StorHouse Metanoia[1] -

New Applications for Database, Storage & Data Warehousing

Felipe Cariño Jr., Pekka Kostamaa, Art Kaufmann & John Burgess
FileTek, Inc.
360 N. Sepulveda Blvd. Suite 1080
El Segundo, California 90245
{FCARINO, PKOSTAMAA, AKAUFMANN, JGB} @FileTek.com

Abstract

This paper describes the StorHouse/Relational Manager (RM) database system that uses and exploits an *active storage hierarchy.* By active storage hierarchy, we mean that StorHouse/RM executes SQL queries *directly* against data stored on all hierarchical storage (i.e. disk, optical, and tape) without post processing a file or a DBA having to manage a data set. We describe and analyze StorHouse/RM features and internals. We also describe how StorHouse/RM differs from traditional HSM (Hierarchical Storage Management) systems. For commercial applications we describe an evolution to the Data Warehouse concept, called *Atomic Data Store*, whereby atomic data is stored in the database system. Atomic data is defined as storing *all* the historic data values and executing queries against them. We also describe a *Hub-and-Spoke Data Warehouse architecture,* which is used to feed or fuel data into Data Marts.

Furthermore, we provide analysis how StorHouse/RM can be *federated* with DB2, Oracle and Microsoft SQL Server 7 (SS7) and thus provide these databases an active storage hierarchy (i.e. tape). We then show two federated data modeling techniques (a) logical horizontal partitioning (LHP) of tuples and (b) logical vertical partitioning (LVP) of columns to demonstrate our database extension capabilities. We conclude with a TPC-like performance analysis of data stored on tape and disk.

1.0 Intorduction

This paper's main goal is to establish, describe and analyze the need for an active storage hierarchy in database systems and applications. Commercial Database Management Systems (DBMS) have evolved to support a diverse range of applications. DBMSs have been based on hierarchical, network, relational, object-oriented and the emerging object/relational database models. With few exceptions, these database systems and applications use disk media as their primary storage. Hierarchical Storage Management (HSM) is used by some of these applications to exploit the benefits of cost-effective optical and tape storage systems. However, as discussed later in this paper, these systems have limitations that make their use impractical for actively retrieved data by multiple applications. We propose and discuss the idea that database systems may use a complete active storage hierarchy (i.e. tape, optical, and disk). The key proposal is that active data be stored, queried and analyzed on tape farm libraries and optical jukeboxes, as well as magnetic disk.

[1] Metanoia – process by means of which one changes one's mind; of a conversion of perception or thinking, so as to attain a fundamental transformation of the mind.

In this paper, we use the term active storage hierarchy when (say, SQL) queries execute against data stored on diverse media. This should not be confused with the research term “active disks”.

At VLDB 1998, during the 10-year best paper award [GB88], Jim Gray described emerging disk technology trends. A key disk technology trend is more storage and CPU-like processing capabilities [RGF98]. Applications that require massive amounts of storage often cannot be served by magnetic disk based solutions. They need removable media storage and system management software that can scale far beyond magnetic disk based solutions and does so at a fraction of the costs.

The following table (see Figure 1) shows the relative costs of removable storage (including library, drives and media) versus RAID magnetic disk storage: Typically, at any time, an application may need to access any of the stored data. Therefore, accessibility and availability are critical. Removable media has become also the primary basis for data archives. Typical active storage systems also serve the purpose of being the archive. Hence, permanence is critical.

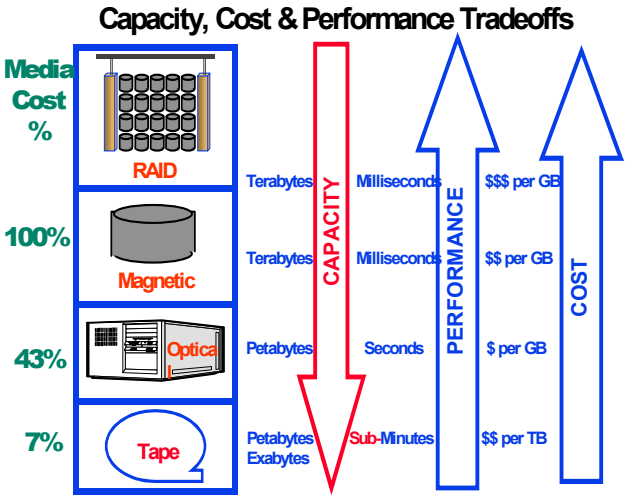

Figure 1: Cost, Capacity & Performance Tradeoffs

Removable media technology gains its cost advantage through the use of automated storage libraries, which can be scaled for little cost (sheet metal and media). There are, of course, many performance issues with the storage drives, media and libraries used in these systems. These are best addressed by discussing the state of technology in each area. The focus will be on those components available to open systems.

A VLDB 99 panel [CBOSJ99] discussed the storage and data trends and the need to balance storage cost, performance and database’s need for capacity.

