THESIS REPORT

Master's Degree

Hierarchical Storage Management for Relational Databases

by David Isaac
Advisor: N. Roussopoulos

M.S. 93-19

The Institute for Systems Research is supported by the National Science Foundation Engineering Research Center Program (NSFD CD 8803012), the University of Maryland, Harvard University, and Industry

Abstract

Title of thesis: Hierarchical Storage Management for Relational Databases

Name of degree candidate: David Isaac

Degree and year: Master of Science, 1992

Thesis directed by: Ali Ghovanlou

Hierarchical storage management has the potential to optimize the cost and performance of data storage and access by automatically managing the placement of data on different storage media. Unfortunately, this potential is not realized for data stored by database management systems (DBMSs), because current DBMSs require all database tables to be stored on disk. This paper describes a prototype that integrates a database management system with a storage management system, allowing simple, cost effective access to large volumes of data stored in a tape archive.

Hierarchical Storage Management for Relational Databases

by

David Isaac

Thesis submitted to the Faculty of the Graduate School
of The University of Maryland in partial fulfillment
of the requirements for the degree of
Master of Science
1992

Advisory Committee:

Associate Professor Nick Roussopoulos, Advisor
Associate Professor Alan Hevner
Assistant Professor Mark Austin

Table of Contents

Section **Page**

1. Operational Needs Assessment 1
   - 1.1. Background 1
   - 1.2. Problem 3
   - 1.3. Objective 4
2. Concept Exploration 6
   - 2.1. Hierarchical Storage Management 6
3. Requirements 9
   - 3.1. Functional Requirements 9
   - 3.2. Performance and Capacity Requirements 10
   - 3.3. Physical Requirements 11
   - 3.4. Cost Requirements 11
4. Process Considerations 13
   - 4.1. Project Management and Scheduling 13
   - 4.2. Engineering Specialty Integration 13
   - 4.3. Acquisition Management 14
   - 4.4. Requirements Allocation 14
   - 4.5. Requirements Traceability 14
5. Architecture Trade-Offs and Analysis 15
   - 5.1. Schedule 17
   - 5.2. Cost 18
   - 5.3. Performance 18
   - 5.4. Portability 18
   - 5.5. Transparency 18
   - 5.6. Design Trade-Offs 19
6. Prototype Design 20
   - 6.1. DBMS Storage Manager 20
   - 6.2. File Storage Manager 22
7. Implementation Trade-Offs 24
   - 7.1. Cost 27
   - 7.2. Capacity 27

7.3. Media Cost 27
7.4. Media/Drive Ratio 27
7.5. Weight 27
7.6. Port 28
7.7. SCSI Interface 28
7.8. Standard Drives 28
7.9. Company Revenues 28
7.10. Months in Production 28
7.11 Results 29

8. Prototype Implementation 30
8.1. Physical Configuration 30
8.2. Operational Flow 32
8.3. Performance Optimization 33

9. Future Work 37
9.1. Database Row Caching 37
9.2. Commercial DBMS Storage Manager 37
9.3. Other DBMS Extensions 37
9.4. DBMS Name Server 38
9.5. Object-Oriented DBMS Storage Manager 38

10. Conclusions 39

11. References 40

List of Figures

Number **Page**

1. System context 2
2. Traditional data system architecture 3
3. Goal data system architecture 5
4. Cost versus access time for various media 6
5. Storage media hierarchy 7
6. Alternative architectures 15
7. Alternative architecture ranking 17
8. Prototype structure 20
9. DBMS storage manager structure 21
10. File storage management system 23
11. Objective ranking matrix for alternative tape libraries. 26
12. Prototype physical configuration 31
13. Prototype response to a data request 32
14. DBMS insert bypass for special cases 34
15. Query performance with and without optimization 36

1. Operational Needs Assessment

1.1. Background

The prototype described in this paper is oriented toward science data systems. These systems have certain characteristics that strongly influenced the design of the prototype, including the following:

- They contain large volumes of complex data, such as multispectral satellite imagery, making the cost of data storage a primary concern. Although the prototype can be scaled down to small systems, it is probably not cost effective to do so for databases much less than 10 GB in size.[1]

- Data are used primarily for analysis, and thus updates are infrequent compared to searches. Designing for search rather than update greatly simplifies the prototype.

- The primary goal of the end users is research, and thus the information sought from the system is constantly changing. Systems with more predictable (e.g., monthly batch) usage patterns probably cannot justify the use of a DBMS for accessing data.

- The end users are primarily scientists, often with little or no background in computer science. The prototype sacrifices a certain amount of efficiency to avoid requiring users to write custom programs to low-level data interfaces.

[1]The minimum cost-effective size is approximately the disk capacity that could be purchased for a given storage management system cost.

An example of this type of system is the NASA Earth Science Data and Information System [1].

It is also important to keep sight of the overall problem. Data management is certainly an important part of the total system, but it is only one part nonetheless. Science data systems also contain many other components to provide functions such as data collection and analysis. For example, Figure 1 depicts the system context in which the integrated storage/data management prototype was developed.

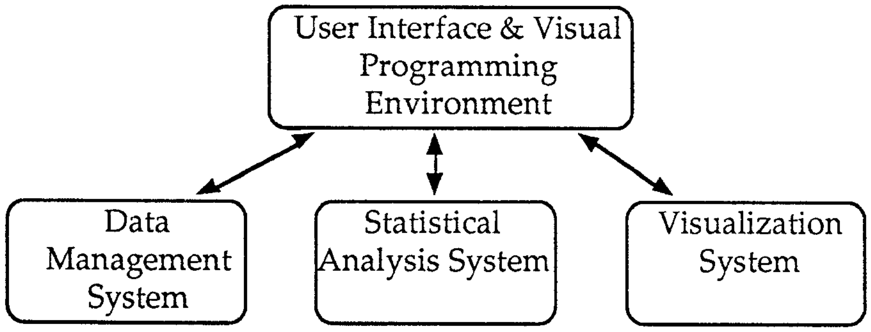
User Interface & Visual Programming Environment
Data Management System
Statistical Analysis System
Visualization System

Figure 1. System context. The data management system should provide high-level interfaces to facilitate construction of the overall system.

Application developers for these systems are likely to be using powerful tools for automatically generating graphical user interfaces and perhaps visual programming environments for construction of the application. They are likely to be unimpressed if the highest level function our data management system can provide is the capability to read a string of bytes. In this context, raising the system-provided data management services to the highest practical level should reduce the amount of custom application software required, thus reducing the development time and cost required to provide a powerful total system.

1.2. Problem

Science data systems often contain tens or hundreds of terabytes of data, making the cost of storage a primary concern. However, efforts to reduce storage costs generally make it more difficult for users and applications (hereafter called clients) to access the data. For example, typical systems place data in different storage subsystems based mostly on the size of the data and the cost of storage for each subsystem:

- Small, structured data are placed in database tables;
- Large datasets are placed in disk files;
- Very large or infrequently accessed datasets are placed in tape archives.

