analysis, identify their query patterns, and compute the frequency for each query pattern. We select query patterns with highest frequencies as materialized views.

Once we have designed the materialized views, we use them to rewrite user queries at runtime. We describe the query translation problem as follows. Suppose D is the original database schema (a set of relations) and V a set of materialized views. When a user query Q formulated over D is requested, it has to be rewritten to another (equivalent) query Q' utilizing the materialized views in V so that it involves the least number of joins. Note that when we generalize the problem to include D and V as a part of the input [16], the new problem is exactly the *minimum cover* problem which is known to be NP-complete. In our case the query Q is the only input to deal with, which permits an efficient solution to our problem.

During database design and maintenance, conducted off-line, we analyze and collect user query patterns, and select materialized views as introduced before. Then we generate a look-up table T. The look-up table stores the query patterns and the optimal translation rules for the patterns. Specifically the table includes the following information: 1) the query pattern, 2) the relations in the query pattern, and 3) the translation rules using the materialized views.

Consider ADL as an example. Part of the ADL database contains named geographic features and the administrative area information about those features – e.g. the airport in Santa Barbara County. Named geographic features are stored in the relation *Features* with their feature identifiers and spatial footprints. Administrative area information is stored in corresponding relations such as *States*, *Counties*, *Countries*, etc; as a result of the database design process. Each named geographic feature is related to its administrative area through a unique identifier. Several relations are used to store the relationship, e.g. *Features_Counties* associates feature identifiers with county identifiers. The following shows the look-up table for the ADL database (we use F for *Features*, C for *Counties*, FC for *Features_Counties*):

Pattern	*Relations*	*Translation Rules*
p_1	F, C and FC	F, C / FC → County_View
...	...	...

Here $p_1 = \{Features, Counties, Features\_Counties\}$ is a query pattern. *County_View* is a materialized view and is defined as $Features \bowtie Features\_Counties \bowtie Counties$, where $\bowtie$ is a join operation. The translation rule in the table for p_1 means that we can use *County_View* to replace the join of three relations in p_1, hence eliminating the join operations in it.

When a user query arrives, the system first checks if it contains any known query patterns. If a pattern is found, the entry in table T is used to identify the optimal rewriting rule(s) for the query. Otherwise, no rewriting occurs. As an example, suppose the query Q: "*Find all airports in the county Santa Barbara*" is requested. The query is to find all named geographic features about airports whose administrative area is "Santa Barbara". Thus the query involves the selections on the natural join of three relations $\{Features, Counties, Features\_Counties\}$. To rewrite the query using materialized views, the rule of pattern p_1 in the above table is applied and the joins in Q are completely eliminated. Using the indexes on the materialized view, the query is answered efficiently.

We applied the materialized view approach to part of the ADL schema with 10 relations. Some of these relations have about 6 million records. Virtually all attributes are indexed. In our experiments, we defined two materialized views and measured the evaluation time of queries against the original relations and against the materialized views. The results show dramatic improvements in performance. We studied the following queries in our experiments:

Q_1 : Find all airports in Santa Barbara county.
Q_2 : Find all vegetation in the country Chad.
Q_3 : Find all vegetation in the U.S.

Each of the above queries involves joining three relations, two of which are very large. The original database as well as the materialized views are maintained in Sybase. The space for the materialized views is roughly 1.5 times that of the original database. In Sybase, query results may be stored in the buffers which can be used for subsequent queries. We measure the performance of queries under both buffered (executing the same query consecutively) and unbuffered (clearing the buffer before executing the query) situations. Each query is executed ten times, and the averages and standard deviations are shown in Figure 4. These results indicate that the space overhead

	Buffered (ms) (avg±stddev)	Unbuffered (ms) (avg±stddev)
Q_1	771.20 ± 1.99	33671.20 ± 96.51
Q_1 after rewriting	151.80 ± 1.47	4130.00 ± 66.30
Speedup	5.08	8.15
Q_2	99.40 ± 3.23	878.60 ± 122.32
Q_2 after rewriting	2.40 ± 1.20	40.00 ± 0.00
Speedup	41.42	21.97
Q_3	93.00 ± 1.34	662.80 ± 5.96
Q_3 after rewriting	2.40 ± 1.20	51.50 ± 28.32
Speedup	38.75	12.87

Figure 4: Performance of Query Evaluation

for maintaining materialized views as well as the computational overhead for rewriting queries are offset by the significant performance improvements.

