DATA BASE RESEARCH AT BERKELEY

Michael Stonebraker
Department of Electrical Engineering and Computer Science
University of California, Berkeley

1. INTRODUCTION

Data base research at Berkeley can be decomposed into four categories. First there is the POSTGRES project led by Michael Stonebraker, which is building a next-generation DBMS with novel object management, knowledge management and time travel capabilities. Second, Larry Rowe leads the PICASSO project which is constructing an advanced application development tool kit for use with POSTGRES and other DBMSs. Third, there is the XPRS project (eXtended Postgres on Raid and Sprite) which is led by four Berkeley faculty, Randy Katz, John Ousterhout, Dave Patterson and Michael Stonebraker. XPRS is exploring high performance I/O systems and their efficient utilization by operating systems and data managers. The last project is one aimed at improving the reliability of DBMS software by dealing more effectively with software errors that is also led by Michael Stonebraker.

In this short paper we summarize work on POSTGRES, PICASSO, and software reliability and report on the XPRS work that is relevant to data management. Moreover, the scope of this paper is limited to the Computer Science Division of the EECS Department. As such, research efforts at the Lawrence Berkeley Laboratory (LBL) and in the School of Business Administration are not discussed.

2. The POSTGRES PROJECT

The POSTGRES project is focused on building a next generation DBMS with novel object management, knowledge management and time travel capabilities. This system is now operational and consists of about 170,000 lines of code in the language C, and it runs on DECstation 3100s, SUN 3s, SUN 4s, and Sequent Symmetry machines. POSTGRES is available on tape for a nominal fee or it can be copied electronically from our Berkeley machine. Information on either distribution mechanism can be obtained from Claire Mosher, 521 Evans Hall, University of California, Berkeley, Ca. 94720.

In this section we report on projects oriented toward efficient object management, knowledge management and time travel support. Since POSTGRES is now mostly operational, the project is focusing on improving the performance of the prototype, finishing the implementation of promised function, and on integrating tertiary memory into the POSTGRES environment in a more flexible way.

General POSTGRES information can be obtained from the following references.

[MOSH90] Mosher, C. (ed.), "The POSTGRES Reference Manual, Version 2," University of California, Electronics Research Laboratory, Technical Report UCB/ERL M90-60, Berkeley, Ca., August 1990.

[STON86] Stonebraker M. and Rowe, L., "The Design of POSTGRES," Proc. 1986 ACM-SIGMOD Conference on Management of Data, Washington, D.C., June 1986.

[STON90] Stonebraker, M. et. al., "The Implementation of POSTGRES," IEEE Transactions on Knowledge and Data Engineering, March 1990.

2.1. Object Management

POSTGRES supports a general abstract data type (ADT) system whereby a user can add new data types on the fly as well as operators and functions for such data types. When an operator is defined on a data type, sufficient information is specified by the definer to allow the POSTGRES optimizer to choose heuristically optimal plans for queries which include such operators. This information is not recorded for functions, and the optimizer will not be able to optimize queries including functions. In addition, fixed and variable length arrays of abstract data types are allowed, and the query language, POSTQUEL has been extended with standard referencing into arrays. In addition, multiple inheritance of tables (classes) is supported, and functions that are defined for specific tables are inherited down the inheritance hierarchy in the standard way.

POSTGRES also supports complex objects by allowing a field of a table (class) to contain a procedural (executable) object. The value of the field is then the result of the procedure execution. We have spent considerable time researching the optimization of procedural

fields. We have investigated algorithms ranging from executing the procedure in full when needed (complete materialization) to query rewrite, whereby the definition of the procedure in POSTQUEL is substituted into the user query to produce a revised query for subsequent execution. In addition, we have investigated the *caching* of procedures prior to their evaluation being requested by a user. We have results both from simulation studies as well as from experiments on implementations of procedures in POSTGRES. Further details are available in the following references.

[JHIN88] Jhingran, A., "A Performance Study of Query Optimization Algorithms on a Data Base System Supporting Procedural Objects," Proc. 1988 VLDB Conference, Los Angeles, Ca., Sept. 1988.