This separation of data is depicted in Figure 2.

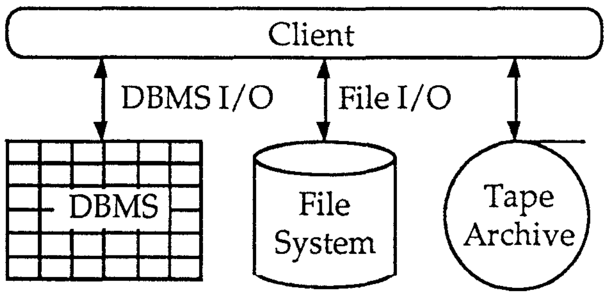

Figure 2. Traditional data system architecture. Separate interfaces to the DBMS, file system, and tape archive increase the complexity for applications and end users.

In current science data systems, for example, data are placed in the file system and tape archive, with the DBMS only containing metadata and application-defined pointers to the data (e.g., a dataset name). Thus, these systems require client intervention between the steps of locating the data (using the DBMS), retrieving the data (from the tape archive), and accessing the data (using the file system). This loose coupling between the data subsystems introduces the following problems:

- Clients become more complex (or more confused) and less productive because the semantics for accessing data depend on the location of the data.
- Software complexity and cost are increased because custom applications must actively participate in data management (e.g., to maintain referential integrity between the DBMS and file system).
- Innovative research is impeded, because manual procedures slow the analysis/synthesis cycle and because scientists are less likely to use data that is difficult to retrieve [2]. Further, excluding data from the DBMS increases dependencies on pre-defined metadata, which may not identify unexpected features of interest.

What is needed is a means of automatically managing the placement of data on different storage media and within different data subsystems to eliminate the problems described above while optimizing the cost and performance of data storage and retrieval.

1.3. Objective

Our primary objective is to make large volumes of data easily accessible by end users. By removing the syntactic and operational barriers between the DBMS, file system, and tape archive, we can simplify and speed access to data within very large information systems, where much of the data of interest may reside on tape (see Figure 3). For science data systems, this means we can remove the distinction between the DBMS and archive or, alternatively, allow the DBMS to contain large volumes of information such as browse data.

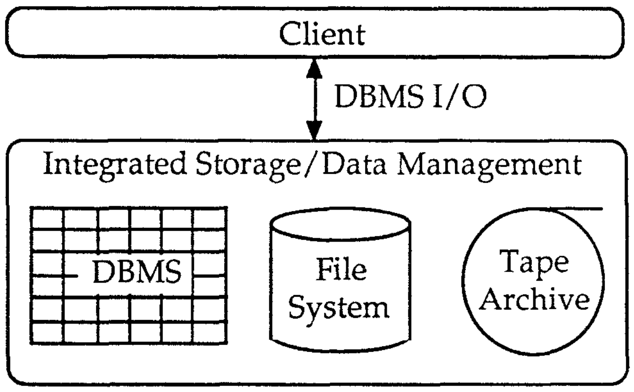

Figure 3. Goal data system architecture. A single interface to the data system with transparent storage management reduces the complexity for applications and users.

One way of achieving this objective is to enable the DBMS to access data in the file system and tape archive. While this capability has been sought by a number of people, DBMS implementations that provide this capability do not currently exist [3-5]. The concept of using hierarchical storage management to provide this capabiltiy is discussed below.

2. Concept Exploration

2.1. Hierarchical Storage Management

Hierarchical storage management is the process of managing the placement of data within a storage media hierarchy. Different storage media—such as memory, magnetic disk, and tape—have different cost, capacity, and performance characteristics. In general, media with the lowest access time has the highest cost, as show in Figure 4.

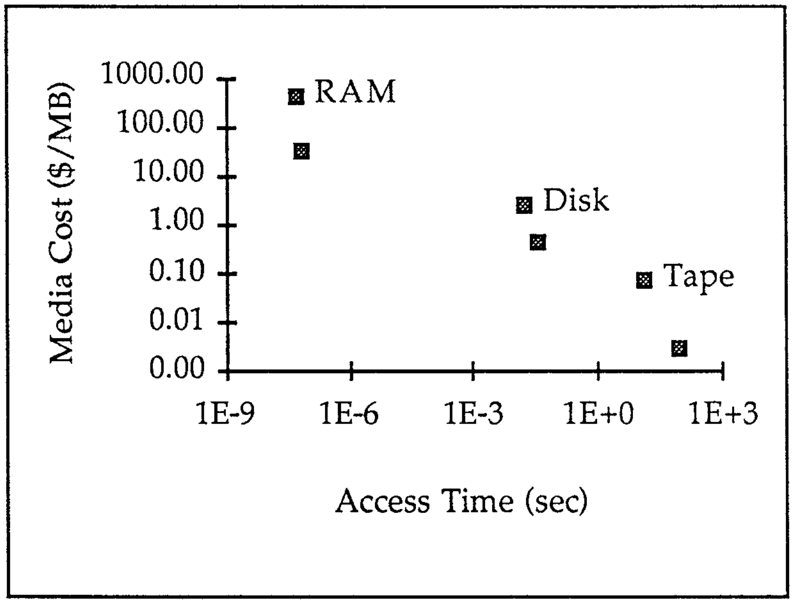

Figure 4. Cost versus access time for various media. The higher cost of fast storage media results in the use of slower media for bulk data storage.

The relatively high cost of fast storage media creates an incentive to keep only the most frequently accessed data there. Because the frequency of usage of different data tends to vary over time, the placement of data needs to be managed on an ongoing basis. Hierarchical storage management systems perform this task automatically. As data are accessed, they are "cached" to a higher level in the storage hierarchy. If there is not enough room at the higher level, some of the data there are "migrated" to a lower level. This movement of data within the hierarchy is depicted in Figure 5.

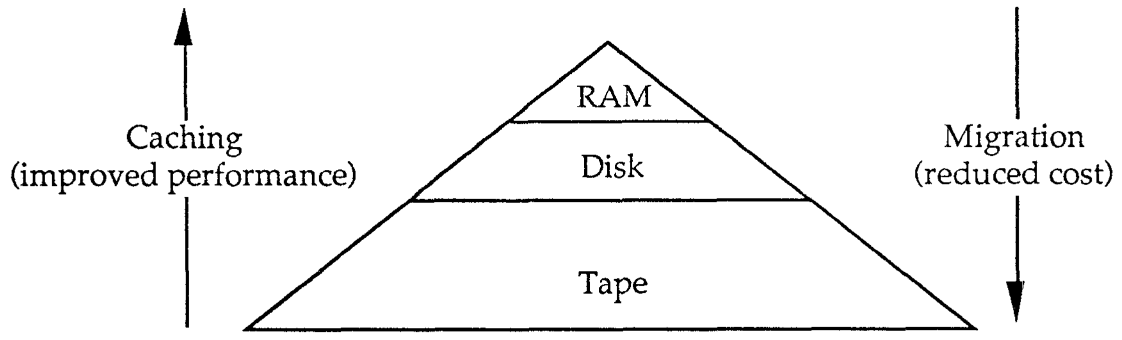

Figure 5. Storage media hierarchy. The cost/performance trade-off results in large storage capacities only on slower, cheaper media. The resulting structure can be viewed as a storage media hierarchy.