4 Indexing Multidimensional Data

In the previous section, we examined different strategies for the potential reduction of query overheads by the selection of appropriate materialized views of the database and query-rewriting techniques. In this section, we explain our strategies for efficiently locating data items that satisfy the rewritten query. The data is multidimensional in nature and falls into two specific categories – spatial data and image data. Spatial data is characterized by low dimensionality. Common queries on such data are "retrieve all data items contained or intersecting a user-specified query box" (range/intersection queries) and "retrieve data items that have specific attributes" (exact match query). Image objects are characterized by high-dimensional feature vectors and queries retrieve images similar to user-specified patterns (similarity queries). In [23], we show that the time complexity of range queries on n d-dimensional objects ($d > 1$) using simple tree structures is $O(n^{1-1/d})$. In contrast, for 1-dimensional data, range queries can be answered in logarithmic time. Given this inherent complexity of multidimensional indexing, we adopt the following two approaches to efficiently support arbitrary queries on multidimensional data. First, we explore the impact of parallelism on query processing in multidimensional index structures. Next, we devise new indexing strategies to improve the performance of queries on low dimensional data. Finally, for high dimensional data we note that the query overheads increase with the dimensionality of the data [9]. Hence we examine several techniques to reduce the dimensionality of such data.

4.1 Parallelism in Indexing

Parallelism can take advantage of the need for searching multiple portions in a multidimensional index for range and intersection queries. To illustrate this situation, consider the following example. Figure 5 shows a set of 2-dimensional data objects and an R^* tree [2] indexing these data objects. The dashed box in the figure represents an arbitrary range query. Note that no data objects are contained in the query box and hence the query result is empty. However, the query is processed in the R^* tree from the root to the leaves, searching in all nodes that intersect the query box. Consequently, all the nodes of the tree are searched despite an empty query result.

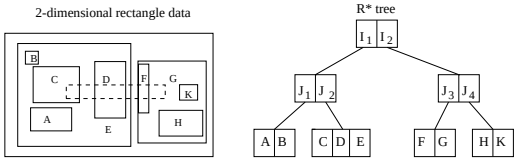

Figure 5: 2-dimensional data; R*-tree for the data

In this context, let us illustrate how parallelism can help. Using the techniques proposed in [22], we partition the R^* tree into two components, each assigned to a separate processor. These techniques estimate the *proximity* (likelihood of being retrieved in the same query) of every pair of nodes and assign nodes with high proximity to different processors [15]. This ensures that most queries can be processed in parallel. Figure 6 shows such a partitioned R^* tree.

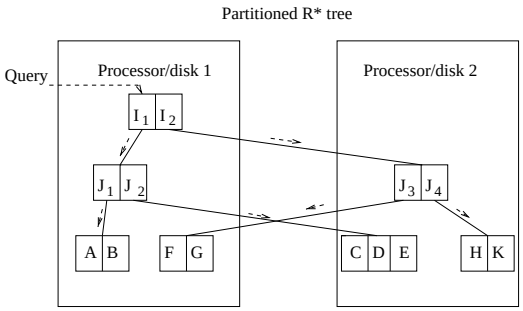

Figure 6: Parallel processing of queries in an R*-tree

Using these ideas, we parallelized two index structures – the R^* tree [2] and the LIB structure [21]. We chose the R^* tree because it is a versatile multidimensional index that caters to arbitrary data. We selected the LIB structure because it maintains multiple index components that can be searched in parallel. These components are formed by first decomposing the data

into multiple levels based on their spatial nesting and then creating a separate index for each level. We evaluated the performance of the parallelized structures on the Gazetteer and Catalog data of the Alexandria Digital Library. In our experiments, both structures registered a speed-up of 3 on 5 processors. Besides, our parallelizing techniques can also be applied to other index structures like BANG files [10] and BV-trees [11] to improve their query performance.