[JHIN90] Jhingran, A., and Stonebraker, M., "Alternates in Complex Object Representation: A Performance Perspective," Proc. 1990 IEEE Data Engineering Conference, Los Angeles, Ca., Feb. 1990.

2.2. Knowledge Management

Extending a DBMS to include knowledge management support through use of *production rules* is one of the main goals of the POSTGRES project. Such a rules system allows complex integrity constraints to be supported as well as enabling certain expert systems to be implemented entirely within the DBMS kernal. Moreover sophisticated protection and view support can be easily enabled.

We have explored two different implementations for rules. The first one is a *query rewrite* system. In this approach, an incoming user query or update is modified by the rule system into one or more alternate queries and/or updates that guarantee that the rules currently being enforced are satisfied. Hence, support for rules occurs at a high level in the DBMS directly after parsing and prior to query optimization.

The major benefit of this implementation compared to an implementation deep in the DBMS kernal is the reduction in book-keeping compared to lower level support for rules. Also, we expect this system to outperform a lower level implementation when a small number of rules apply to a large number of tuples.

The second implementation we are exploring is a tuple level implementation. When an event (retrieval, insertion, deletion or update) happens to a tuple, a special module, the *rule manager*, checks for all applicable rules and activates them. We use *rule locks* and *rule stub records* to efficiently retrieve all the rules that affect a given tuple. This rule system should be very efficient when a large number of rules is present but each one affects only a small number of tuples.

Our knowledge management research has focused on several problems. First, we have constructed a general purpose algorithm which will support rewriting all POSTGRES commands. In addition, we have also focused on adapting the rules system to support more general view mechanisms than those currently available in commercial systems. Third, we have worked on making the rules system support *versions* of objects. Fourth, we are exploring the various rule semantics that can be implemented within either implementation. Moreover, we have both implementations running and are designing a benchmark to test performance of them under a variety of conditions. Sixth, we are investigating higher level rule notations that can be built on top of this rule system. Lastly, we are experimenting with various types of locks, of various granularities, and with different locking decisions and algorithms. As the best choice for all these parameters depends on the specific application, we also plan to design and implement a *rule lock optimizer* that will choose the most efficient locking scheme for a particular rule.

[STON90] Stonebraker, M. et. al., "On Rules, Procedures, Caching and Views in Data Base Systems," Proc. 1990 ACM-SIGMOD Conference on Management of Data, Atlantic City, N.J., May 1990.

[ONG90] Ong, L. and Goh, J., "A Unified Framework for Version Modeling using Production Rules in a Database System," University of California, Electronics Research Laboratory, Technical Report UCB/ERL M90-33, April 1990.

2.3. Time Travel

POSTGRES is a *no-overwrite* data manager that keeps the past history of the states of the data base and supports *time travel* queries on historical data. Therefore, this research focuses on several indexing techniques to facilitate time travel queries for historical data. In addition, many of the concepts extend to spatial access methods in general. The work consists of three main approaches: (1) *Segment Indexes*, which are useful for historical data as well as spatial data consisting of arbitrary multi-dimensional interval data; (2) *Lop-Sided Indexes*, which are multi-way tree-structured indexes that are not necessarily balanced to support non-uniform query distributions; (3) *Mixed-Media Indexes*, which are

useful for indexing large historical data relations that are maintained in a temporally partitioned storage architecture. Each of these subjects is briefly described below.

Segment Indexes are tree-structured indexes which store line segments in leaf and non-leaf nodes. In essence, the Segment Index concept is to store each segment in the highest level node such that the extent of the segment spans (covers) all of the intervals contained (represented by) its descendant nodes. Using this approach, long intervals are stored in higher level nodes, and short intervals in lower level nodes. Our research has focused on applying this idea to multiway tree structures such as B-trees and R-trees.

Lop-Sided Indexes are multi-way, tree-structured indexes which are not necessarily balanced, in order to support query distributions which are not uniform. Various methods of node splitting were developed to control the evolution of an index so as to maintain its desired degree of *lop-sidedness*.