Note that the system will attempt to cache data which is most likely to be used in the near future, and migrate data which is least likely to be referenced in the near future. Hierarchical storage management systems typically employ a least-recently-used algorithm to accomplish this, under the assumption that data that have not been referenced in some time are not likely to be reference again soon. While this is clearly not the case in certain cyclical data references, the algorithm generally performs well in many environments.

A key assumption—a correct assumption for most hierarchical storage management systems—is that the location of the data within the hierarchy is not visible to users of the data. That is, clients of the system see the same interface to the system and access the data in the same way independent of the actual location of the data.

From this discussion, it is clear that hierarchical storage management has the potential to eliminate the problems discussed earlier by transparently managing the placement of data on different storage media. In fact, several commercial products are available which provide hierarchical storage management for data stored in a file system. Unfortunately, this potential is not realized for data stored in databases, because current DBMSs can only operate on data that are on disk. Use of tape requires an explicit export utility, which does not meet our need for a single, simple interface. In concept, however, hierarchical storage management could be added to the DBMS. This would provide the benefits of cost/performance optimization to data stored within the DBMS, yet maintain a single, high-level interface to the data.

3. Requirements

The concept presented above suggests that a solution may exist to meet the operational need. Before proceeding with design, however, a more precise statement of the system requirements is needed. The requirements for the prototype system are different in many ways from those of an operational system. In particular, the requirements for a prototype system are less precise to allow exploration in the design, implementation, and technologies employed.[2] An operational system will include more constraints to ensure fulfillment of the operational need (e.g., [6]). Nonetheless, the categories of requirements presented below are applicable to an operational system, and the specific requirements presented are representative of the full set of requirements needed for an operational system.

3.1. Functional Requirements

Data and Metadata Storage. The system shall provide the ability to store and retrieve satellite images (data) and descriptive information for each image (metadata).

Single Interface. The system shall provide access to both data and metadata through a single interface. This is intended to simplify the system from the viewpoint of users and applications.

[2]It is interesting to note that the conflict between a top-down system development methodology and system optimization is encountered already.

Set Retrieval. The system shall provide the ability to retrieve all images for a user-specified criterion. This allows users to obtain data that is relevant to a particular scientific study.

Criteria. The criteria available for set retrieval (see above) shall include location and time. The system shall allow a database administrator to define additional criteria.

Hierarchical Storage Management. The system shall manage the placement of data on different storage media to optimize cost and performance. This is intended to provide the best possible average access time at the lowest possible cost. Movement of data between storage media shall be automatic and not require human intervention. The system should manage the placement of data with a least-recently-used (LRU) algorithm.

Location Transparency. The system shall provide location transparency. Location transparency means that the mechanism and parameters used to access data are independent of its location, although the performance of data access may vary with its location. In particular, system managed movement of data as a result of hierarchical storage management shall not affect the mechanism or parameters used to access the data. This is intended to maximize the simplicity and portability of application programs.

3.2. Performance and Capacity Requirements

Access Latency. The system shall provide access to any single image in the tape archive within 5 minutes. Access latency is defined as the time between entry of a query by a user and the receipt of the data at the user interface, and may be measured on a quiescent system. This is intended to allow a user to obtain data in a near-interactive manner, and to enable interactive testing of the prototype.

Capacity. The system shall provide a minimum usable storage capacity of 50 gigabytes (GB). This capacity is adequate to handle a working set of satellite images and exceeds common workstation disk storage capacities to demonstrate the use of storage management.

Scalability. The system architecture shall allow the capacity to be increased to 1000 terabytes while meeting access latency requirement above.

3.3. Physical Requirements

Space. The system should occupy less volume than a system with the same storage capacity provided by disk storage.

Media Library Location. The system shall place the storage media library component in Bedford, Massachusetts. This is intended to allow use of the storage media library by the Bedford Research Computing Facility.

Development Access. The system shall allow programming, operations, and data access from McLean, Virginia.

3.4. Cost Requirements

Disk Equivalent. The system shall have a total cost that is less than $2 per megabyte of usable storage provided. This is intended to provide a total cost less than the cost of an equivalent system that uses magnetic disk for all storage.

Media Cost. The system shall provide storage on media with a present cost less than $0.01 per megabyte. This is intended to meet the long-term cost goals of the user.

Existing Equipment. The system should use existing equipment for the prototype to minimize cost.

4. Process Considerations

Many aspects of systems engineering are not reflected in the system development documentation. This is because many (if not most) aspects of systems engineering are processes, not products. Some of the processes that are not immediately evident in the development products are described below.

4.1. Project Management and Scheduling

Although systems engineering should not be confused with project management, there is a strong coupling between the two through the systems engineering management plan. First, the systems engineering process results in inputs to the schedule, including a work breakdown structure, technical reviews, and measurements of progress against various figures-of-merit. Second, scheduling results in inputs to the systems engineering process, primarily in the form of schedule considerations for system life-cycle cost calculations and trade-offs. For example, the buy-versus-build trade-off for the DBMS storage manager component was biased toward "build" because of the lengthy procurement time for a suitable commercial product relative to the overall system development schedule.

4.2. Engineering Specialty Integration

Typically, engineering specialty integration might include involvement of electrical engineers for circuit design, mechanical engineers for packaging design, manufacturing process engineers for manufacturability considerations, software engineers, safety engineers, reliability engineers, and so on. The results of this process are inputs to the derived requirements and trade-off analyses. However, the scope of the prototype is rather limited compared to modern complex systems, and thus involved only software engineers and procurement specialists.

4.3. Acquisition Management

In the case of the prototype, acquisition management primarily involves extraction of derived requirements and trade-off results into purchase orders for specific pieces of equipment (such as the storage media library).

4.4. Requirements Allocation

Only the final results of the allocation of requirements to functional components is evident in the development documentation. In the prototype, the requirements allocation process was informal because of the emphasis on proving technical feasibility, rather than on developing an operational system. In an operational system, a formal mechanism for allocation of requirements might be employed. In that case, the resulting product (typically a requirements allocation matrix) would provide more insight into the process.

