An Architecture for Using Tertiary Storage in a Data Warehouse

Theodore Johnson
johnsont@research.att.com
AT&T Laboratories – Research
Florham Park, NJ 07932
tel +1-973-360-8779
fax +1-973-360-8050

Abstract

In this paper, we present an architecture for a data warehouse that provides convenient, flexible, and high-performance access to tape-resident data. Data warehouses allow an organization to store and analyze operational data. Many data warehouses (e.g., for telecommunications) create very large data sets that can be economically stored only on tertiary storage. At the same time, data analysts need to make a wide variety of decision support and data mining queries on the data. While one can (and typically does) extract compact summaries of the warehoused data, many queries can only be answered by using data from the full data set.

Using database systems to access large tape resident tables of small objects is conventionally viewed as a difficult problem, because it is easy to express an query that takes a very long time to process (e.g., the join of two tape-resident tables). Fortunately, decision support queries can usually be expressed in a way that can be implemented efficiently on tape-resident data. In addition, many data mining algorithms have been optimized to make a small number of passes over a data set. We present the features of a tape-based data warehousing system that provides efficient support for data mining and decision support queries on very large collections of small objects, a prototype implementation, and preliminary performance results.

1 Introduction

Interest in data warehousing has increased dramatically in recent years because of the vast increases in storage capacity of moderately priced on-line and near-line storage systems. Users from a wide variety of applications find that they can store and make use of information that previously was discarded or was inaccessibly stored in off-line tape racks. In a typical scenario, a data set (such as regional sales) is periodically harvested and ingested into the data warehouse. The data set is heavily indexed to accelerate decision support queries on subsets of the data. For example, a query on the sales database might be, “What is the weekly trend of shoe sales, listed by supplier and state?”.

A commonly used data organization is a *star schema*. The primary table (called the *fact table* or the *detail data*) is indexed for joins against supporting tables (sometimes called "dimension tables"). The dimension tables allow selections of subsets for analysis. For example, dimension tables for the sales fact table might include a table on suppliers, on locations, and on products. The example query can be evaluated by finding all entries in the products table related to shoes, and performing a foreign key join on the sales table. Special indices have been developed for this type of operation, including join indices [20] and bitmap indices [14].

To facilitate decision support queries, summaries of the detail data are precomputed (companies such as Arbor and Red Brick specialize in databases for these type of "cube queries"). However, queries that cannot be answered from the summary tables are answered from the detail data. In addition, the detail data is made available for data mining queries.

Although the steep declines in the price of on-line storage has encouraged the increased use of data warehouses, many applications generate more data than can economically be stored on-line. One example is telecommunications data warehouses. AT&T collects very large volumes of billing and usage data for each of the services it provides. This data is used for a wide variety of applications, including identifying business trends, validation of billing procedures, detecting fraud, monitoring and optimizing network performance, and debugging network problems (other telecommunications companies have similar needs).

When such large volumes of data are collected, only the most recent detail data is stored on-line. Older detail data is migrated to tape, and only small summary tables are retained in secondary storage. When old detail data is queried, it is migrated back to secondary storage for analysis.

The current storage and structure of tape-resident detail data discourages experimentation. While HSM products can simplify the process of migrating data between levels of storage, the large-scale processing of tape-resident resident data remains difficult. Data that resides in a database must be exported to migrate it to tape, and imported to use it when migrated back to disk. The alternative to importing and exporting database tables is to use radically different methods for querying disk resident and tape resident data. While the user would like to give a declarative specification of the data set to be processed, the tape-resident files must be explicitly named and imported. If the total size of the queried data is larger than the migration cache space, the user might need to implement their own cache management algorithms (depending on the sophistication of the HSM cache management algorithms).

Data analysts would prefer to have a database-centric view of warehouse data instead of a file system-centric view, as common queries are easy to express with powerful query languages or 4GL tools. However, interfacing databases with tertiary storage is viewed as a difficult problem (see Section 7 for an discussion of related work). The problem is that databases which support standard query languages such as SQL (Structured Query Language) are designed for disk-resident data. Disk-resident file systems permit fast access to a large number of open files, fast random access, and updates in place, and database systems take advantage of these characteristics. Tape-resident data files have long access

latencies, provide slow random access, and tape data is largely append-only. As a result, it is easy to express a query that is very hard to answer.

We observe that decision support data warehouses [17] have characteristics that make the use of an automated tape library for storage a reasonable option. First, the fact table is append only. Second, most queries make very long sequential scans through the data. Third, joins to tape resident data take restricted forms, being either joins to disk-resident "dimension tables", or are the equivalent of merging vertically partitioned[1] relations.