4.2 Indexing Low Dimensionality Data

In this subsection, we examine different ways of structuring multidimensional data to support efficient location of data using different types of queries. As part of this study, we are developing a new, generalized implementation of BANG indexing [10]. BANG indexing was originally developed as a first step in a research effort to overcome the problem of indexing rules in large-scale deductive database systems. Over the years, it evolved into an efficient multidimensional index [11] with broad applicability to different types of data. In its current form, BANG indexing combines the indexing of points (n-tuples) and spatial objects (n-dimensional bounding boxes) through a dual recursive partitioning of an n-dimensional space. In the original implementation [10], which was restricted to point indexing, this partitioning was represented by a balanced, multiply-branched index tree structure similar to a B-tree. Such an index organization exhibited an important feature of multidimensional indexing: it is not necessary to represent the whole data space in the index, if the data is not uniformly distributed. Unless the query distribution coincides with the data distribution, the average path length traversed to answer an exact match query will be less than the height of the index tree. This advantage is reinforced by the extremely compact index - typically 1%-2% of the size of the data set for 1K index and data pages - due to the (on average) very short index entries. This leads to high fan-out ratios and consequent faster access.

In all practical cases encountered, the original index design displayed the characteristics of a B-tree for single access and update. However, such performance could not be *guaranteed*, and pathological cases could be devised which would give much worse and uncontrolled performance. The new version preserves all the principles of BANG indexing, but is implemented as a BV-tree [11]. The properties of this structure make it possible to guarantee logarithmic single-entry access and update, and a minimum 33% index and data node occupancy, for both point and spatial object indexing. We believe that this is the first time that such a combination of properties has been achieved in a dynamic indexing method. (The *average* index and data node occupancy is approximately 69%, as in a B-tree). With this development, we are carrying out a new comparative performance study, specifically against R^* trees [2] and Oracle's SDO.

In addition, we are incorporating a new technique to improve the performance of conventional index structures for different types of queries on spatial data. Spatial data usually consists of rectangles and points in 2 or 3 dimensions and is non-uniform, i.e., data objects vary in their sizes (areas). Conventional index structures are not well suited to index them for two reasons. First, index entries for non-uniform data are more likely to overlap than for uniform data. This effect is more pronounced for nested data, where one rectangle encloses several others. Such overlapping index entries may lead to multiple search paths during queries. Second, clustering of non-uniform data in an index is likely to include more dead space (part of the dataspace that is not covered by any data object) than clustering of uniform data. Indexing such dead space implies accessing irrelevant parts of the tree that may not contribute to the result of the query in any way. To reduce the effect of these two anomalies in current index structures, we proposed a promotion strategy in [21]. This is explained using the following example. Consider the non-uniform data of Figure 7. The data object A contains data object B and overlaps with data objects C, D and E as shown in Figure 7(a). Assuming a node capacity of 3, an R^* tree organizes the data as shown in Figure 7(b). Note that overlapping index entries (e.g. entries g and h corresponding to dashed boxes in the figure) and indexing dead space (e.g. the dotted region k) are inevitable in most index structures.[2] This is a consequence of maintaining all the data objects at the same level (leaf level) in the index hierarchy. Based on this observation, the promotion strategy reorganizes the same set of data objects as in Figure 7(c). Here object A, due to its large extent, is stored at the root level and all others are stored at the leaf level (with i and h as the corresponding index entries). This index organization solves both the problems of non-uniformity – (1) it reduces the overlaps in index entries, and (2) it reduces the dead space that is indexed (region k is excluded).

Based on this simple idea, we devised different criteria for data object promotion [21]. In our experiments with the Gazetteer and Catalog data of the Alexandria Digital Library, these strategies yielded a performance enhancement of upto 45% for the R^*-tree.

[2] Structures such as R^+-trees take explicit actions to remove such overlaps.