This paper builds on the main trends (Figure 2) that disk technology is improving at a 35% CAGR, but data is growing at 100% CAGR. Furthermore, the *current* economics (see Figure 1 percentages) of the storage media are that storing data on tape is approximately 7% the cost of storing the data on disk and the cost for storing data on optical is about 42% of storing the data on disk. Disk-only solutions result in a stealth cost-barrier to complete application development or present an artificial limit to what and how much data to keep online. Thus, applications do not store all the historical data and/or large object (LOB) values needed to efficiently run an enterprise.

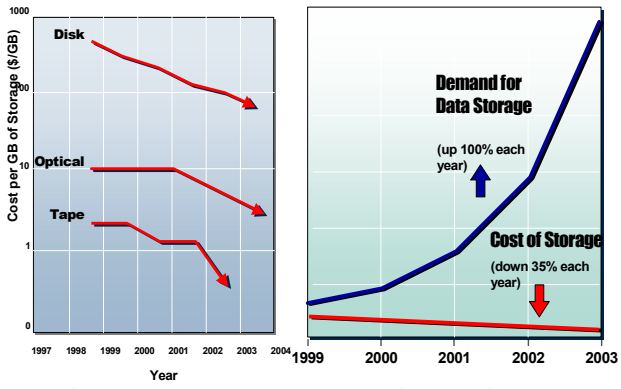

Figure 2: Storage Cost & Usage Trend

StorHouse/Relational Manager (RM) [CB00] is a commercial relational database system that supports SQL queries for data stored in an active storage hierarchy. The StorHouse/RM storage hierarchy includes RAID, Optical Jukeboxes and Tape Farm libraries. Again, we use the term *active* storage hierarchy because StorHouse/RM *directly* stores and fetches data from a storage hierarchy (e.g. Tape Farm Library).

StorHouse/RM (Figure 3) was designed and optimized to store atomic or historical data on diverse media. StorHouse/RM works in conjunction with StorHouse/SM, which specifically administers the storage, access and movement of relational data.

StorHouse/SM, FileTek's comprehensive data management software, controls a hierarchy of storage devices comprised of cache, redundant array of independent disk (RAID), erasable and write-once-read-many (WORM) optical disk jukeboxes, and automated tape libraries. StorHouse/SM is also responsible for critical system management tasks, like data migration, backup, and recovery. StorHouse/SM provides system-managed storage that optimizes media usage, response time, and storage costs for each application. StorHouse/SM runs on Sun® Microsystems Ultra Enterprise Servers and comes standard with all StorHouse systems. For completeness, we mention StorHouse.com [CB01], a new DMSP (Data Management Service Provider) that uses StorHouse/RM's unique storage and database system to store historical data for customers (i.e. outsource data to our DMSP service).

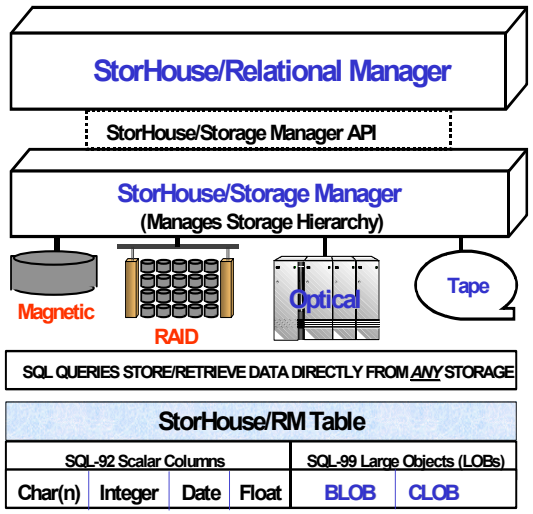

Figure 3: StorHouse/RM Architecture

SQL access is available from different platforms through a variety of industry-standard protocols. StorHouse/RM greatly differs from Hierarchical Storage Management (HSM) systems as shown in Figure 4 in that all data can be accessed on-line at anytime using SQL and *without DBA intervention* to reload any data. We summarize below the major differences between the StorHouse/RM and HSM approach to managing and retrieving data from hierarchical storage.

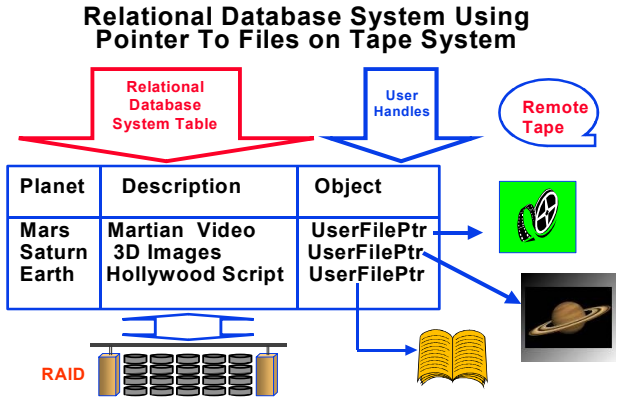

Figure 4: HSM and/or File Pointer Approach

StorHouse/RM Versus HSM Solutions

StorHouse/RM has a direct data store and SQL query retrieval of data stored on heterogeneous storage devices, like tape farm libraries and optical jukeboxes. It handles database duplexing, recovery and internal pointers to data values (not files). StorHouse/RM provides ANSI SQL Large Object (LOB) and LOCATOR capability to LOB columns. A central database store can be used to support multiple database applications (see Section 4.0 analysis). In most cases, this can be done without modifying the current application (i.e. transparent access to data from an existing application).