Our thesis is that a simple special purpose database can make automated tape library resident data warehouses efficient and easy to use. A small set of efficient access methods can be used as the engine for implementing a wide range of decision support queries and data mining algorithms. Because we know the schema of the data and the nature of the queries on the data, we can layout the data on tape in an optimal manner. By specifying the query in a declarative language, we can build a query plan out of optimized components.

In this paper, we present our views about how such a database should be built. Next, we discuss an architecture for such a system, along with a preliminary discussion of an API. We built a prototype database using the architecture. We discuss how we used the prototype to submit a query, and discuss performance results.

2 Motivation

This work was motivated by the need to perform data warehousing and data mining for telecommunications data analysis. In this section, we will discuss in greater detail the characteristics of typical queries in this problem domain, and show how they are similar to decision support type queries.

Queries over telecommunications data sets are made over sequences of descriptions of telephone calls, login sessions, circuit connections, data transfers, etc. A typical query selects a subset of data, joins each selected tuple to the dimension tables, and computes an aggregate over the joined tuples. The aggregates can require a very large number of tuples to compute accurately. For example, the tail of the distribution of the duration of circuit connections is of great interest. Often, only a subset of the data set is selected for aggregation. The tuples in the data set are small, on the order of 100 bytes or less each.

Telecommunications databases often contain tuples that express sequences of events. For example, one tuple indicates the setup of a connection, a sequence of tuples indicate the status of the call, and a terminating tuple indicates the end of a call. A common query is to report aggregates about such sequences. In some applications, one must be able to retrieve all records related to an individual user. For example, AT&T must store and retrieve telephone call records, in response to law enforcement requests. Often, the selectivity of

[1] *vertically partitioning* a relation means to split each tuple of a relation into two or more pieces, with each piece stored in a different file

these queries is on the order of one record retrieved per billion archived.

Decision support queries in the telecommunications domain have a great deal of similarity to generally accepted notions of what constitutes decision support queries in other domains (e.g., sales, financial, etc). Therefore, we believe that a system which supports telecommunications data warehouses will support general purpose data warehouses as well.

Modern automated tape libraries can support high data rates, whether through the prodigious performance of high end drives, or by using multiple moderately priced drives. However, access latencies and seek times are very large. Therefore, systems that make use of tape storage must access data in long sequential scans in order to achieve high performance. We observe that most data warehouse accesses to tape resident data can be made through long sequential scans, because the queries have the following characteristics.

- Most queries compute large-scale aggregates on the fact table. Often, the queries are simple trend or classification queries. Recent research [1] has shown that has shown that even more complex queries involving layers of conditions on the computed aggregates (e.g., "find all calls longer than the average call") can be made in a small number of sequential passes over the data set.

- Joins involving a tape-resident table have a limited set of forms. Most joins are to join the fact table to one or more of the dimension tables. The dimension tables are usually small, and can be pre-loaded into memory.

  Joins between tape-resident tables are often made between time windows of the respective tables. Tape resident tables might be joined because they represent vertical partitions of a tape resident table. Alternatively, the tables might represent different views of the same activity, collected by different processes (e.g., shipping records and billing records). While the table joins can become complex, joins between tape-resident tables resemble a multiway merge, instead of the cross product as has been investigated in the previous literature. Because of the temporal nature of the fact tables, merging is the natural way to join fact tables.

- The subsets selected for further processing are often fairly dense (e.g., one record selected in 100 or more). Since most disk blocks are accessed, the best query plan for performing the selection is to sequentially read the table, but to unpack and process only the selected records. Many database systems (e.g., Sybase) use *bitmap indices* on ROLAP data to support this type of query processing [14]. The sequential nature of the query plan makes it easy to implement on tape storage.

An additional requirement of telecommunications databases is to find records that occur very infrequently. We refer the interested reader to a related paper [6] for details on this type of tertiary storage indexing.

3 Design

We aim our design at the midrange systems commonly used as database servers: $O(10)$ high speed processors with $O(1\ \text{Gbyte})$ of main memory and $O(1\ \text{Tbyte})$ of on-line storage. Servers of this scale are inexpensive enough to become dedicated warehouse servers. Our goal is to extend the memory hierarchy by another one or two orders of magnitude with a moderate increment in cost by using an automated tape library holding $O(100\ \text{Tbyte})$ of data. Access to the tape resident data should be fast and efficient. At the least, querying tape resident data should be simple, at the best, transparent.