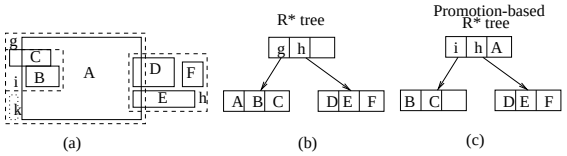

Figure 7: (a): Example of Non-uniform data; (b): R^* tree; (c): Promotion-based R^* tree

4.3 Reducing Image Data Dimensionality

In the previous subsections, we addressed how data items can be efficiently located using their geographical extents. However, some data items, for example images, are more likely to be located by their content. In the content-based retrieval approach, features are extracted from each image and the characteristics of each image are encoded as an n-dimensional feature vector. The features are chosen so that proximity in the n-dimensional space reflects similarity of images. To effectively capture this similarity, a large value of n is usually selected. However this leads to a problem known as the "dimensionality curse": the larger the dimension n, the more difficult to index, and retrieve these feature vectors. For a small number of images, a sequential browsing of images using their feature vectors may suffice. However, for large image databases such as in ADL, the problem of efficient browsing and retrieval of images has to be addressed. Multidimensional search structures like the R^* tree [2] do not scale with the dimensionality of data [9] and hence are not directly helpful. Our approach to solving this problem [25] is to first reduce the dimensionality of the image data and then use multidimensional index structures to support efficient retrieval.

To reduce the dimensionality of the feature vector data, we consider two different techniques based on the Discrete Fourier Transform (DFT) and the Singular Value Decomposition (SVD) in [25]. These techniques have been used to reduce the dimensionality of information in several other projects [9]. Both these methods have their relative merits and demerits. SVD is computationally expensive, requiring $O(mn^2)$ time to process m n-dimensional feature vectors. By using Fast Fourier Transform techniques, the DFT can be calculated in $O(mn \log n)$ time. Moreover, adding new vectors into the database requires the SVD to be recalculated, whereas the DFT needs to be computed only for newly inserted vectors, saving much recalculation effort.

To compare whether truncated SVD components or DFT coefficients provide better retrieval performance, we simulate a series of queries on a collection of textures from the Brodatz album [4]. This collection has 1856 feature vectors, each representing an image. Every vector is processed at the time of insertion into the database with SVD or DFT. Given a set of m n-dimensional vectors, SVD orients the dimensions such that they are arranged in decreasing order of the variance of the m vectors. Hence after SVD, the m vectors are spread out most widely along the first dimension, then along the second and so on. Likewise, we compute the DFT of each n-dimensional feature vector transforming it into a sequence of n complex values.

After transformation, we reduce the feature vectors from n-dimensional space to k-dimensional space. In SVD processing, truncation is performed by picking the first few components (which SVD ensures to be the largest singular values). For the DFT, we can truncate by retaining the first few Fourier coefficients as shown in [9]. This approach works well when the data is very smooth and dominated by low-order coefficients, however as our experiments on texture image data indicates, better results can be obtained by taking the first few *dominant* Fourier coefficients [25].

In our experiments, we measured the quality of the two transformations by evaluating the *recall* and *precision* [24] for different types of queries on the reduced-dimension data. Suppose a user issues a similarity query and retrieves m images from the database. The number of relevant images stored in the database is n; of these only k are returned from the query. *Recall* is defined as k/n, the proportion of the number of relevant results retrieved to the number of relevant results stored in the database. *Precision* is defined as k/m, the proportion of the number of relevant results retrieved to the number of images retrieved.

In a range query, the user specifies a reference image I and a radius R. Using the feature vector for I, the range query retrieves all other images lying in a neighborhood of radius R about I. Alternatively, a nearest-neighbor query retrieves the M images closest to the reference image I. As explained in [25], range queries provide 100% recall, while in nearest-neighbor queries recall and precision are the same. In our experiments, we performed range and nearest-neighbor queries on each feature vector in the database, for varying values for radius R and number of nearest-neighbors M on the truncated feature space. We then evaluated the average precision and average recall of each similarity query at different values of R and M (Tables 1–2). We have found that, in general, SVD provides superior precision and recall. However, for 10–20 nearest-neighbors, we note that the performance of DFT is very close to that of SVD.