Mixed-Media Indexes are indexes which span magnetic and optical disk media, in order to exploit the benefits of both, i.e., the high performance of magnetic disk access times and the low media cost and large capacity of optical disks. Several methods of migrating index nodes from magnetic to a write-once read-many (WORM) optical disk were explored in order to determine whether such mixed-media indexes may be competitive with other approaches that are maintained entirely either on magnetic or optical disk.

[KOLO89] Kolovson, C. and Stonebraker, M. "Indexing Techniques for Historical Databases," Proceedings of the 1989 IEEE Conference on Data Engineering, Los Angeles, Ca., February 1989.

[KOLO90] Kolovson, C. and Stonebraker, M. "Segment Indexes: Dynamic Indexing Techniques for Multi-Dimensional Interval Data," University of California, Electronics Research Laboratory, Technical Report UCB/ERL M90-55, Berkeley, Ca., August 1990.

3. THE PICASSO PROJECT

The PICASSO project consists of three related efforts. The first is focused on supporting persistence in programming languages, storing computer programs in data bases, and on providing easy to use interfaces to such program data bases. This group has also constructed an implementation of persistent CLOS objects though a shared object hierarchy. On top of persistent CLOS, the PICASSO programming environment has been constructed which facilitates building advanced data base applications. Lastly, a variety of Computer Integrated Manufacturing (CIM) applications have been built using this toolkit. We discuss these three research topics below.

PICASSO is now available for Unix systems including Sun-3's and Sparcstations, DECStation 3100's, and Sequent Symmetry machines and uses Franz Allegro Common Lisp. Information on acquiring the software should be directed to Claire Mosher, 521 Evans Hall, University of California, Berkeley, Ca. 94720. Further information on PICASSO is available in the following document.

[SCHA90] Schank, P. et. al., "PICASSO Reference Manual," University of California, Electronics Research Laboratory, Technical Report UCB/ERL M90-79, July 1990.

3.1. Data Base Interfaces for Programming Environments

We are implementing a shared object hierarchy for the PICASSO interface Programming system. It is an extension to the Common Lisp Object System (CLOS) that allows classes to be declared persistent and shared (*shared classes*). The definition of a shared class and all its instance objects are stored in a POSTGRES database or in a commercial relational DBMS. We have completed the implementation of a single-user application program cache that allows programs to reference private or shared objects using the same abstractions. Early experiments indicate that access time for shared objects in the cache is within 10% of the access time for private objects.

A typical multi-user environment will execute applications on workstations connected to a database server through a network. This architecture leads to the problem of managing distributed caches. We are currently experimenting with different concurrency control algorithms and distributed cache update protocols. A simulation study is being conducted to determine the relative performance of these different algorithms and protocols.

[ROWE87] Rowe, L., "A Shared Object Hierarchy," in "Object-Oriented Database Systems," (K. Dittrich, et. al. editors), Springer-Verlag 1987.

[ROWE89] Rowe, L., "Database Representation for Programs," Proceedings 1989 ACM

SIGMOD/SIGSOFT Workshop on Software CAD Databases, Napa, Ca, February 1989.

3.2. The PICASSO Programming Environment

The PICASSO programming environment is a graphical user interface development environment (GUIDE). The system consists of an application framework, an interface toolkit, and a visual WYSIWYG application builder/editor. The PICASSO system is written in the Common Lisp Object System (CLOS) extended with persistent objects as noted above and uses the X Window System.

Applications are composed of a collection of frames, dialogs, and panels. These objects, called *PICASSO objects (PO's)* are analogous to procedures in a conventional programming language because they can be called with parameters and they can define local variables. The PICASSO framework supports writing reusable interface components through libraries of parameterized PO's.