Given the nature of the data sets, the nature of the queries on the data sets, the technology used in database servers, and the performance characteristics of automated tape libraries (see [7]), we feel that efficient access to tape resident data dictates the characteristics of the data warehouse. Furthermore, we need a flexible design to support data mining and ad-hoc queries. In particular, we feel that the data warehouse should have the following characteristics:

1. **Lightweight architecture:** Conventional database systems incorporate locking, recovery, client-server computing, etc. Each of these features adds overhead to the data handling, but provides little benefit in our context. By leaving out these features (i.e., client-server) or using a lightweight implementation (i.e., locking and recovery) we can achieve a much higher throughput (other implementors [4] have taken the same approach). Similarly, implementing SQL on a tape-resident table involves immense difficulties in query planning, scheduling, data allocation, and so on. By restricting the query language, we can efficiently support the queries that a user would typically ask.

2. **Control over data layout:** Developing a data warehouse requires a great deal of planning. So we can assume that the data warehouse implementor understands what are typical queries on the data. For example, if a typical query makes a very long scan over a tape resident table, then the table can be horizontally partitioned[2], and the partitions distributed among n tapes. Other considerations might lead to an even more complex layout.

3. **Direct tape to memory transfers** Many architectures for implementing databases on tape resident data assume that the tape resident data is first loaded to disk, and then regular disk-oriented database algorithms are applied to the data. While this architecture is appropriate for some applications, we believe that for our domain processing data directly from tape is the better approach.

   Loading data from tape to disk before processing allows the use of existing database technologies, but it also creates two problems. First, transferring the files from tape

[2] *Horizontally partitioning* a relation means dividing the tuples into different tables, based on the value of the tuple.

to disk before use wastes resources – cache disk space, I/O bandwidth, etc. Second, managing the cache disk resources becomes a difficult problem itself, and the database system becomes complex, more difficult to implement, and more difficult to optimize.

Loading the tape resident data onto disk can improve throughput if the cached data is accessed repeatedly. However, we expect that this benefit to be minimal. First, typical accesses to tape resident data are likely to be long sequential scans, so repeat accesses before a cache flush are unlikely. Second, accesses to tables cached on disk are likely to also be sequential scans, and therefore can be performed efficiently from tape. Third, our architecture does not preclude loading a view of a tape-resident table onto disk for intensive analysis.

4. **Support for I/O parallelism:** Although tape drives have very slow seek times, modern tape drives have high transfer rates. Moderately priced high capacity tape drives, such as the DLT 7000, can provide a transfer rate of 7 Mbytes/sec or greater using reasonable assumptions about data compressibility. Furthermore, a collection of these drives, in parallel, can provide very high data rates. These high data rates are necessary because of the volume of data to be processed, and because in most cases selections are based on sequential scans.

5. **Support for CPU parallelism:** The complex aggregation of typical queries can be cpu intensive. Certainly, the high data rates available from the tape drives requires multiple CPUs to keep pace with the data processing. In addition, tape-resident data might need to be joined with disk resident data before processing.

6. **Support for indexing** While tapes are an inherently sequential media, indexing can significantly improve performance. Many decision support type queries process small subsets of the tape resident table. Subset membership is often determined by the value of an attribute, and therefore selections can be accelerated by using indices. Even if every tape block must be read into memory, unpacking and processing only the selected records can greatly reduce CPU requirements. We note that record unpacking might involve significant computation, because our studies have shown [7] that pre-compressing records can improve on the effective tape capacity and transfer rate.

   Existing work on indexing tape resident data uses technology developed for disk-resident data, or uses "metadata" to describe the contents of tape-resident files. These techniques are not sufficient for indexing tape-resident tables of small objects. We are researching more suitable indexing technology. For example, in [6], we describe an algorithm for indexing individual small records in a multi-terabyte tape-resident table, in which a typical key value will occur only a few times on a tape. We are refining this technology to make it applicable to cases where keys occur frequently.

7. **Support for massive aggregation and data mining:** As we have discussed, most decision support and data mining queries can be expressed as taking aggregates over a very large data set. Data mining queries and "complex" aggregation [1] might require two or more passes through the data set. In addition, the data can have the

semantics of a large number of interleaved time series. Aggregates on a time series are likely to group by the object that created the time series. This creates problems for parallel processing, because time series will cross partition boundaries.

4 Architecture

We have designed a prototype architecture to test our ideas. The architecture has a component that handles data ingest and a component that handles queries. The two components communicate through a control database as shown in Figure 1.