4.5. Requirements Traceability

For the prototype, traceability of design specifications to requirements was ensured by a review process. In an operational system, this process might be visible as a product, such as a traceability matrix.

5. Architecture Trade-Offs and Analysis

There are two architectures which were considered for the prototype: a "top-to-top" (TT) architecture, and a "bottom-to-top" (BT) architecture. These two architectures are depicted in Figure 6. Although only the integration of the DBMS to file system is shown, the same architectures are applicable for integrating the file system and tape archive.

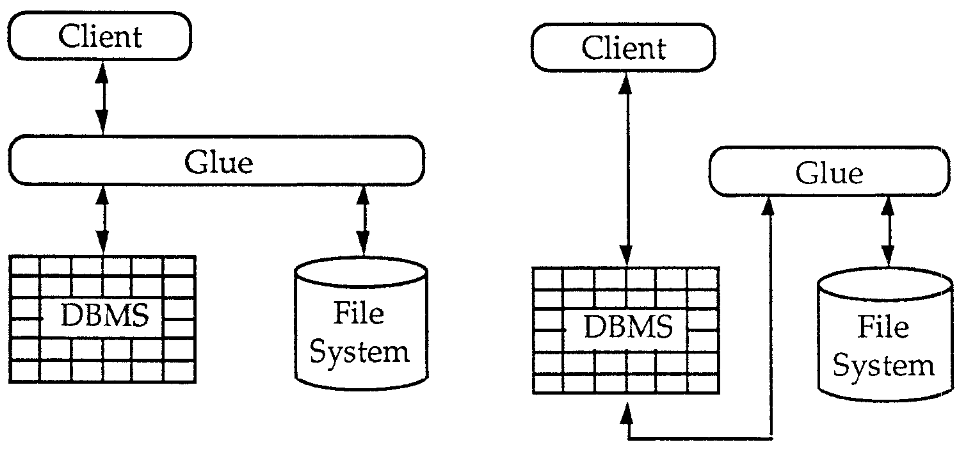

Figure 6. Alternative architectures. A top-to-top architecture (left) requires less modification to commercial software than a bottom-to-top architecture (right), but incurs some performance penalties.

The TT architecture involves intercepting client requests to the DBMS and using the normal interfaces to the DBMS and file system to perform storage management functions and to satisfy the request. Because the "glue" code uses only published interfaces to the DBMS and file system, the TT architecture does not require modifying any DBMS or file system software. Also, because the DBMS and file system interfaces are fairly well standardized, there is the possibility of using a commercial product to implement the glue code itself.

The BT architecture, by contrast, does not intercept client requests. Instead, it intercepts lower-level storage requests by the DBMS and uses internal DBMS interfaces to perform storage management functions. Because these internal interfaces are not exposed to applications, the BT architecture requires modifying the storage subsystem software of the DBMS. This is the approach most likely to be taken by the vendor of a commercial product, because it represents and incremental enhancement to their existing software. Further, because storage management functions are implemented at the physical storage layer, rather than through the standardized high-level (i.e., slow) interface of the DBMS, movement of data can be accomplished with much lower overhead. This means that an implementation based on the BT architecture would probably have better performance than one based on the TT architecture.

Criteria were established to compare the two architectures. Subjective weights were then assigned to each criteria and values assigned for the alternative architectures. The ranking is described in the following sections and summarized in Figure 7. This trade-off analysis indicates that the TT architecture is the preferred architecture. The wide spread in the total scores of the two alternatives suggests that this subjective ranking is sufficient, and that neither a detailed objective ranking nor a sensitivity analysis is required.

		Bottom-to-Top		Top-to-Top	
Criterion	Weight	Score	Weighted Score	Score	Weighted Score
Schedule	10	5	50	7	70
Cost	5	5	25	10	50
Performance	1	10	10	5	5
Portability	10	1	10	7	70
Transparency	10	10	100	10	100
Total			195		295

Figure 7. Alternative architecture ranking. The Top-to-Top architecture is preferred, primarily because it does not require modifying the DBMS software.

The following sections describe the criteria in the trade-off matrix.

5.1. Schedule

Because of external constraints (i.e., graduation), schedule was considered very important and weighted accordingly. The BT architecture would require procurement delays to obtain the DBMS source code—delays which might not allow completion within the external constraints. The TT architecture would potentially involve some delay to obtain or develop an integrating component, but these delays would probably be comparatively small. The scores assigned to each reflect the relative risk of each architecture to the schedule.

5.2. Cost

Cost was considered less important than schedule and weighted accordingly. Because the source code license required by the BT architecture is very costly, it was given a lower score than the TT architecture.

5.3. Performance

Because the prototype is intended to demonstrate a novel function, not high performance, performance was given a low weight. The performance of the BT architecture was estimated to be twice as good as that of the TT architecture, because it can take advantage of direct manipulation of files on disk rather than requiring data to be inserted from the file system into the DBMS.

5.4. Portability

The transfer of results of the prototype to other technologies is considered very important and weighted accordingly. The BT architecture, because it requires modification of a specific DBMS product, scores poorly. In contrast, the TT architecture can use the standard DBMS and file system interfaces, and thus is given a high score.

5.5. Transparency

Location transparency is a fundamental requirement, and is weighted accordingly. Both architectures provide complete location transparency, and thus are given a high score.

5.6. Design Trade-Offs

Additional trade-offs, similar to the one above, were performed for various design decisions. For example, a trade-off was performed which selected a relational DBMS over an object-oriented DBMS. The details are omitted here for conciseness. The following section describes the resultant design.

6. Prototype Design

In addition to the DBMS, file system, and tape archive, the prototype consists of two important components:

- A DBMS storage manager, and
- A file storage manager.

The structure of the prototype is depicted in Figure 8 and described in the following sections.

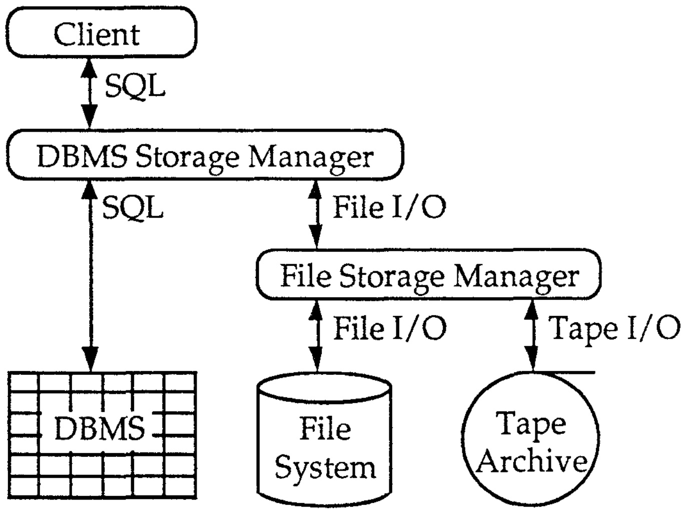

Figure 8. Prototype structure. The file storage manager and DBMS storage manager provide location transparency for data.