HSM Solutions

HSM software solutions vary from vendor to vendor. Our discussion applies to most, if not all, of the HSM products on the market. In general, HSM software places a marker in the database tuples(s). A query may typically wait for the data, while usually an entire partition is found and restored. HSM software may require that database values be restored and deleted. Sometimes this labor-intensive process reduces the use of historical archived data. Most HSM systems return a pointer to a file and not a specific tuple value; thus, the receiving application may have to perform post-processing of the file to get the desired tuple value.

We suggest that, like any other database operations, database management system should place, duplex and retrieve tuple values (automatically) via SQL (without DBA intervention).
For example, an application using LOBs may create a pointer for LOB values it stores in files. When compared to StorHouse/RM, where ANSI SQL LOB tuple/column values are stored and retrieved directly without any DBA intervention or application pre/post-processing of any kind.
This paper is organized as follows: Section 2.0 analyzes the StorHouse/RM database system internals that support the diverse storage systems. Section 3.0 describes an evolution to Data Warehouse notions and supporting architectures. Section 4.0 describes two database-modeling techniques that federate StorHouse/RM to other RDBMS systems to address OLTP/DSS and large database applications. We describe a scenario where we can integrate diverse database islands-of-automation. We conclude in Section 5 with a performance analysis.

2.0 StorHouse/Relational Manager (RM)

StorHouse/RM is a database system that supports SQL queries against the storage hierarchy. This database system was designed and optimized to store (atomic) data on diverse media. StorHouse/RM (RM for short) works in conjunction with StorHouse/SM (SM for short) to specifically administer the storage, access, and movement of relational data. An in-depth technical analysis can be found in [CB00]. SQL access is available from different platforms through a variety of industry-standard protocols. An RM database consists of the following:

- User table data that you store and access.
- Optional indexes—value, hash, and range—that locate the table data.
- Metadata that describes database components.

RM databases have both a logical (Figure 5) and a physical structure. Logically, user tables and associated indexes reside in user tablespaces, and metadata resides in a system tablespace. Physically, user tables, indexes, and metadata are stored in SM files. RM user tables consist of one or more table segments. Each table segment is a separate SM file. Whether a user table is composed of one or multiple segments depends on the size of the user table and whether you later load more data into the table.

Extents

Table and index segments are RM files that can reside on any storage device in the RM storage hierarchy. These files are composed of different extents.

- A Data extent holds user data and/or control data.
- A Definitions (DF) extent contains information necessary to retrieve the data.
- A Map extent is the high-level index that RM always reads first when doing index lookups.

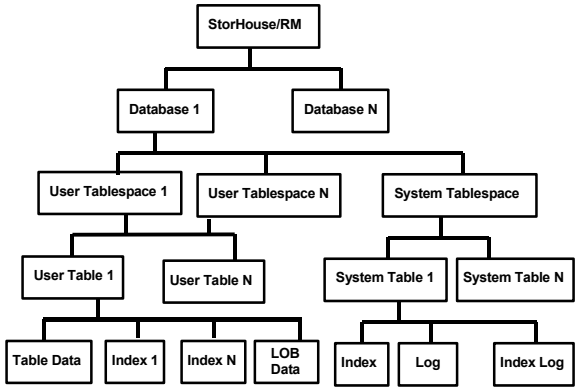

Figure 5: StorHouse/RM Internal Architecture

You can retain some or all extents in the magnetic disk performance buffer to enhance performance. For instance, you can hold the value index and hash index DF and Map extents on the performance buffer longer than the table Data extent to speed access to the data.
Value and hash indexes also consist of index segments. Each index segment is a separate SM file. For each value index on a table, there is always one value index segment associated with each table segment. For each hash index on a table, there is always one hash index segment associated with each table segment. If you load more data into a table, RM will create a new table segment and corresponding value and hash index segments. Range indexes do not consist of index segments; they are stored in the metadata.

UNIX Database Files

RM database metadata and range indexes reside on and are managed by RM on the UNIX file system. Each system table, system table index, system table log, system table index log, and range index is a separate UNIX file.

User Tables

A user table (Figure 6) is the basic unit of data storage in a RM database. User tables hold user-accessible data. Logically, RM user tables are like most RDBMS user tables; they consist of columns and rows of data. Each row contains data values conforming to the constraints of the columns that make up the row. Physically, an RM user table is one or more files on the RM storage hierarchy. User tables can reside on any storage device in the hierarchy. The user tablespace defines the target storage device and the migration path through the storage hierarchy. For instance, you can store time critical data on RAID and then migrate that data to tape or optical as the data ages. The RM software automatically manages storage and migration based on your user tablespace parameters.