The PICASSO toolkit provides a wide variety of interface abstractions that can be used to create interesting graphical interfaces. In addition to buttons, radio buttons, check boxes, pull-down and pop-up menus, and scrolling text widgets, PICASSO provides color graphics images and drawings, scrolling tables that can contain heterogeneous data, video widgets that display full-motion video (from a tape or disk player) in real time, and hypermedia documents that connect all types of data together. The toolkit is extensible. Users can easily add new widgets, new looks for existing widgets, and new strategies for laying out widgets in an application.

It also contains a browser that allows a user to visualize the structure of a PICASSO application by displaying a "call graph" of the application. Heuristic algorithms are used to display this graph in a visually pleasing way.

[ROWE90] Rowe, L. et. al., "The PICASSO Application Framework," University of California, Electronics Research Laboratory, Technical Report, UCB/ERL M90-18, May 1990.

[KONS90] Konstan, J. and Rowe, L., "Developing a GUIDE using Object-Oriented Programming," University of California, Electronics Research Laboratory, Technical Report, UCB/ERL M98-82, October 1990.

[MESS90] Messinger, E. et. al., "A Divide-and-Conquer Algorithm for the Automatic Layout of Large Directed Graphs," IEEE Transactions of Systems, Man, and Cybernetics, Dec. 1990.

3.3. IC- CIM Data Base Applications

Our IC-CIM suite of applications has been built using PICASSO connected to a commercial relational DBMS. It consists of a facility manager tool, a process flow language and a hypermedia system.

The facility manager displays a 2D schematic view of an IC fabrication laboratory and allows users to access other facility and manufacturing information stored in the DBMS including equipment, utility, and lot information.

The process flow system allows a user to specify a complete representation of the operations to manufacture and test a semiconductor integrated circuit. Both a process flow language and an executor for this language have been constructed. Moreover, complex heuristics can be coded in the process flow language to automate actions traditionally performed by hand. Lastly, dynamic changes to active runs are supported by the executor. This system has been used to specify the CMOS process in use at Berkeley Microlab. Further experimentation and the inclusion of a version control system are planned.

In order to allow occasional users, such as facility managers and process engineers, easy access to the CIM data, we are designing a graphical facility management tool called CIMTOOL. One unique feature of CIMTOOL is that it provides an easy to use user-interface that allows free mixing of spatial, graphical and numeric data queries using a novel paradigm for query specification. We have recently extended CIMTOOL to include support for video and audio data, making a very flexible multimedia database browser. Multimedia data can be used to train personnel using video instruction and to integrate non-traditional data types, such as images, with more traditional information, such as yield data.

[HODG87] Hodges, D. and Rowe, L., "Information Management for CIM," Proc. Advanced Research in VLSI, Stanford University, March 1987.

[SEDA90] Sedayao, J. and and Rowe, L., "A Structured Editor for Process Specification," University of California, Electronics

Research Laboratory, Technical Report, UCB/ERL M90-48, April 1990.

[HEGA90] Hegarty, C. et. al., "The Berkeley Process-Flow Language WIP System," Proceedings 1990 Semiconductor Research Technical Conference, October, 1990.

4. The XPRS PROJECT

XPRS is focused on building a redundant array of inexpensive disks (RAID), file systems that run on top of a RAID, distributed RAIDs, and on parallelism that can be exploited by a data manager in a RAID environment. We report on these four areas in turn.

4.1. RAID

RAID (Redundant Arrays of Inexpensive Disks) has been proposed as a new way to organize disk systems to achieve high bandwidth, high I/O rates, and high availability for low capacity cost. The project has completed a first prototype system constructed from off the shelf components. The hardware consisted of a SUN 4/280 with 128 MBytes of RAM, 4 Jaguar dual SCSI string host bus adapters, and 32 Imprimis 5.25" synchronous SCSI disk drives. This first prototype is operational but suffers from bottlenecks in the I/O controllers as well as high CPU overhead.

Now we are hard at work on RAID II, which will include a custom designed I/O controller to alleviate the bottlenecks. The goal of RAID II is to support a peak transfer rate of 100 MB/s and a sustained transfer rate of 40 MB/s. A prototype system, based on 150 3.5" IBM Lightning disk drives, should be operational by the summer of 1991.