6.1. DBMS Storage Manager

The DBMS storage manager provides a standard Structured Query Language (SQL) interface for all client data accesses, including data in the file system, with automatic staging of data from the file system to the DBMS. It is based on the Teradata Transparency Interface Project (TIP)[3] software adapted for Unix. Typically, large fields (such as images) are stored in the file system, while the relatively small keys are stored in the database. This can be viewed as dividing a single virtual table vertically to create a long but skinny key table, as depicted in Figure 9.

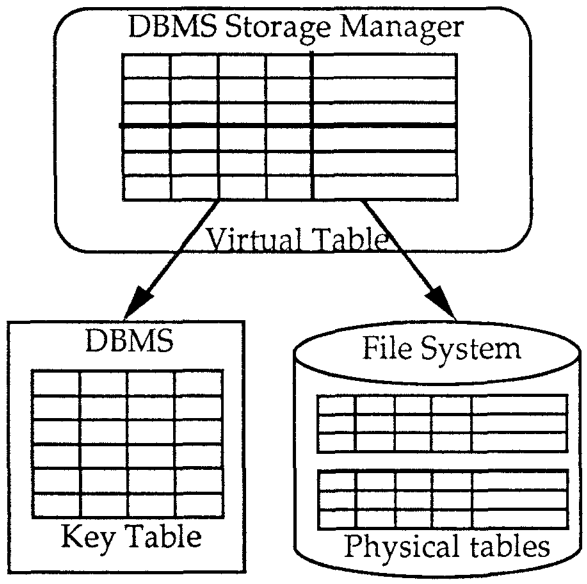

Figure 9. DBMS storage manager structure. Virtual tables are mapped to key tables (in the DBMS) and data files (in the file system).

The number of columns stored in the key table can range from one to all, with the specific choice made by a database administrator based on cost and performance goals. This decision is, of course, transparent to clients. A complete copy of the data is stored in one or more files in the file system. Maintaining a complete copy simplifies the process of staging to the DBMS and allows virtual table reconstruction for failure recovery and export/import.

[3]Previously called the Optical Call Level Interface.

Allowing multiple physical files per virtual table enables the virtual table to be physically divided horizontally to simplify inserts on write-once media and to give finer granularity for file migration to the archive (as we will see below).

When a query is received from the client, the DBMS storage manager parses the query and looks up the target table name in a system control table to determine if the target table is a real table or a virtual table. If it is a real table, the query is issued to the DBMS without further intervention. If it is a virtual table, the DBMS storage manager forms a sub-query that contains only the fields present in the key table and issues the sub-query to the key table. The result is a row subset of the virtual table that contains not the actual data from the virtual table, but pointers to records of the physical table(s) in the file system. This subset of records is staged from the file system to a temporary table in the DBMS. The original query is then reissued against the temporary table, and the results returned to the client. The granularity for staging from the file system to the DBMS is one record.

6.2. File Storage Manager

The file storage manager provides a standard Unix file system interface for all data accesses, including data on tape, with automatic caching from the tape archive to the file system. It is based on the UniTree Central File Manager software adapted to operate with the SunOS operating system and Exabyte EXB-120 tape library. A description of UniTree can be found in [7].

Physical tables, stored in files in the UniTree file system, are migrated to tape when not used and cached to disk on demand, as depicted in Figure 10.

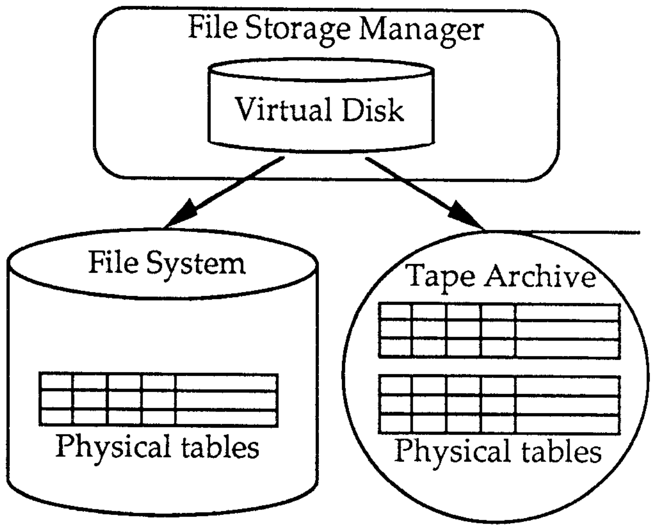

Figure 10. File storage management system. Files containing physical tables are migrated to tape and cached to disk on demand.

The granularity for caching is one file. Recall, however, that a virtual table can be divided into several physical tables, each stored in a separate file. Thus, clusters of rows can be migrated and cached separately. Rows representing older observations, for example, might not be accessed by a query, and thus remain on tape.

7. Implementation Trade-Offs

Trade-off analyses were performed to select implementations for each component of the prototype. These included selection of the following:

- UniTree Central File Manager over Epoch Renaissance and Epoch Dynasty for the file storage manager.
- Teradata Transparency Interface Project software over a commercial DBMS integration product for the DBMS storage manager.
- Magnetic tape over optical disk for the storage media.
- Exabyte EXB-120 tape library over the Summus JBL-125 for the storage media library.
- Sun SPARC 2 over MIPS-based workstations for the server platform.

Unlike the design trade-offs, the alternatives in the implementation trade-offs were often quite close, so objective criteria were required. Most of the trade-offs are omitted for conciseness. However, the tape library selection is described below as an example of an objective trade-off. In particular, note that each criterion is given an objective value, rather than a subjective score. The actual score is computed based on relatively objective thresholds determined by the requirements. Note, however, that the existence of subjective values for the weights means that the analysis is not truly objective and is thus subject to bias. For this reason, a sensitivity analysis is required to determine if a small shift in the weights could change the outcome of the trade-off.

Figure 11 shows the objective ranking matrix for the candidate tape libraries. The rows of the matrix are the criteria described above. The columns have the following meanings:

Criteria	The objective evaluation criteria.
Units	The measurement units of the specified criteria. For Boolean criteria, this field is left blank.
Weight	The subjective weight of importance, as derived from the requirements of the system. Note that only the relative values of the weights are important: criteria that are twice as important as other criteria should have a weight that is twice as great. This must be true for groups of criteria as well as each criterion taken individually.
Unacceptable	The value beyond which all values are approximately equal in that they do not meet the established criteria.
Perfect	The value beyond which all values are approximately equal in that they completely satisfy the established criteria.
Value	The actual value of the specified criteria for a given alternative.
Score	The objective score for the alternative based on the internal scoring function. In the matrix shown, a sigmoid curve of the form $1/(1+e^{-kx})$ is used.
Wtd. Score	The weighted score formed by the simple product of the criterion weight and the score.