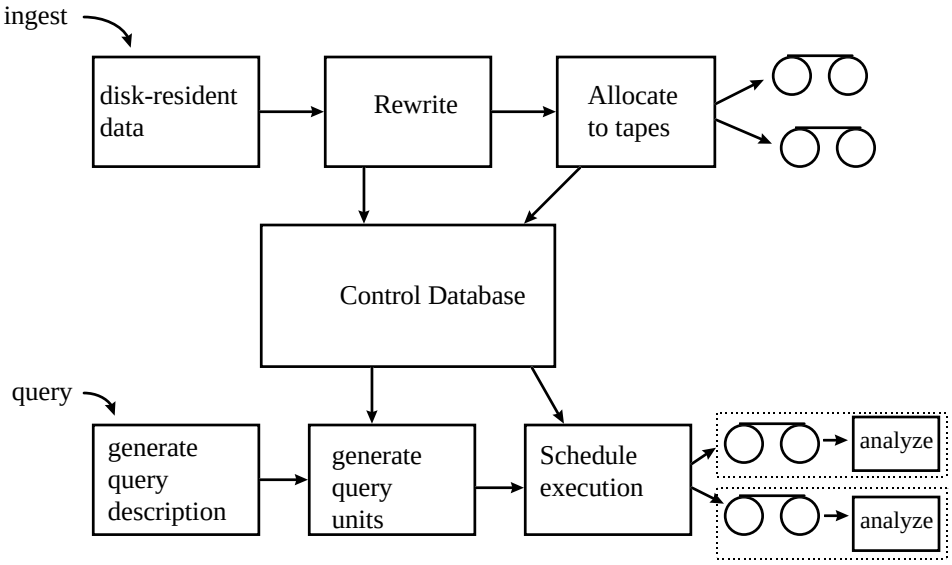

Figure 1: Data warehousing architecture.

We assume that data warehousing system has a disk-resident component and a tape-resident component. Updates to the fact table are put into a disk-resident table. Storing the newest data on disk is necessary because the newest data is usually the hottest data, and because summary tables must be built on the updates to the fact table. To make space for the newest data, older data must be migrated out. At this point a migration process is initiated that operates over the oldest portion of the disk resident fact table, transforming the data, allocating the data to tapes, and performing the migration.

Data transformation is necessary for good performance on tape-based queries, because disk and tape have very different performance characteristics [7]. In particular, queries on disk-resident data can support random access, while tape queries must primarily use sequential access. Strategies for improving tape access speed include vertical partitioning, partitioning on key value, selecting horizontal partitions for efficient seeks, replication, and resorting. Indexing tape resident data requires a different approach than indexing disk resident. The indices for disk resident data are dropped and indices for tape resident data are built.

Disk resident horizontal partitions are transformed into a set of tape resident files, which are then migrated to tape. The allocation strategy has two dimensions – the relative placement of files derived from the same disk-resident table partition, and the placement of files derived from successive partitions. The data placement strategy provides the opportunity for parallel I/O, as the data required for a query can be placed on multiple tapes, all of which can be read simultaneously.

Queries are initiated by submitting a *query description* to the system. A query description consists of a a specification of a *backend server*, of an *aggregation server* and a list of *query units*. The backend server reads tuples from tape, performs selections, projections and buffering, and distributes the results to the instances of the aggregation servers that have been started. The aggregation server performs joins on the tuples it receives from the backend server and computes the aggregate functions. A query unit is a minimal set of tape-resident data files that must be accessed together to perform a query. We assume that the invoking program generates the list of query units, although the control database is available to aid in generating the list. By this separation of responsibilities, a wide variety of programs can make use of the basic querying engine (query languages, data mining algorithms, user-written programs, etc.).

5 Query Language

In addition to a convenient programming interface, we are developing a query language for use in expressing ad-hoc queries. The language is similar to SQL, but it has several restrictions (to ensure efficiency) and several extensions (to provide the desired expressive power).

The restriction is that difficult tape joins are disallowed. Tape-resident tables can be joined to disk-resident tables. In addition, tape-resident tables can be merged. However, "cross product" joins between tape-resident tables are prohibited.

The query language is based on the extensions to SQL discussed in [1]. Instead of relying on joins to compute complex functions, "grouping variables" are used to express complex conditions. Queries expressed in this language are easy to optimize for execution on tertiary storage. We will publish details of this language at a further date.