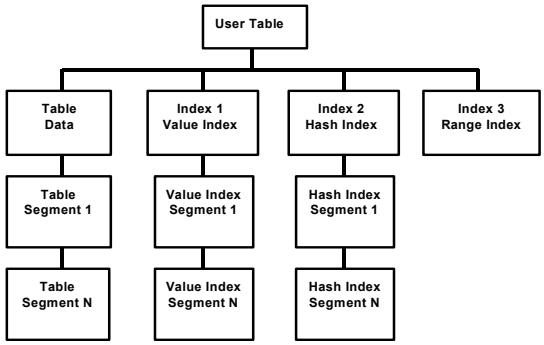

Figure 6: User Table Organization

Indexes

Indexes provide efficient access to table data. You can create an index on a column or combination of columns in a user table. An index based on one column is a simple index. An index based on more than one column is a compound index. RM supports three index types—value, hash, and range.

Value index

Value indexes work best with queries that return multiple rows based on a range of values. A value index contains an ascending list of all the values in a column (or group of columns for a compound value index). For each column value, the index contains an index map to the table row containing that value. By searching the index rather than the table, then matching column values to row Ids, RM can more efficiently find requested table rows.

Hash index

Hash indexes work best with queries that return a specific record based on a specific value. A hash index is a two-part index based on an index map extent and a hit list that uses a proprietary RM algorithm to effectively locate individual table rows based on individual index values.

Range index

Range indexes are useful for user tables with multiple segments. A range index contains the lowest and highest column data values for each table segment for a user table. Instead of searching through multiple table segments, RM first looks at the range index to find the specific table segment with the requested data values. Then, RM might use any hash or value indexes to find a specific data value or range of values in the table segment.

Each time you load new data into a table, RM enters the low and high data values into the range index. If there is only one table segment, then there is only one low and high value pair in the range index. If there are 100 table segments, then there are 100 low and high value pairs in the range index.

User tablespaces

A user tablespace defines where table segments, hash index segments, and value index segments are stored on the RM storage hierarchy. It also sets attributes that influence storage management like backup and migration. When you create a user table, you assign it to a user tablespace. The table's segments and corresponding index segments then are stored according to the specifications of the user tablespace.

Allocating Storage

You allocate storage by identifying the volume sets and file sets for all table, hash index, and value index segments stored in the user tablespace. Whether you use multiple volume sets and file sets in a user tablespace depends on your data and your access and performance requirements.

Volume

A volume is a unit of media on which data can be recorded and read. RAID, magnetic disk, erasable and WORM optical disk and DLT cartridges are all examples of RM volumes, or media.

A **volume set (VSET)** is one or more physical volumes that are treated as a logical unit of storage. You use volume sets to control the physical grouping of files. A user tablespace identifies the volume sets that will contain the table, hash index, and value index segments for all tables and indexes in that tablespace. You can store table segments and associated hash and value index segments on the same volume set or on different volume sets.

A **file set (FSET)** is an area of storage within a volume set. Files are stored in file sets. Table segments and index segments are files. You can store table and associated hash and value index segments in the same file set or in different file sets. For example, if you want to minimize volume mounts and don't need to manage the storage of table data and indexes in different ways, then you could allocate one VSET with one FSET for all components. Or if you want to manage the storage of table data and indexes in different ways but remove them from RM at the same time, you could allocate separate FSETs in one VSET. Or if you want to manage the storage of table data and indexes in different ways and don't need to remove them from RM at the same time, you could allocate separate VSETs.

Performance and Backup Copies

A user tablespace contains a **Vulnerability Time Factor (VTF)** attribute that determines whether and when you create performance copies and/or backup copies of user table and index data. Performance copies reside on the RM magnetic disk performance buffer. Backup copies reside in primary file sets on the designated RM media.

A user tablespace contains an **Access Time Factor (ATF)** that works with RM migrate function to keep data that is most likely to be accessed in the RM performance buffer while maintaining a supply of free space. In a user tablespace, you can assign the same or different ATF values for table, hash index, and value index segments. This means you can retain index segments on the performance buffer longer than the data for faster query processing.

StorHouse/Storage Manager (SM)

StorHouse/SM, FileTek's comprehensive HSM software, controls a hierarchy of storage devices comprised of cache, redundant array of independent disk (RAID), erasable and write-once-read-many (WORM) optical disk jukeboxes, and automated tape libraries. StorHouse/SM is also responsible for critical system management tasks, like data migration, backup, and recovery. StorHouse/SM provides system-managed storage that optimizes media usage, response time, and storage costs for each application.

We provide a performance analysis of StorHouse/RM using TPC-like benchmark queries in section 5.0. The next few section describes how StorHouse/RM can be federated to provide active storage to other database systems.

3.0 Data Warehousing

Relational database systems, like DB2, Oracle and SS7, are used to store and manage data in a Data Warehouse, Data Mining and/or operational enterprise systems. This section deals with the Data Warehouse, Data Mart and database architecture options for organizing and accessing large volumes of historic atomic and SQL LOB (Large Object) data. The key to *successful* Enterprise-wide DW is to store and mine all detail data and convert it into strategic information [Arm98] and [CK92].

