Database Research at Wisconsin

Introduction

This report briefly describes the main database research projects currently active at the University of Wisconsin-Madison. The emphasis here is on brevity rather than completeness; for more information on the projects mentioned below, please consult the papers listed, or contact the researchers involved in the project. Most of the technical reports listed are available by anonymous ftp from `ftp.cs.wisc.edu`; also, many of the papers listed as "submitted for publication" are expected to be available as technical reports by the time this report is published.

1 CORAL

The CORAL project was initiated in 88-89 (under the name
Conlog). The PI of the CORAL project is Raghu Ramakrishnan (`raghu@cs.wisc.edu`). CORAL is a database programming language that seeks to combine features of a database query language, such as efficient treatment of large relations and aggregate operations, with those of a logic programming language, such as more powerful inference capabilities and support for incomplete and structured data. CORAL significantly extends the expressiveness of standard database query languages such as SQL, and differs from logic programming languages such as Prolog in supporting a modular and non-operational semantics. CORAL is integrated with C++ by extending C++ with features to manipulate relations (defined, possibly, using CORAL rules).

This work is supported by a David and Lucile Packard Foundation Fellowship in Science and Engineering, an NSF Presidential Young Investigator Award, with matching grants from Digital Equipment Corporation, Tandem and Xerox, and NSF grant IRI-9011563. In addition to the authors of various papers cited below, the following people have contributed to the CORAL system at various times: Joseph Albert, Tom Ball, Lai-chong Chan, Sumeer Goyal, Vish Karra and Bob Netzer.

1.1 Declarative Programming in CORAL

This has been a main focus of the project. We have proposed semantics for extensions of logic programs with aggregates and negation, investigated control aspects, and developed several algorithms to support these novel features. We have also investigated the issue of memoing complex, non-ground structures. This research has been carried over into the implementation of the CORAL system. The range of declarative programming features in CORAL is arguably broader than that found in any other deductive database or logic programming system: the data structures supported include sets, multisets, non-ground terms, lists and arrays; the language extends Horn clauses with modular stratified negation and set-generation; and it is possible to tailor execution using several control annotations.

While the range of features is extensive, the most important difference with respect to other deductive/logic programming systems is an emphasis on providing users with a number of ways to control the execution. [1] The range of evaluation methods that are supported — and that can be selected by the user explicitly or by the system using defaults — goes from Prolog-style top-down backtracking to bottom-up fixpoint evaluation with a number of refinements. A simple module mechanism allows for a combination of programming styles; different execution strategies can be used in different modules. Extended C++ code can be used to define predicates, and relations computed through logical rules can be manipulated in extended C++ code, if this is desired.

CORAL also has an *explanation* system that allows users to graphically explore the *derivation trees* that underlie the system's inferences.

1.1.1 How Is CORAL Used?

The system can be used in one of two ways. First, it can be used by invoking the CORAL shell, by typing *coral* at the Unix prompt; this is similar in many respects to using the Prolog interface. In this mode, the user can create one or more *modules*, compile them, load them, execute them, etc. from the CORAL

[1] CORAL stands for COntrol, Relations And Logic.

prompt. CORAL can also be used essentially as a C++ extension. In this mode, as usual there is a main program and possibly several subprograms, some of which can be CORAL modules. The modules can be written using the declarative sublanguage or extended C++, and the main program can be written in extended C++.

1.1.2 Current Status

The current prototype of CORAL is optimized primarily for main-memory execution, but is interfaced with the EXODUS storage manager to provide access to data that is stored on disk. The CORAL system is in the public domain, and a copy of the software can be obtained by ftp from ftp.cs.wisc.edu. An object-oriented extension called CORAL++ is planned.

1.1.3 Recent papers from the CORAL project

Raghu Ramakrishnan, Per Bothner, Divesh Srivastava, and S. Sudarshan. CORAL: A database programming language. In Jan Chomicki, editor, *Proceedings of the NACLP '90 Workshop on Deductive Databases*, October 1990. Available as Report TR-CS-90-14, Department of Computing and Information Sciences, Kansas State University.

R. Ramakrishnan, D. Srivastava and S. Sudarshan. CORAL: A Database Programming Language, Proceedings of the International Conference on Very Large Data Bases, Vancouver, Canada, 1992.

R. Ramakrishnan, D. Srivastava, S. Sudarshan and P. Seshadri. Implementation of the CORAL Deductive Database System, Submitted for Publication.

D. Srivastava, R. Ramakrishnan, P. Seshadri, S. Sudarshan. CORAL++: Adding Object-Orientation to CORAL, Submitted for Publication.

S. Sudarshan and R. Ramakrishnan. Optimizations of Bottom-Up Evaluation with Non-Ground Terms, Submitted for Publication.

S. Sudarshan, D. Srivastava and R. Ramakrishnan. Extending Valid and Well-Founded Semantics for Programs With Aggregation, Submitted for Publication.