Radius	Ave No. Relevant	Ave Retrieved		Ave Precision	
		DFT	SVD	DFT	SVD
1.0	1.45	5.72	5.66	0.2531	0.2559
1.5	4.83	13.99	12.72	0.3455	0.3798
2.0	10.22	27.63	23.59	0.3700	0.4332
2.5	18.70	49.75	40.73	0.3758	0.4591
3.0	31.56	84.24	67.08	0.3746	0.4704

Table 1: Range Queries using 7 components

Num Neighbors	Ave No. Relevant		Ave Recall	
	DFT	SVD	DFT	SVD
1	0.32	0.50	0.5124	0.5038
5	2.62	3.256	0.7039	0.7112
10	6.16	7.93	0.7767	0.7931
15	9.85	12.56	0.8066	0.8376
20	12.22	15.95	0.7590	0.7973
30	17.19	23.257	0.7379	0.7858

Table 2: Nearest-Neighbor with 7 components

This range of 10–20 nearest-neighbors is significant. In querying an image database, we seek to retrieve only those images that are closest to the specification. Due to the difficulty of formulating an exact query, we often must resort to browsing a database, to gain familiarity with its contents as well as to explore related material. When browsing, though, we are limited by the number of images that can be displayed and viewed on screen. In browsing a database we want to retrieve a limited number of the most accurate images. It is in this setting that DFT compares favorably with SVD.

Thus, by choosing the *dominant* Fourier coefficients in the DFT-processed vectors as the components in the reduced dimensional space, we obtain nearly the precision and recall performance of SVD while still retaining the computational advantages of DFT. Once the high dimensional image data is reduced to low dimensional data, they can be indexed using conventional multidimensional index structures to support the efficient location of relevant images that are similar to user-specified query patterns.

5 Secondary and Tertiary Storage

Once the requested items are identified through the processes described in the earlier sections, they need to be retrieved from the database. In this section we discuss the problem of efficient retrieval of large data items. As seen in Subsection 4.3, content based retrieval is a useful tool for browsing image databases such as within ADL. In Subsection 5.1 we describe some of the disk placement techniques that we have developed to improve the browsing performance. The volume of data stored in a library cannot economically be stored on magnetic disks. For such data volume, tertiary storage is the only feasible storage medium. The slow speed and high latency of tertiary storage become bottlenecks and limit the performance of the library. In Subsection 5.2 we discuss the problem of tertiary storage in the context of digital libraries and present some possible solutions.

5.1 Placement of Multidimensional Data

The library should not only provide access to the holdings but also provide the means for users to browse the contents of the library. Browsing is useful in locating items of interest in situations where users are not sure which items match their interests and would like to navigate through a set of items to determine those that are of interest. Since most items viewed during browsing may not be of interest to the user it is not necessary to transmit the complete item, which would consume large amounts of system resources. For browsing purposes, the user is not shown the complete item, rather a summary is presented to enable the user to evaluate the suitability of the item. When items of interest are identified, the user may wish to see these items in greater detail. It is therefore important to provide the ability to browse items and view them at multiple levels of detail or resolution.

For the purposes of ADL, which is designed to handle mostly image data, the technique of wavelet decomposition [5] has been proposed for generating multiresolution images. Using this technique, an image is decomposed into a low resolution copy or icon and several coefficients. The icon is the lowest resolution copy available. Higher resolution copies can be progressively reconstructed from the icon and other coefficients. The original image can be reconstructed from the icon and all the coefficients. In order to improve the I/O performance of browsing, we have investigated efficient placement techniques on magnetic disks [20]. The latency of disk access is reduced by fetching data in parallel from multiple disks. The study focuses on browsing access patterns when images are retrieved by similarity matching, where similarity is based on the content of the images. Two orthogonal declustering schemes have been evaluated. The first scheme declusters the image icons based on their similarity. Since similar images are likely to be accessed together, it would be beneficial to place them on different disks and access them in parallel. The second scheme declusters the coefficients of each image across disks since the coefficients will be accessed together when the image is reconstructed to obtain a higher resolution copy.