					System A			System B		
Objective Criteria	Units	Weight	Unacc-eptable	Perfect	Value	Score	Wtd. Score	Value	Score	Wtd. Score
Cost	$K	0.9	100	10	50	0.64	0.57	25	0.97	0.87
Capacity	GB	0.9	50	1000	580	0.68	0.61	125	0.02	0.02
Media cost	$/MB	0.5	2.5	0	0.002	0.99	0.50	0.002	0.99	0.50
Media/drive ratio	GB/dr	0.2	500	5	145	0.90	0.18	62.5	0.98	0.20
Weight (lower)	lbs		0	100	147	1.00		65	0.82	
Weight (upper)	"		1000	200	147	1.00		65	1.00	
Weight (aggregate)	"	0.2				1.00	0.20		0.82	0.16
Port		0.2	0	1	1	0.99	0.20	0	0.01	0.00
SCSI Interface		0.2	0	1	1	0.99	0.20	0	0.01	0.00
Standard drives		0.4	0	1	1	0.99	0.40	0	0.01	0.00
Company revenues	$M	0.4	1	1000	1500	1.00	0.40	75	0.01	0.01
Month in production	mo.	0.2	0	12	1	0.02	0.00	9	0.92	0.18
Total		2.5					1.861			1.581
Total (percent)		100%					74%			63%

Figure 11. Objective ranking matrix for alternative tape libraries. The similarity between implementations requires an objective trade-off to select the preferred alternative. Note that system "A" is the preferred alternative in spite of its higher cost and relatively high weighting of cost as a criterion.

The following sections describe the criteria used in the trade-off.

7.1. Cost

The cost of the tape library hardware. Because the two alternatives provide the same basic functionality, cost is the most important factor and weighted accordingly.

7.2. Capacity

The capacity of the library, derived from the number of tapes times the capacity per tape. The prototype needs to show a reasonable capacity to provide a convincing demonstration, but the truly large capacity demanded by an operational system is not needed.

7.3. Media Cost

The cost per megabyte for the tape handled by the library. The tape used by the two alternatives is the same, so this criterion does not affect the trade-off results.

7.4. Media/Drive Ratio

The average amount of media serviced per drive. Large media/drive ratios will result in poorer performance because requests for data may have to wait for an available drive.

7.5. Weight

The physical weight of the unit. This measure also serves as a surrogate measure of quality. Because tape libraries have a large mechanical component to move tapes to drives, they must be sturdy. Light units are likely to be made of cheap materials and thus be more subject to mechanical failures. Very heavy units, on the other hand, present logistical problems for installation. The aggregate scoring function is a "band-pass" function with both a lower limit and upper limit to the acceptable range.

7.6. Port

Some units provide a port which allows a tape to be entered into the library without interrupting operations. This is more important for operational use than a research prototype, and is weighted accordingly.

7.7. SCSI Interface

A Small Computer System Interface (SCSI) allows the use of standard device drivers and provides better error reporting than other interfaces.

7.8. Standard Drives

Some units modify the tape drives to facilitate robotic access, which requires that drives be purchased from the tape library vendor. By contrast, the use of standard drives simplifies drive maintenance and replacement.

7.9. Company Revenues

This is a surrogate measure for stability of the vendor. The mechanical nature of tape libraries and subsequent need for relatively frequent adjustment and repair means that it is desirable to have some assurance that the company will be in existence to maintain the product.

7.10. Months in Production

This is used as a predictor of problems that might interfere with research or operational use of the equipment. Products under a year old are deemed more likely to have problems in the field. If accurate mean time between failure (MTBF) estimates were available, they could be used instead.

7.11 Results

The trade-off analysis resulted in the selection of the Exabyte EXB-120 Cartridge Handling System. The following section briefly describes the other system components and the configuration of the prototype system.

8. Prototype Implementation

8.1. Physical Configuration

The physical configuration of the prototype is depicted in Figure 12. Note that the components for the file storage manager are located in Bedford, Massachusetts, while the components for the DBMS storage manager are located in McLean, Virginia. This configuration was dictated by other uses of the equipment unrelated to the prototype, but coincidentally allowed investigation of some effects of geographic distribution on the design.

The prototype implementation is capable of managing databases over 500 GB in size. An operational system would undoubtedly increase the cache size for better performance and large file handling. Nonetheless, the prototype successfully demonstrates relational access to this volume of data at a system cost of about 40¢ per megabyte, suggesting that very large databases are feasible from a cost perspective.