We have just begun thinking about how to integrate tertiary memory behind a disk array in a memory hierarchy. Our technology of choice is not optical disk, but rather helical scan tapes (8mm or 4mm). These have many of the same desirable features and drawbacks of small format disks: very high volumetric efficiency (e.g., a terabyte in a cubic foot or less) with low cost tape readers, but slow transfer rates (250 - 500 KBytes per second). We are investigating methods of implementing horizontal correction across multiple tapes (i.e., tape arrays) as well as robotic handling for staging tapes from shelves to readers. Another area of interest is to embed application specific compression into the I/O system to perform compression at the system level. The goal is to improve the severely limited bandwidth rather than increase an already generous tape capacity. The technical challenges are how to provide hints from applications to the I/O system so that the most effective compression techniques will be applied as well as how to build index and directory structures over variable length and compressed data.

[PATT88] Patterson, D. et. al., "A Case for Redundant Arrays of Inexpensive Disks (RAIDS)," Proc. 1988 ACM-SIGMOD Conference on Management of Data, Chicago, Ill., June 1988.

[CHEN90] Chen, P. et. al., "Performance Evaluation of a RAID on an Amdahl Mainframe," ACM Sigmetrics Conference, Boulder, Co, May 1990.

[KATZ89] Katz, R. et. al., "Disk System Architectures for High Performance Computing," Proceedings of the IEEE, December 1989.

4.2. RADD

RAIDs have the desirable property that they survive disk crashes and require only one extra disk for each group of G disks. Hence, the space cost of high availability is modest compared to traditional schemes which mirror each physical disk at a space cost of 100 percent.

The purpose of this research is to extend the RAID concept to a distributed computing system. We call the resulting construct, RADD (Redundant Array of Distributed Disks). RADDs can support redundant copies of data across a computer network at the same space cost as RAIDs do for local data. Such copies increase availability in the presence of both temporary and permanent failures (*disasters*) of single site computer systems as well as disk failures. As such, RADDs should be considered as a possible alternative to traditional multiple copy techniques. Moreover, RADDs are also candidate alternatives to high availability schemes such as *hot standbys*.

Our research has focused on identifying the key issues in the viability of RADDs. All RAID algorithms work directly in a distributed environment, however distribution generates its own significant problems. First, we must consider how to perform updates when the underlying network is not perfectly reliable. Furthermore, algorithms are needed to deal with an unequal amount of storage at each site. Next, we must be able to mix different group sizes on the same network as well as add or subtract disk storage from a RADD without requiring global reconstruction. Lastly, we have constructed a prototype RADD using our LAN based environment of DECstations 3100s, and we are in the process of fine-tuning our implementation. Our goal is to

boost performance as well as to experiment with different algorithms for parity updates generated in transaction processing.

[STON90] Stonebraker, M. and Schloss, G., "Distributed RAID: A New Multicopy Algorithm," Proc. 1990 IEEE Data Engineering Conference, Los Angeles, Ca., February, 1990.

4.3. File Systems and Transactions in a RAID Environment

This research focuses on file system allocation strategies, the performance of the file systems resulting from these strategies, and providing transaction support in and on such file systems. We are especially interested in taking advantage of the presence of an underlying disk array. The file system design space is divided into those file systems which allocate disk storage to optimize for writes and those which optimize for sequential reads. In the first category appear log-structured file systems which write dirty pages sequentially to disk blocks, regardless of the previous location of the block. Since blocks from multiple users may be interspersed on the disk, then logically sequential reads may not be physically a sequential. In the second category are extent-based and large-block based file systems which guarantee logically sequential reads are physically sequential.

We have constructed a simulator of several read-optimized and write-optimized file system allocation policies and have performed experiments on a wide range of workloads designed to represent transaction processing applications, supercomputer applications, and time sharing environments. Our simulations have shown that read-optimized file systems do not suffer poor disk utilization due to internal and external fragmentation. Moreover, extent-based file systems offer the best I/O performance over a wide variety of workloads. We expect to implement one or more of these strategies in a real file system and compare measured results with those from our simulation.