The various strategies were extensively tested using simulation as well as experiments with real disks. It was found that declustering the thumbnails based upon image content was always beneficial. The gains of coefficient declustering were, however, not as con-

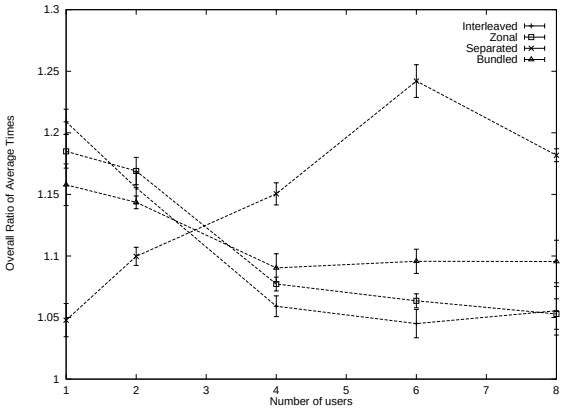

Figure 8: Comparison of declustering strategies

sistent. In Figure 8 we present some representative results of tests conducted with real disks. The x-axis gives the number of concurrent users and the y-axis gives the ratio of the average access time for a random policy to the average access time for the other placement strategies. A higher ratio indicates better performance of the strategy as compared to the random strategy. All strategies shown use the same declustering strategy for the thumbnails, however they differ in the declustering strategy for the coefficients of each image. The *Interleaved* and *Zonal* strategies decluster image coefficients across all disks whereas the *Separated* strategy places the non-thumbnail coefficients together on a designated disk and the *Bundled* strategy placed the thumbnail and all other coefficients on the same disk. The experiments were repeated several times and 95% confidence intervals were generated based upon the results, as shown in Figure 8. From the figure it is clear that when only 1 or 2 concurrent users are present, the strategies that decluster the coefficients perform better than those that do not decluster. As the number of concurrent users increases to 3 or more, both policies that decluster the coefficients, Interleaved and Zonal, are outperformed by the policies that do not decluster coefficients. In general the study concludes that although it is always beneficial to decluster similar images, declustering the coefficients of a single image may or may not be advantageous. In particular, coefficient declustering is good in a single user system, but as the number of concurrent users increases, coefficient declustering becomes harmful. This work can be extended for browsing data that is resident on tertiary storage and other non-image data types where access is based on more general relationships between items.

5.2 Managing Tertiary Storage

Libraries need to store and make available very large amounts of data. The cost and volume of magnetic disks makes them prohibitive for such purposes. Currently and in the foreseeable future, tertiary storage is the only economically viable solution to storing large amounts of data. This requires a revision of the role of tertiary storage, magnetic tapes in particular, which have traditionally been used for backup purposes. When data is stored on tertiary devices, efficient I/O becomes a major problem due to the high latency of these devices. The high latency comes from two factors. Firstly, tertiary storage is more economical than secondary storage because a small number of expensive drives are shared among large numbers of removable media. This implies that in order to access data, it is first necessary to load the medium holding the data onto the drive as opposed to secondary storage where a small number of platters are always "loaded" on the disk drive. A typical load time for tape drives or optical jukeboxes is a few tens of seconds, whereas the typical access time for magnetic disks is a few hundreds of milliseconds. The second factor contributing to the slow speed of tertiary devices is the sequential access nature of magnetic tape, which is currently the dominant tertiary storage technology. It can take up to a few minutes to seek to the desired location on a tape. Further details about tertiary storage technology and recent work on current issues in tertiary storage can be found in [19].

In contrast to other studies, we have explored the problem of scheduling requests for more than one medium [18]. Due to the great difference in speeds between the processors and tertiary storage, we expect that request traffic will be bursty. Consequently, we have studied the problem of scheduling bursty I/O requests. Unlike other solutions which are applicable only to disk technology or only to tape technology, our results are more general and are applicable to any removable medium technology. The study investigates only those schedules that service all requests for the same medium consecutively once the medium is loaded. These schedules are optimal if the switch time is large compared to the average seek time. This is true for optical and magneto-optical disks. For tapes however, the seek time is usually large and can be greater than the switch time. Therefore for tapes, the results may not be optimal but from the experimental results we observe that even for tapes these scheduling policies achieve almost an order of magnitude improvement over first-come-first-serve scheduling policies. The criterion used to compare different schedules

is the average waiting time, W, for each request:

$$W = \sum_{i=1}^{m} \left(C_i + T \cdot i \cdot n_i + n_i \sum_{j=1}^{i-1} P_j \right) \quad (1)$$