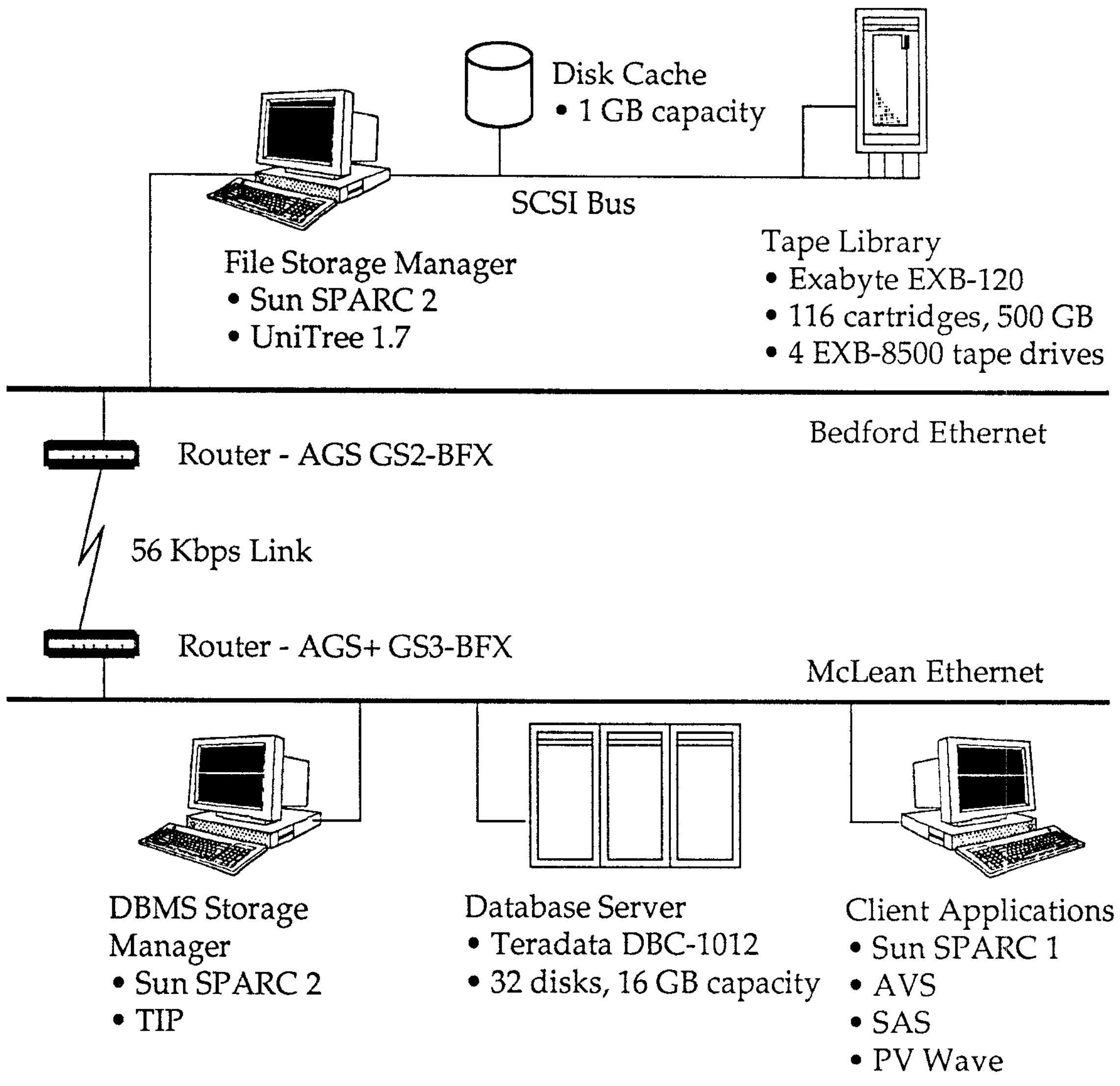

Figure 12. Prototype physical configuration. Resources are geographically distributed as typical of real systems.

8.2. Operational Flow

The response to a client query is depicted in Figure 13.

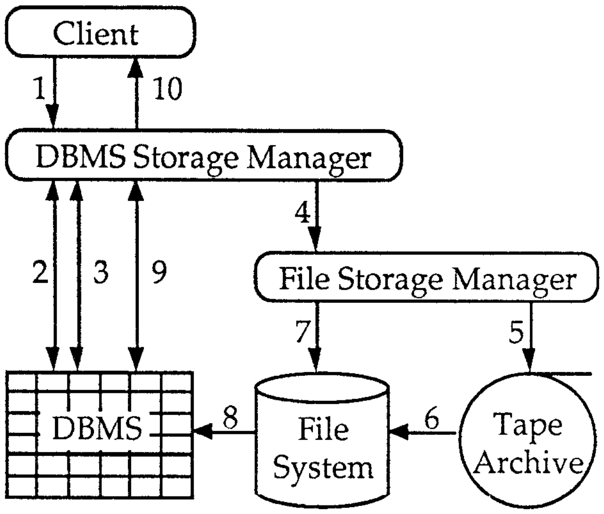

Figure 13. Prototype response to a data request. The system may perform many storage management activities transparent to the client.

In the case where the selected rows are in the tape archive, the following steps are performed:

1. The client issues a query requesting the desired data.

2. The DBMS storage manager issues a control query to determine the physical location of the data. The response to the query indicates that the physical table resides in the file system.

3. The DBMS storage manager issues a sub-query (keys only) and retrieves a row subset pointing to records in the file system.

4. The DBMS storage manager accesses the file storage manager for the specified records, including non-key fields.

5, 6. The file storage manager recognizes that the needed files are not resident in the file system, mounts the proper tape, and stages the necessary files to the file system.

7, 8. The DBMS storage manager reads the record subset through the file storage manager and inserts the records as rows into a temporary table in the database.

9, 10. The DBMS storage manager reissues the original query and returns the results to the client.

Subsequent queries to the same physical table are likely to execute much more quickly, because the data will probably be found in the file system cache and the tape archive will not need to be accessed.

8.3. Performance Optimization

Early in the design of the prototype we realized that inserting data into the DBMS would be time consuming. Experimentation with the prototype confirmed this: retrieving sixteen 24KB satellite images required about one minute to stage the file from tape to a disk file, and another minute to stage the rows from the disk file to the database. For the row insertion step, response time becomes progressively worse as the number of rows increases.

The reason for inserting the records into the database is to avoid having to implement a full SQL processor in the DBMS storage manager. The final query (Figure 13, step 9) may have to process non-key predicates, data formatting qualifiers in the "project" clause, "having" clauses, and so on. It would be self-defeating to re-implement a full DBMS on the file system.

However, there are common queries that require very little additional processing. In particular, if all the fields referenced in the "where" clause are present in the key table, all of the records indicated by the sub-query to the key table (Figure 13, step 3) will be returned to the client. It seems unproductive to insert them into the database only to retrieve all of them again.

The DBMS storage manager was changed to recognize this special case, order the fields according to the "project" clause, and return the data directly to the client. This bypass is depicted in Figure 14, step 8.

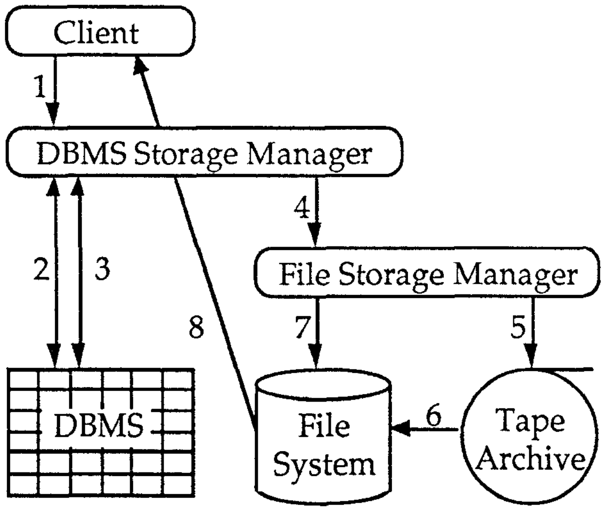

Figure 14. DBMS insert bypass for special cases. Avoiding inserts reduces query latency more than 95%.

The key table was also revised so that it contained all the searchable metadata for each satellite image, but not the image itself (which appears in the "project" clause but never appears in the "where" clause). This met the requirements for the special case, and reduced the latency introduced by the DBMS storage manager in our test scenario from about one minute to a few seconds—a reduction of more than 95%. This performance improvement is shown in more detail in Figure 15. Note that without the performance optimization, the query time is substantial, even if no rows are selected. This is due to the overhead associated with establishing a session with the DBMS, and because some rows meet the criteria of the intermediate query (Figure 13, step 3) and must be inserted into the temporary table, even though none of these rows meet the criteria of the final query. With a larger number of rows (more than 44), the marginal performance cost of inserting additional rows is more apparent. It is also interesting to note that with the bypass, performance is actually better than with an unmodified DBMS when the number of rows is large. This is because the file system has better performance characteristics for delivering large amounts of data than the DBMS in the test configuration.[4] The bypass case essentially adds a small overhead to what is otherwise local file system I/O.