6 Preliminary Implementation

We developed a preliminary implementation of our data warehouse architecture. The migration component will call a user-supplied program to transform a segment of data from its disk-resident data format into a tape-resident format. The partitions of tape-formatted data are migrated to tape in a controlled order. The location and defining predicates of each tape resident partition is recorded in the control database.

For an experiment, we migrated a 60 Gbyte data set to three tapes (the data set reduced to 45 Gbytes in the binary tape data format). The tape resident files were about 500 Mbytes each. The data set was "striped" across three tapes, in stripe units of about 1 Gbyte (every tuple was written on exactly one tape). The DLT 4000 achieved a 2 to 1 compression ratio on this data, and a (read) transfer rate of 1.9 Mbyte/sec.

The data set that we migrated to tape contains network operations data. Our test query collects all tuples related to a particular connection, and reports an aggregate function of them (this is a typical query). In the preliminary implementation, a query is submitted by specifying a data set and a time range (which determine the list of query units to be processed), the aggregation server executable, and a back-end executable. The aggregation server executable is identical to the one we use for queries on disk-resident data. The back-end executable is a simple single-threaded program that hashes tuples from tape to the aggregation servers.

We submitted a query using three back-end executables and five aggregation server executables. The query required 176 minutes to run, implying a processing rate of 4.4 Mbytes/sec. This processing rate is about 77% of the 5.7 Mbytes/sec that we would obtain if all three drives were transferring data at their maximum rate. An investigation showed that at most times, the back-end servers were transferring data at their maximum rate (the DLT 4000 is very forgiving of short delays in requesting the next data block [7]). Further, the time to mount all the tapes was small (a few minutes), and the seek time between files was negligible (as we accessed the files in their order on tape). However, the aggregation servers pause at the end of processing each partition to finish the processing associate with the partition and thereby free up memory. These pauses accounted for the bulk of the slowdown. We modified the aggregation server to reduce the lengths of the pauses, and obtained a processing rate of 5.2 Mbytes/sec, which is 91% of the maximum processing rate.

We note that the issues discussed in Section 3 were necessary for obtaining good performance. We did not have 45 Gbytes of temporary storage available, so we could not have processed the query by first loading the data to disk. The query made a single sequential pass through the data, so loading the data to disk would not have improved performance in any case. To obtain a moderate data rate, we needed to make use of parallel I/O and parallel processing. We expect to obtain a high data rate by using parallel I/O with the higher performance DLT 7000 drives. Finally, we note that although the interface to the data warehouse is very simple, it is easy to submit a wide range of queries. The back-end server and the aggregation server are generic, changing the query can be accomplished by changing an aggregation object and relinking.

7 Related Work

The database research community has recently investigated the integration of tertiary storage with a database management system. The drivers of this research include scientific databases [8, 18], multimedia [19, 2], and digital libraries [9, 3].

Several works have investigated methods for implementing general SQL databases with tertiary storage as the lowest layer in the memory hierarchy. Moran and Zak [10] describe an experimental integration of Oracle with the AMASS HSM managing an optical jukebox. Sarawagi [16, 15] discusses the architecture of *Sequoia 2000* for handling queries on tape-resident data. Tape-resident tables are partitioned into contiguous pieces, and a query on a table is transformed into a sequence of queries on the partitions. This architecture permits arbitrary joins, which are implemented by joining a cross product of partitions on the joined tables. Myllymaki and Livny [12, 13, 11] compare alternative algorithms for joining two tape-resident tables, again taking a cross product of the tables.

Much of the work on integrating tertiary storage with database systems is motivated by scientific database applications. Many of these (e.g., EOSDIS [8]) make extensive use of "large objects", for example tiles of satellite images. Issac [5] proposes an architecture in which a database that references file-resident large objects can be interfaced to a HSM, greatly simplifying the retrieval of data. DeWitt [22, 21] refines this idea in the Paradise DBMS.

8 Conclusions

The pervasive computerization of an organization's activities permits the large scale and fine grained collection of data related to vital operations. By loading this data into a data warehouse, operations can be analyzed and optimized. While the cost of on-line storage has dropped dramatically in recent years, the volume of data collected still exceeds on-line capacity by an order of magnitude in many application domains. In this paper, we propose an architecture by which a queries on a data warehouse can be easily and efficiently implemented. We base our architecture on the observation that most decision support type queries work well when the fact table is sequentially scanned. Thus most queries can be answered by reading data directly from tape.

We implemented an initial prototype of the tape-resident data warehouse. Though the architecture of the prototype is simple, we achieved good efficiency in reading from multiple tapes, and used a aggregation server that we developed for queries on disk-resident data. The good efficiency was because the the prototype has the characteristics listed in Section 3 (except for indexing).