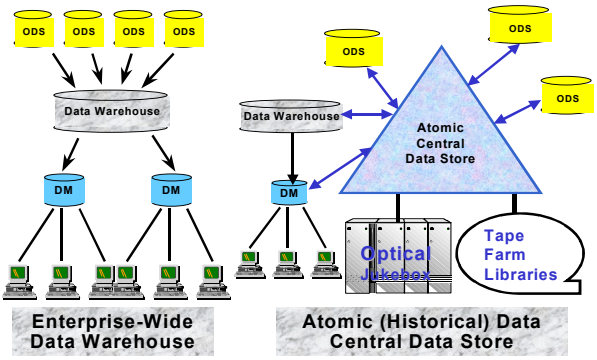

Figure 7: Atomic Data Store DW Concept

We propose a new DW concept evolution, called Atomic Data Store (figure 7) where *all* the detailed (atomic) historical (and archival) data is stored and used for information mining, decision support (DSS), customer relationship management (CRM) or to support other operational requirements.

The new enabling factor is cost-effective storage hence all detail data can be directly queried and/or used to fuel/load data into operational systems (e.g. using predictive triggers) or a DM/DW (e.g. using data summaries or frequent data access).

The economics of magnetic disk storage, however, force users to decide what data to maintain online and what to migrate out; migrated data is frequently replaced online with less useful summaries. This creates a "*Stealth Storage Cost-Barrier*" whereby application developers limit the amount of disk-based online data.

It is clear that the majority of data does not need to be maintained on costly high-speed media. Data Warehouse systems that use disk storage as the only active storage (i.e. where queries are directed to) artificially limit the Enterprise-Wide DW and/or may not integrate diverse Data Marts or Data Warehouse applications throughout the enterprise and we emphasize the use of historical data.

Figure 8 shows a Hub-and-Spoke Architecture drill down to the central ADS concept, which can be used by operational data stores to store all their atomic/historical data into a central DW (with cost-effective active storage). Then, Data Marts can be used to feed, cleanse and aggregate the data (e.g. also solves island-of-automation data problem).

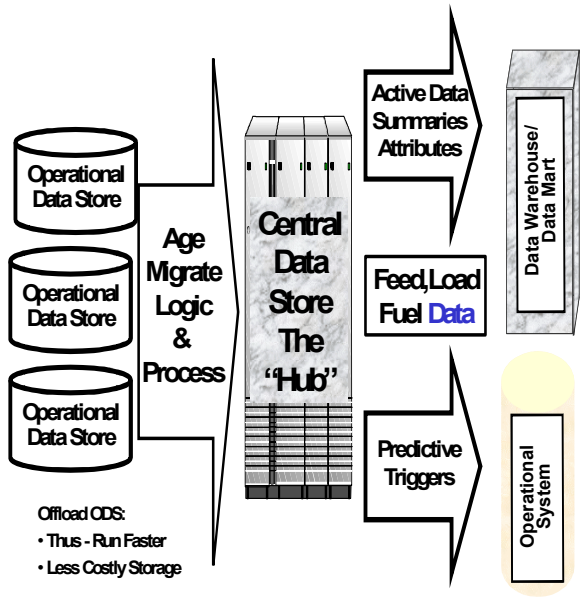

Figure 8: ADS in Hub-and-Spoke Architecture

4.0 Federated Databases & Database Modeling

Federated Databases [Car86] can provide a "single system image" for diverse data sources. These products integrate the database's data sources and provide uniform access to the data. Furthermore, it is important to keep in mind that sizes of database applications will grow to petabytes and even to yottabyte sizes [CKK00].

This section describes two database-modeling techniques that use federation to bring hierarchical storage to an RDBMS. StorHouse/RM can be transparently federated to Oracle, DB2 DataJoiner/UDB and SQL Server 7. Thus, some or all the data from these RDBMSs can be migrated and/or duplexed in StorHouse/RM. A key point is that the current application can then transparently access data from the RDBMS and/or the StorHouse storage hierarchy. This architecture also provides location transparency.

StorHouse/RM is meant to provide SQL access to large volumes of data that span multi-terabytes, petabytes and exabytes. For applications that require OLTP or DSS performance an RDBMS like Oracle or DB2 should be used. For these applications, StorHouse/RM can be federated as a data source, thus providing these RDBMS's new capabilities. The bottom-line is that with StorHouse/RM and federation, the challenges enterprises will face are converted to data modeling problems. Database designers must make "reasonable" trade-offs to balance storage, reliability, cost and total system performance. Our examples use StorHouse/RM as a data source to DB2 DataJoiner or UDB 7.1 to illustrate our two database modeling techniques.

4.1 Logical Horizontal Partitioning

Logical Horizontal Partitioning (LHP) – as shown in Figure 9 – is a horizontal partition of rows, whereby different rows from the same logical table are assigned to different data sources.

Users can transparently migrate rows to StorHouse/RM and use an SQL VIEW (Figure 10) to avoid having to modify existing applications. For example, use DataJoiner or UDB 7.1 federation to access data from different data sources, which includes StorHouse/RM, and hence hierarchical storage.