R. Ramakrishnan, D. Srivastava and S. Sudarshan. Rule Ordering in Bottom-Up Fixpoint Evaluation of Logic Programs, To appear, IEEE Transactions on Knowledge and Data Engineering, 1993.

R. Ramakrishnan, D. Srivastava and S. Sudarshan. Controlling the Search in Bottom-Up Evaluation, Proceedings of the Joint International Conference and Symposium on Logic Programming, Washington, D.C., 1992.

C. Beeri, R. Ramakrishnan, D. Srivastava and S. Sudarshan. Valid Model Semantics for Negation, Proceedings of the ACM Symposium on Principles of Database Systems, San Diego, CA, 1992.

R. Ramakrishnan, D. Srivastava and S. Sudarshan. Efficient Bottom-Up Evaluation of Logic Programs, In The State of the Art in Computer Systems and Software Engineering, Ed. P. DeWilde, 1992, Kluwer Academic Publishers, to appear.

R. Ramakrishnan and S. Sudarshan. Top-Down vs. Bottom-Up Revisited, Proceedings of the International Logic Programming Symposium, San Diego, 1991, pp. 321-337.

S. Sudarshan and R. Ramakrishnan. Aggregation and Relevance in Deductive Databases, Proceedings of the International Conference on Very Large Data Bases, Barcelona, Spain, 1991, pp. 501-511.

S. Sudarshan, D. Srivastava, R. Ramakrishnan and J.F. Naughton. Memory Management in the Bottom-Up Evaluation of Logic Programs, Proceedings of the ACM SIGMOD International Conference on Management of Data, 1991, pp. 68-77.

J.F. Naughton and R. Ramakrishnan. Bottom-Up Evaluation of Logic Programs, In Computational Logic: Essays in Honor of Alan Robinson, Ed. J-L. Lassez, 1991, MIT Press, pp. 640-700.

R. Ramakrishnan. Parallelism in Logic Programs, Annals of Mathematics and Artifical Intelligence, special issue on deductive databases, Vol. 3, N0. 2-4, March 1991, pp 295-330.

2 Scientific Databases

Yannis Ioannidis (yannis@cs.wisc.edu) and Miron Livny (miron@cs.wisc.edu) are the PI's on this project. The goal of the project is to develop an *Experiment Management System (EMS)* that could be used by scientists in arbitrary disciplines to conduct experimental studies. For any such study, the EMS will be connected with multiple, possibly diverse, environments in which experiments could be conducted as well as other similar systems, from which data pertinent to the study may be collected. In order to understand the nature of experimental studies in diverse disciplines, we are interacting with scientists from several other fields, most notably soil sciences (John Norman), molecular biology (Gary Borisy), and genetics

(Fred Blattner). We have identified a common life cycle for arbitrary experimental studies based on which the architecture of the EMS has been designed. Work is underway on most major pieces of the system and some preliminary results have already been obtained and reported. Specifically, with respect to the traditional database issues that arise in an EMS, the Moose object-oriented data model has been developed; it employs some unique characteristics that address specific requirements raised by the special nature of experimental studies and its associated data. With respect to the user interface, an effort is being made to develop a graphical interface that matches the characteristics of the life cycle of experimental studies and is intuitive to domain scientists who may not necessarily be experts in database systems. Finally, with respect to addressing the heterogeneity issues that arise in having the EMS communicate with other such systems and experimentation environments, foundations are being laid out for a uniform methodology that can be applied for translating data between representations that differ at various levels of abstraction as well as for automating the process as much as possible.

2.1 Recent Publications from the Scientific Databases Project

R. Miller, Y. Ioannidis, and R. Ramakrishnan, "Understanding Schemas", To appear in *RIDE-IMS 1993*, Vienna, Austria, April 1993.

R. Miller, Y. Ioannidis, and R. Ramakrishnan, "Foundations of Schema Translation", Submitted for publication.

Y. Ioannidis and M. Livny, "Conceptual Schemas: Multi-faceted Tools for Desktop Scientific Experiment Management", *Journal of Intelligent and Cooperative Information Systems*, December 1992.

Y. Ioannidis, M. Livny, and E. Haber, "Graphical User Interfaces for the Management of Scientific Experiments and Data", *ACM-Sigmod Record*, Vol. 20, No. 1, pp. 47-53, March 1992.

Y. Ioannidis and M. Livny, "Advanced User Interfaces for Accessing Earth and Space Sciences Data", In *Proc. International Space Year Conference on Earth and Space Science Information Systems*, Pasadena, CA, February 1992.