Our future work on tape-based data warehousing includes:

- Research into system support issues, such as indexing, scheduling, and data layout.
- Language design for ad-hoc queries.
- Algorithm design and query optimization.
- Integration with data mining tools.

Acknowledgements

We'd like to thank the Session Chair, Rodney Van Meter, for his comments on an initial draft of this paper, and Damianos Chantziantoniou for his discussions of decision support query languages.

References

[1] D. Chatziantoniou and K. Ross. Querying multiple features of groups in relational databases. In *Proc. 22nd Very Large Data Base Conf.*, 1996.

[2] S. Christodoulakis. Multimedia databases. In *Intl. Conf. on Very Large Data Bases*, 1997. tutorial.

[3] R. Coyne and H. Hulen. Toward a digital library strategy for a national information infrastructure. In *Proc. 3rd NASA Godddard Conf. on Mass Storage Systems and Technologies*, pages 15–18, 1993.

[4] R. Grossman and X. Qin. Ptool: A low overhead, scalable object manager. In *Proc. SIGMOD 94*, 1994.

[5] D. Issac. Hierarchical storage management for relational databases. In *Proc. 12th Symp. on Mass Storage Systems*, pages 139–144, 1993.

[6] T. Johnson. Coarse indices for a tape-based data warehouse. In *Int'l Conf. on Data Engineering*, 1998.

[7] T. Johnson and E. Miller. Performance measurements of robotic storage libraries. In *Proc. IEEE Conf. on Mass Storage Systems / NASA Goddard Conf. on Mass Storage Systems and Technologies*, 1998.

[8] B. Kobler, J. Berbert, P. Caulk, and P. Hariharan. Architecture and design of storage and data management for the nasa earth observing system data and information system (eosdis). In *Proc. 14th IEEE Mass Storage Systems Symp.*, pages 65–78, 1995.

[9] A. Kraiss and G. Weikum. Vertical data migration in large near-line document archives based on markov chain predictions. In *Proc. 23rd Very Large Database Conf.*, pages 246 – 255, 1997.

[10] S. Moran and V. Zak. Incorporating Oracle on-line space management with lon-term archival technology. In *Proc. 5th NASA Goddard Conf. on Mass Storage Systems and Technologies*, pages 209–228, 1996.

[11] J. Myllymaki and M. Livny. Disk-tape joins: Synchronizing disk and tape access. In *ACM SIGMETRICS*, 1995.

[12] J. Myllymaki and M. Livny. Efficient buffering for concurrent disk and tape I/O. *Performance Evaluation*, 27:453 – 471, 1996.

[13] J. Myllymaki and M. Livny. Relational joins for data on tertiary storage. In *Proc. Intl. Conf. on Data Engineering*, 1997.

[14] P. O' Neil and D. Quass. Improved query performance with variant indices. In *Proc. ACM SIGMOD*, 1997.

[15] M. S. S. Sarawagi. Reordering query execution in tertiary memory databases. In *Proc. 22st Very Large Database Conference*, 1996.

[16] S. Sarawagi. Query processing in tertiary memory databases. In *Proc. 21st Very Large Database Conference*, pages 585 – 596, 1995.

[17] D. Schneider. The ins and outs (and everything inbetween) of data warehousing. In *Proc. 23rd Intl. Conf. on Very Large Data Bases*, pages 1–32, 1997. in Tutorials.

[18] M. Stonebraker. Sequoia 2000: A next-generation information system for the study of global change. In *Proc. 13th IEEE Symp. on Mass Storage Systems*, pages 47–53, 1994.

[19] P. Triantafillou and T. Papadakis. On-demand data elevation in a hierarchical multimedia storage server. In *Proc. 23rd Very Large Database Conf.*, pages 226–235, 1997.

[20] P. Valduriez. Join indices. *ACM Trans. on Database Systems*, 12(2):218–246, 1987.

[21] J. Yu and D. DeWitt. Processing satellite images on tertiary storage: A study of the impact of tile size on performance. In *Proc. 5th NASA Goddard Conf. on Mass Storage Systems and Technologies*, pages 460–476, 1996.

[22] J. Yu and D. DeWitt. Query pre-execution and batching in paradise: A two-pronged approach to the efficient processing of queries on tape-resident data sets. Technical report, University of Wisconson, Madison, 1996. Available at http://www.cs.wisc.edu/ jiebing/tape.ps.