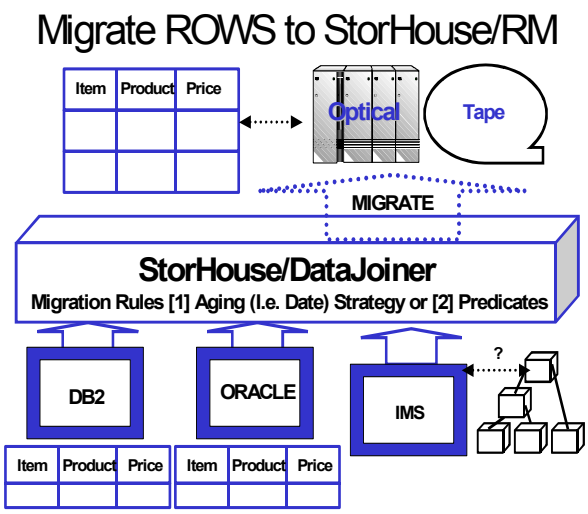

Figure 9: Logical Horizontal Partitioning

The common strategies for migrating rows from the RDBMS to StorHouse would be to:

a) "Age Out" tuples based on a date, say a 6th quarter of data when you keep five quarters on-line in the disk-based RDBMS; or 7 years of historical bank or call detail data (as required by law);

b) "Least frequently accessed" tuples to reduce disk-storage needs and making RDBMS run faster; and

c) "Complex predicate" where a predicate is used to migrate values to the active storage hierarchy.

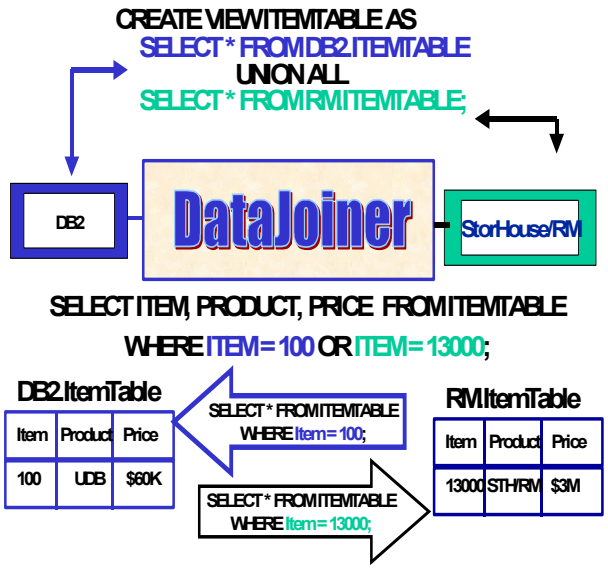

Figure10: Transparent Access - Using SQL VIEWs

4.2 Logical Vertical Partitioning (LVP)

Logical Vertical Partitioning (LVP) is a vertical partition of table columns across the RDBMS and StorHouse/RM. With LVP, different columns from the same (logical, federated) table may be assigned to different data sources (or storage media). Figure 11 shows an LVP data model where text and video columns are stored on tape or optical and scalar values are stored on disk. The examples show a StorHouse/RM with SQL-99 LOB (Large Object capabilities) to store video and supporting text as a relational table shown below. The example is simple, but illustrates the proposed solution(s).

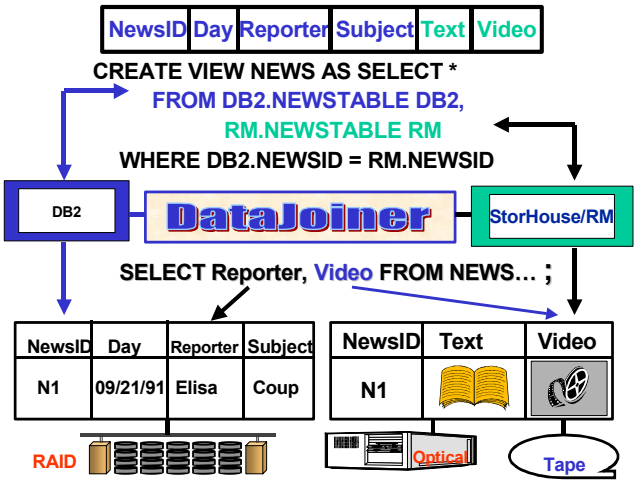

Figure 11: Logical Vertical Partitioning

Like data modeling for relational databases, there is a trade-off between performance, data replication and cost that each application must make. The LHP and LVP techniques above are new options available to developers of existing and new web applications. A future third technique will rely on the Automatic Summary Table (AST) capability being added to relational database systems, like DB2 UDB.

A critical part of the Data Warehouse is integrating all the information from diverse operational (i.e. Data Mart or Data Warehouse) systems.

Figure 12 shows that a common central data store used by heterogeneous databases, in this case Oracle and DB2, provides a mechanism for Oracle applications to access DB2 historical data and vice-versa.