For the single drive case, Equation 1 gives the total waiting time incurred by all requests for a given schedule, where i represents the order in which the media are loaded onto the drive. We see that the total time can be viewed as a sum of the wait times for items on each medium (m is the number of media for which requests are made). In Equation 1, C_i is the wait time incurred by all requests on the ith medium after the medium is loaded onto the drive, $T \cdot i \cdot n_i$ represents the wait time incurred by all requests on the ith medium (n_i is the number of requests for the ith medium) due to the loading and ejecting of the earlier media (T is the time required to load and eject a medium) and $n_i \sum_{j=1}^{i-1} P_j$ represents the wait time for all requests on the ith medium due to the processing of the requests on the earlier media (P_j is the time required to service all the requests on the jth medium and to rewind the medium if it is a tape). By observing the common factor n_i in the second and third terms and also that the third term is the sum of $i-1$ addends and the second term has a factor of Ti, we can rewrite the total waiting time as in Equation 2. The term C'_i is constant independent of the schedule and therefore minimizing W is the same as minimizing the expression $\sum_{i=1}^{m} n_i \left(\sum_{j=1}^{i-1} (P_j + T) \right)$ which is very similar to the expression for the minimization of the average waiting time for weighted tasks on a single processor.

$$\begin{aligned} W &= \sum_{i=1}^{m} \left(C_i + T \cdot n_i + n_i \sum_{j=1}^{i-1} (P_j + T) \right) \\ &= \sum_{i=1}^{m} \left(C'_i + n_i \sum_{j=1}^{i-1} (P_j + T) \right) \end{aligned} \quad (2)$$

Based on the above observations, we have developed an optimal scheduling policy for single drives. The policy, which we call OPT, schedules items in the order of non-increasing ratios of number of requests to the sum of the switch time and the processing time, i.e. in non-increasing order of the ratio $\frac{n_i}{T+P_i}$ for each medium. Note that we do not restrict the order in which the requests on any given medium are processed – we only need to know the time required to process all the items, P_j. Therefore, any efficient scheduling policy for a single medium may be used. For multiple drives, the problem of finding an optimal solution is shown to be NP-Complete by a simple reduction from the *Minimum Weighted Completion Time* [12]. In the absence of an efficient optimal solution, we have shown that the policy that achieves optimal solutions for the single drive case, OPT, is an effective heuristic for the multiple drive case also. From experimental results we show that OPT produces schedules that are within 1% of the optimal in all tests. It should be pointed out that even though the schedules are simple, they do require knowledge of the processing time for each medium. In the absence of such information we propose a heuristic which schedules media based solely on the number of requests for each medium. We evaluated this simpler heuristic and found that it produces schedules that are well within 1% of the optimal results.

6 Concluding Remarks

This paper has addressed the design of digital libraries from the viewpoints of a) location of possible information sources for a user query, b) query optimization once relevant sources have been identified, c) indexing of multidimensional data for supporting efficient searches, and d) efficient retrieval of answers from secondary and tertiary storage. The design of a digital library requires the solution of a number of scalability problems, and a final design needs to incorporate all these solutions. In this paper, we isolated each of these problems and described preliminary solutions. The embodiment of these solutions into a comprehensive test-bed is our future goal.

References

[1] D. Andresen, L. Carver, R. Dolin, C. Fischer, J. Frew, M. Goodchild, O. Ibarra, R. B. Kemp, R. Kothuri, M. Larsgaard, B. S. Manjunath, D. Nebert, J. Simpson, T. R. Smith, A. Wells, T. Yang, and Q. Zheng. The WWW Prototype of the Alexandria Digital Library. In *Proceedings of the International Symposium on Digital Libraries*, Tsukuba, Japan, 1995. Also appeared in IEEE Computer, 29(5): 54-60, May 1996.

[2] N. Beckmann, H. Kriegel, R. Schneider, and B. Seeger. The R* tree: An efficient and robust access method for points and rectangles. *Proc. of the ACM SIGMOD Intl. Conf. on Management of Data*, pages 322–331, May 23-25 1990.

[3] C.M. Bowman, P.B. Danzig, D.R. Hardy, U. Manber, et al. The harvest information discovery and access system. In *Proceedings of the Second International World-Wide Web Conference*,

pages 763–771, Chicago, Illinois, October 1994. http://harvest.cs.colorado.edu/.