We have also studied the performance of transaction systems in a read-optimized and write-optimized environment. We have focused both on transaction management implemented inside the file system using physical locking and logging and on transaction support by DBMS software outside the file system using logical logging. Our simulations indicate that the transaction support inside a write-optimized file system appears to be performance competitive with a DBMS implementation, especially in workloads with small transactions. We expect to add transaction support to both kinds of file systems and compare their performance and ease of implementation. We expect to also compare actual performance with our earlier simulations.

[SELT90] Seltzer, M., Stonebraker, M., "Transaction Support in Read Optimized and Write Optimized File Systems," Proc. 1990 VLDB Conference, Brisbane, Australia, August, 1990.

[SELT91] Seltzer, M., Stonebraker, M., "Read Optimized File Systems: A Performance Evaluation", Proc. 1991 IEEE Data Engineering Conference, Kobe, Japan, April 1991.

4.4. Parallelism in XPRS

Our objectives for XPRS are to achieve near-linear query speedup in a shared memory multiprocessor computer system containing a RAID. In addition, we are focused on problems with load balance and intelligent management of resources in a multi-user environment. There are two specific problems that we are trying to solve: optimization and scheduling. The search space of sequential query processing plans is small enough that conventional query optimizers can perform an exhaustive search. However, it is too expensive to do an exhaustive search in the space of all the possible parallel query processing plans in a multi-user environment. Our strategy to overcome the complexity is to divide the optimization into two phases: the first phase is a conventional query optimization that finds a best sequential plan, and the second phase finds the best parallelization of the best sequential sequential plan. We fix a buffer size in the first phase and then dynamically adjust the plan in the second phase according to the current available buffer size.

The problem of scheduling is the contention of resources among the parallel queries. The main resources that we are considering are CPU cycles, disk bandwidth and main-memory buffer space. Too little parallelism will under-utilize the resources while too much parallelism will make the machine saturate and thrash. We are designing and evaluating scheduling algorithms that will choose the optimal degrees and optimal forms of parallelism.

In the past year, we have implemented the parallel query optimizer and executor for XPRS. Our initial performance results show that the system does achieve near-linear speedup. Our experiments also show that our two-phase optimization strategy does not compromise optimality of parallel plans in most cases. More studies on the scheduling problem are under way.

[HONG90] Hong, W. and Stonebraker, M., "Parallel Query Processing in XPRS," University of California, Electronics Research Laboratory, Technical Report, UCB/ERL M90-47, May 1990.

5. SOFTWARE ERRORS

Software errors (and not hardware failures) are becoming the dominant cause of data base unavailability. Therefore, we have focused on mechanisms to improve the availability of data in the presence of such errors.

Our approach has been to extend the operating system to allow a DBMS to write-protect (or guard) important shared data structures like the lock table, buffer pool, and buffer pool meta-data. Because DBMS software must explicitly unguard data when access is required, errors caused by uninitialized pointers or array bounds overruns will be prevented from propagating damage to these structures.

With a copy-on-write style of update, guarding limits error propagation as well. When a transaction first tries to write a record, the DBMS copies the record into a writable memory area. At the end of transaction, a system call copies the changes into write-protected memory. If a corrupted transaction detects its error before end of transaction, shared data is never unprotected and never modified. Deferring updates also restricts the number of modules which are permitted to unprotect memory and reduces system call overhead.

We are currently evaluating the performance and reliability of guarding. Initial performance measurements show guarding and deferred update cause two to four percent degradation when records are small (less than 1K bytes). For reliability estimation, we used failure data from commercial programs to develop a model of software errors. By seeding errors into POSTGRES according to the model, we can observe changes to the error detection and propagation rate due to guarding.

[SULL90] Sullivan, M., "Improving Software Fault Tolerance in Highly Available Systems," University of California, Electronics Research Laboratory, Technical Report, UCB/ERL M90-11, February 1990.