---

[4]In the test configuration, the DBMS was attached via Ethernet, while the file system used local disks.

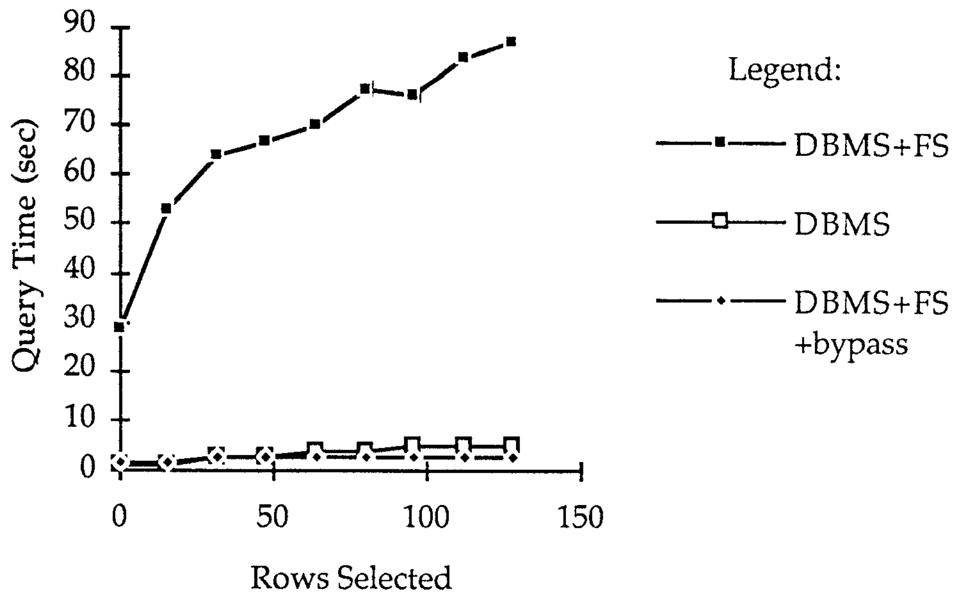

Figure 15. Query performance with and without optimization. Performance is shown for three cases: all data in the DBMS , data in the DBMS and File System, and data in the DBMS and File System with the insert bypass optimization added. Note that the time to cache data from tape to disk is not included. For the DBMS+FS case, 22 rows are inserted for every 16 rows selected.

9. Future Work

There are many areas of the prototype that would benefit from further investigation. Some of them are described in the following sections.

9.1. Database Row Caching

The prototype currently discards the data staged in the temporary tables after the successful completion of each query. In the cases where inserts cannot be completely bypassed, it should be possible to retain data in the temporary tables and avoid inserting rows that have been previously inserted and thereby improve performance.

9.2. Commercial DBMS Storage Manager

The DBMS storage manager is a special case of a heterogeneous database manager, of which several commercial implementations exist. Replacing the DBMS storage manager with one of these commercial products would allow the prototype to be connected to many different types of DBMSs.

9.3. Other DBMS Extensions

The DBMS storage manager interprets SQL to provide the storage management extension to the DBMS. Similarly, many other extensions—such as spatial indices, integration with a geographic information system, or user-defined functions—could be added. Some of these extensions might be transparent, and some might be provided as extensions to the standard query language. In particular, integration of this prototype with the R-tree spatial index prototype at the University of Maryland [8] would be interesting, because it uses a system structure compatible with the prototype described here.

9.4. DBMS Name Server

The prototype currently references tables stored in the file system by file name, requiring a file name lookup. Replacing the file name with the Bitfile Identifier would result in a DBMS-based Name Server per the IEEE Mass Storage Reference Model [9]. Note, however, that we exploited the non-uniqueness of file names by redirecting the DBMS storage manager to both local and remote directories for test purposes—a capability that would be lost with this change.

9.5. Object-Oriented DBMS Storage Manager

The current DBMS storage manager migrates and caches tables or portions of tables. Similarly, it should be possible to migrate and cache objects in an object-oriented DBMS. The recursive structure of objects is more amenable to fine granularity caching than is the definition of tables.

10. Conclusions

The prototype presented above successfully integrates data management with storage management. This effectively provides a high-level DBMS interface to large volumes of data stored in tape archives, and suggests that it is feasible from both technical and cost perspectives to store not just metadata, but large satellite images in a DBMS. The actual location of the data, whether in the DBMS, file system, or tape archive, is transparent to clients, allowing cost/performance optimization to be performed without affecting the interface to the client. Further, in common cases where database inserts can be bypassed, the performance of the prototype is roughly equivalent to file system I/O.

11. References

1. William Callicott. "Data management in NOAA". *Goddard Conference on Mass Storage Systems and Technologies*. NASA Goddard Space Flight Center, 22-24 September 1992.

2. Teradata Corporation. *Optical Call-Level Interface: Concepts and Facilities Manual,* OCLI-0010-01, pp. 21-26. Teradata Corporation, 1991.

3. Sara Graffunder. "User requirements for large databases in a mass storage environment". *Digest of Papers: Eleventh IEEE Symposium on Mass Storage Systems: Crisis in Mass Storage,* 7-10 October 1991. Los Alamitos, CA: IEEE Computer Society Press, 1991.

4. W. Farrell and Jean Anderson. "Very large databases and mass storage technology". *Digest of Papers: Eleventh IEEE Symposium on Mass Storage Systems: Crisis in Mass Storage,* 7-10 October 1991. Los Alamitos, CA: IEEE Computer Society Press, 1991.

5. Greg Kenly. "Opportunities for hierarchical data management in the relational database environment". *Digest of Papers: Eleventh IEEE Symposium on Mass Storage Systems: Crisis in Mass Storage,* 7-10 October 1991. Los Alamitos, CA: IEEE Computer Society Press, 1991.

6. NASA. *Functional and Performance Requirements Specification for the Earth Observing System Data and Information System (EOSDIS) Core System.* NASA Goddard Space Flight Center, 14 September 1990.

7. General Atomics DISCOS Division.*UniTree Training Manual: Internals.* General Atomics, April 1991.

8. Nickolos Roussopoulos, Timos Sellis, and Stephen Kelley. "The R-Tree software", 20 July 1992.

9. Sam Coleman and Stever Miller, ed. *Mass Storage System Reference Model: Version 4*, May 1990.