[4] P. Brodatz. *Textures: A Photographic Album for Artists & Designers*. Dover, New York, New York, 1966.

[5] K. R. Castleman. *Digital Image Processing*. Prentice-Hall, Inc., 1996.

[6] X. Cheng and J. Su. Materialized views as a technique to improve database performance. Manuscript, 1996.

[7] R. Dolin, D. Agrawal, L. Dillon, and A. El Abbadi. Pharos: A scalable distributed architecture for locating heterogeneous information sources. Technical Report TRCS96-05, Computer Science Department, University of California, Santa Barbara, 1996. http://www.cs.ucsb.edu-/TRs/TRCS96-05.html.

[8] R. Dolin, D. Agrawal, and A. El Abbadi. Classifying network architectures for locating information sources. In *DASFAA*, Melbourne, Australia, April 1997.

[9] C. Faloutsos, M. Ranganathan, and Y. Manolopoulos. Fast subsequence matching in time-series databases. In *Proc. ACM SIGMOD Int. Conf. on Management of Data*, pages 419–429, Minneapolis, May 1994.

[10] M. W. Freeston. The bang file: a new kind of grid file. *Proc. of the ACM SIGMOD Intl. Conf. on Management of Data*, pages 260–269, May 1987.

[11] M. W. Freeston. A general solution of the n-dimensional b-tree problem. *Proc. of the ACM SIGMOD Intl. Conf. on Management of Data*, May 1995.

[12] M. R. Garey and D. S. Johnson. *Computers and Intractability - A guide to the Theory of NP-Completeness*. W. H. Freeman and Co., New York, 1979.

[13] L. Gravano, K. Chang, H. García-Molina, and A. Paepcke. STARTS: Stanford Protocol Proposal for Internet Retrieval and Search. http://www-db.stanford.edu/~gravano-/starts.html, 1996.

[14] L. Gravano and H. García-Molina. Generalizing *GlOSS* to vector-space databases and broker hierarchies. In *Proceedings of the 21st VLDB Conference*, Zurich, Switzerland, 1995.

[15] I. Kamel and C. Faloutsos. Parallel R-trees. *Proc. of the ACM SIGMOD Intl. Conf. on Management of Data*, pages 195–204, 1992.

[16] Alon Y. Levy, Alberto O. Mendelzon, Yehoshua Sagiv, and Divesh Srivastava. Answering queries using views. In *Proceedings of the Fourteenth ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems*, pages 95–104, 1995.

[17] Library of Congress, Washington, D.C. *LC Classification Outline*, fifth edition, 1986.

[18] S. Prabhakar, D. Agrawal, A. El Abbadi, and A. Singh. Efficient I/O scheduling in tertiary libraries. Technical Report TRCS96-26, Dept. of Computer Science, Univ. of Calilfornia, Santa Barbara, 1996.

[19] S. Prabhakar, D. Agrawal, A. El Abbadi, and A. Singh. Tertiary storage: Current status and future trends. Technical Report TRCS96-21, Dept. of Computer Science, Univ. of Calilfornia, Santa Barbara, 1996.

[20] S. Prabhakar, D. Agrawal, A. El Abbadi, A. Singh, and T. R. Smith. Browsing and placement of multiresolution images on secondary storage. Technical Report TRCS96-22, Dept. of Computer Science, Univ. of California, Santa Barbara, 1996.

[21] K. V. Ravi Kanth, D. Agrawal, A. El Abbadi, and A. Singh. Indexing non-uniform spatial data. *Manuscript*, 1996.

[22] K. V. Ravi Kanth, D. Agrawal, A. El Abbadi, A. Singh, and T. R. Smith. Parallelizing multidimensional index structures. *IEEE Symposium on Parallel and Distributed Processing*, October 1996.

[23] K. V. Ravi Kanth and A. Singh. Complexity of multidimensional range queries. *Manuscript*, 1996.

[24] G. Salton. *Automatic Text Processing*. Addison-Wesley Publishing Company, Inc., Reading, MA, 1989.

[25] D. Wu, D. Agrawal, , A. El Abbadi, A. Singh, and T. R. Smith. Efficient retrieval for browsing large image databases. *Proc. Conf. on Information and Knowledge Management*, pages 11–18, November 1996.