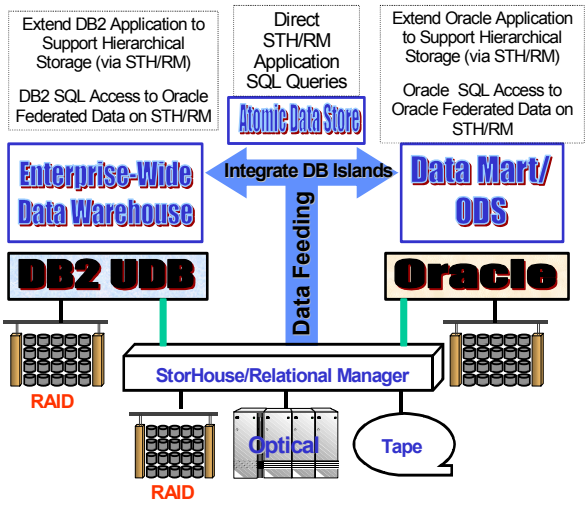

Figure 12: StorHouse – Extending RDBMS & Integrating Islands of Information

5.0 Performance

The first question most users ask – after (i) understanding the difference with HSM systems and (ii) storage costs – pertains to performance. We decided to run *TPC-**like*** benchmark queries since these are well understood. Since this is not an official TPC-H benchmark, we show only some of the representative queries. Its purpose is to show relative performance numbers with respect to using two different storage types namely disk and tape. We used our development test system and purposely did not tune the system for optimal performance, since again our goal was to show some numbers for tape and disk storage.

The environment consists of SUN 3550 with dual 400MHz UltraSPARC processors, 1GB of memory, a Sun StorEdge A1000 Disk Array with a total of 12 18GB 10K RPM drives, and a StorageTek TimberWolf 9738 tape library with 2 StorageTek 9840 tape drives. The software used is DB2 DataJoiner version 2.1, StorHouse/RM version 3.0, running on Sun Solaris 2.6. The DataJoiner data is loaded on the Disk Array, and the StorHouse/RM data is loaded on the tape library, with part of the disk array used as the performance buffer.

The data loaded is the standard TPC-H data as defined in [TPC-H]. The data size loaded is very small, since we are only interested in showing the basic performance impact of the StorHouse/RM solution. We picked the 0.05 scale factor to load. We loaded the data in three different configurations:

1. All tables on DB2/DJ.
2. All tables on StorHouse/RM and
3. About half of the large tables (Lineitem and Orders) on StorHouse/RM and the other half with the rest of the tables on DB2/DJ.

The split for the third option was based on completed orders. By the TPC-H benchmark definition, about half of the orders are completed. This horizontal partitioning of the data logically separates the orders into the current orders, which reside on DB2/DJ, and the historical orders, which reside on StorHouse/RM. Typically, more queries are placed on the current orders, and the historical orders are accessed for analysis purposes only.

The performance is measured against four configurations. The additional configuration from the three described above is a query running against only the current orders in the horizontally split federated configuration (number 3 above). This shows the added benefit of the federated approach of faster queries against only the current data. This can be compared against the DB2 DJ query time, which has to access all the data.

Query	DB2/DJ All Data	DB2/DJ Current Only	STH/DJ All Data	STH All Data
Q1	36.4	22.6	54.6	124.0
Q3	12.6	5.5	29.3	See text
Q6	6.7	5.0	11.4	7.0
Q8	39.3	16.2	43.7	See text
Q18	14.7	9.5	34.2	See text
SUM	109.7	58.8	173.2	-

The table above shows the impact on performance of moving a part of the data from an all disk-based system (DB2/DJ All Data column) to a federated system with a hierarchical storage (STH/DJ column). In addition, queries that only access the current part of the data are sped up, as shown in the (DB2/DJ Current Only column). The last column (STH All Data) is shown as a reference point, if all the data is placed on the StorHouse/RM system. Note that we are only showing values for two of the queries. This is because even though StorHouse/RM is capable of handling complex queries, it is not tuned to perform well. StorHouse/RM is not targeted as an analysis engine, but as a feeder (fueler) of data to such engines.

The queries were chosen to highlight the performance differences. Query 1 (Q1) accesses all of the LINEITEM data, and aggregates it. It does not do any joins. Query 3 (Q3) is interesting, since it accesses data from ORDERS table, some of which resides on DB2/DJ and some that resides on STH/RM. LINEITEM data needed for this query, however, resides all on DB2/DJ. Query 6 (Q6), like Q1, also accesses only the LINEITEM table, but only accesses one year of the seven years. The year that is chosen is in the historical part (StorHouse/RM) of the data. Query 8 (Q8) is one of the more complex queries in the benchmark. It is an eight-way join, including the two large tables ORDERS and LINEITEM. Query 18 (Q18) accesses the large tables ORDERS and LINEITEM also, and in addition joins them to the CUSTOMER table.

Based on these queries, the impact of adding the StorHouse/RM as a federated system to DB2/DJ increased the query run-time by 58% (173.2/109.7) for queries that need to access all the data. Queries that need to only access the current data (half of the total data), however, were sped up by 46% (58.8/109.7). The overall performance that a typical workload will see depends on the mix and frequency of the queries. Notice also that the access times of data from tape depend heavily on the type and amount of data on the hierarchical storage system.

Future Work

Our current challenge is to develop StorHouse/Object-Relational Manager (ORM). StorHouse/ORM (Figure 13) major challenges are to support SQL-99 User-Defined Types (UDTs) and User-Defined Functions (UDFs) on active hierarchical storage. We will have to address several technical issues, like how do you execute and optimize a UDF that retrieves data from tape storage media. We will also in the future run multimedia BLOB and CLOB benchmarks and document the benefits of storing LOB data in StorHouse/RM and/or using LVP (i.e. using DB2 for OLTP/DSS and StorHouse for storing LOB values).

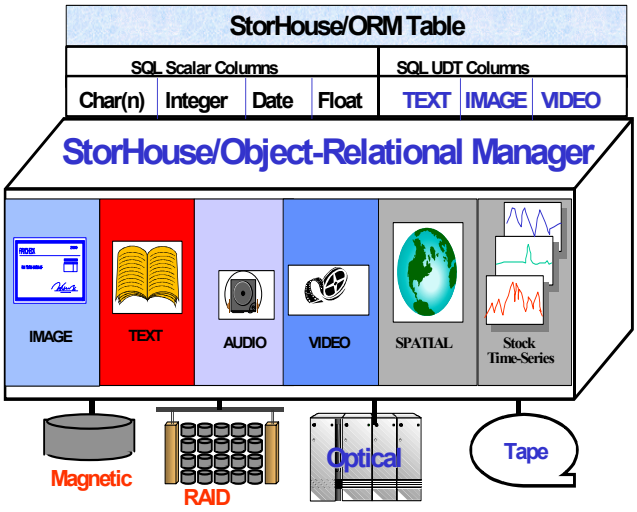

Figure 13: StorHouse/ORM - Future

Conclusion

We described storage and database trends and the need for hierarchical storage. We analyzed StorHouse/RM, an SQL relational database system that stores and retrieves data from a storage hierarchy (i.e. not just disk storage). We described the need for trade-offs between cost, performance and active storage hierarchy. We described the tradeoffs, uses and potential uses that tape, optical and disk storage can provide. We described how database systems use the storage hierarchy media. We also analyzed the StorHouse/Relational Manager, which is an SQL RDBMS that stores and retrieves data from the storage hierarchy using StorHouse/Storage Manager. We defined a new Data Warehouse concept, called Atomic Data Store, whereby applications exploit atomic (historical) data. We described a central data store Hub-and-Spoke architecture, used to feed data into Data Marts. We also showed some performance results of queries run against StorHouse/RM and federated solutions.

Acknowledgements

We want to thank the reviewers for their comments and Matt McBride for running the TPC-like benchmark documented in this paper.

References

[Arm98] Armstrong, B., "Data Warehousing: Dealing with Growing Pains", Data Engineering, March 1998, pp. 199 – 205.

[BZ98] Bontempo, C. and Zagelow, G., "The IBM Data Warehouse Architecture", CACM September 1998, Volume 41, No. 9, pp. 38 –48.

[Car86] Cariño, F. "HETERO – Heterogeneous DBMS Frontend", Proceedings Conference on Methods and Tools for Office Systems, Pisa, Italy, October 1986, pp. 159 – 172.

[CB00] Cariño, F. and Burgess, J., "StorHouse/Relational Manager (RM) – *Active* Storage Hierarchy Database System and Applications", Proceedings of 17th IEEE Mass Storage Systems Symposium, March 2000, College Park, Maryland, pp. 179 – 186.

[CB01] Cariño, F. and Burgess, J., "StorHouse.com – A DMSP (Database Management Service Provider)", To Appear in 18th IEEE Mass Storage Systems Symposium, April 2000, San Diego, California.

[CK92] Cariño, F. and Kostamaa, P., "Exegesis of DBC/1012 and P-90 – Industrial Supercomputer Database Machine", Proceedings of PARLE '92, Springer-Verlag, Paris, France, pp. 877 - 892.

[CKK00] Cariño, F., Kaufmann, A. and Kostamaa, P., "Are you ready for Yottabytes? StorHouse – Federated and Object/Relational Solution", 17th IEEE Mass Storage Systems Symposium, March 2000, College Park, Maryland, Vendor Presentation.

[CBOSJ99] Cariño, F., Burgess, J., O'Connell, W., Saltz, J. and Johnson, T., "Active Storage Hierarchy, Database Systems and Applications – Socratic Exegesis, VLDB 99, pp. 611 – 614.

[GB88] Gray, J. and Bitton, D., "Disk Shadowing", VLDB 88, pages 331 – 338.

[RGF98] Riedel, E., Gibson, G. and Faloutsos, C., "Active Storage For Large Scale Data Mining and Multimedia", VLDB '98, pages 62 – 73.

[TPC-H] Transaction Processing Performance Council (TPC), TPC Benchmark H (Decision Support), Standard Specification, Revision 1.2.1, 1999.