3 Query Optimization and Cost Estimation

The task of query optimizers is made difficult in practice by the following two problems. First, as the query complexity grows, the number of alternative plans that can be used to process the query grows too; performing an exhaustive search over all these plans to find the least expensive one becomes impossible. Second, it is difficult to determine the cost of a plan accurately; the optimizer must compute approximations of the actual cost, and must ensure that the errors in these approximations do not adversely affect the optimization process. The work in this NSF-funded project addresses both problems above and focuses on three main areas: search strategies for query optimization, query cost estimation, and analysis of error propagation in cost estimates. The PI's currently working on this project are Yannis Ioannidis (yannis@cs.wisc.edu) and Jeff Naughton (naughton@cs.wisc.edu.)

3.1 Search Strategies

The work on search strategies focuses on randomized algorithms. We have studied two ~~types~~ of such algorithms: those that perform random walks on a graph whose nodes are the query access plans, and genetic algorithms. Although competitive with the traditional query optimization search strategies, the latter appear to be inferior to the former, which have thus received the greatest emphasis of our work.

In search of the most effective random-walk algorithm, we have focused our attention on the shape of the cost function of plans over the spaces (graphs) associated with large join queries. We have obtained several analytical and experimental results indicating that the shape resembles a '*well*' with smooth sides and an uneven bottom, especially when both deep and bushy plans are included in the space. (2PO), which takes advantage of the 'well'-shape by combining two algorithms proposed earlier, Iterative Improvement and Simulated Annealing. Because of this characteristic, when both deep and bushy plans are taken into account, 2PO outperforms all the other algorithms proposed to date for large queries in terms of both output quality and running time.

In principle, the above optimization techniques are generic in nature, so part of our current and future work is to study their effectiveness on other hard query optimization problems. One such problem is *Parametric Query Optimization*, which is being studied in collaboration with Raymond Ng, Kyuseok Shim, and Timos Sellis. This type of optimization addresses an issue faced by most database systems, where the values of many important run-time parameters of the system, the data, and/or the query are unknown at query optimization time. Parametric query optimization attempts to identify several access plans, each one of which is optimal for a subset of all possible values of the run-time parameters. We have studied this problem primarily for the buffer size parameter. Several randomized algorithms have been adopted for this style of optimization, and enhanced with a 'sideways information passing' feature that increases their effectiveness by forcing the formation of a very clear 'well' in the plan space. Experimental results have shown that these enhanced algorithms optimize queries for large numbers of buffer sizes in the same time needed by their conventional versions for a single buffer size, without much sacrifice in the output quality. Preliminary results with this approach for other types of parameters, e.g., index availability on an attribute, appear promising as well.

The work on search strategies continues in two main directions. First, it appears that for optimizing relational join queries, the optimal search strategy depends on several characteristics. We are working on identifying these characteristics and, based on the obtained results, we plan to develop an optimizer that will choose the appropriate search strategy each time. Second, we intend to investigate the applicability of randomized optimization algorithms to several additional query optimization problems, e.g., in object-oriented or parallel database systems.

3.1.1 Recent Publications on Search Strategies

Y. Ioannidis, R. Ng, K. Shim, and T. K. Sellis, "Parametric Query Optimization", In *Proc. 18th Int. VLDB Conference*, pp. 103-114, Vancouver, BC, August 1992.

Y. Ioannidis, Y. Kang, and T. Zhang, "Cost Wells in Random Graphs", June 1992, Submitted for publication.

K. Bennett, M. C. Ferris, and Y. Ioannidis, "A Genetic Algorithm for Database Query Optimization", In *Proc. 4th Int. Conference on Genetic Algorithms*, pp. 400-407, San Diego, CA, July 1991.

Y. Ioannidis and Y. Kang, "Left-deep vs. Bushy Trees: An Analysis of Strategy Spaces and its Implications for Query Optimization", In *Proc. of the 1991 ACM-SIGMOD Conference on the Management of Data*, pp. 168-177, Denver, CO, May 1991.

Y. Ioannidis and Y. Kang, "Randomized Algorithms for Optimizing Large Join Queries", In *Proc. of the 1990 ACM-SIGMOD Conference on the Management of Data*, pp. 312-321, Atlantic City, NJ, May 1990.

3.2 Sampling-Based Cost Estimation

Our goal in this portion of the project is to develop and evaluate sampling-based strategies for estimating the size of query results, which in turn is useful in query evaluation plan cost estimation. Our work to date on sampling algorithms has focussed on three main issues:

1. How the sampling unit should be defined.

   For example, when implementing an algorithm to estimate a simple selectivity, one can either choose random individual tuples from the relation, or random disk blocks from the relation, using all tuples on each sampled disk block. The second approach clearly examines more tuples per I/O, but these tuples are not necessarily statistically independent. For more complex operations, such as joins, there are more options for how to define the sampling unit. One result of our work is a lattice ordering on the efficiency of a large number of sampling options for general select-join queries.

2. How to know when to stop sampling.

   The simplest stopping condition is to specify a fixed number of samples. However, with a fixed number of samples in some cases the *a priori* fixed number of samples will be too low and estimate will have an unacceptably large errors, while in other cases the fixed number of samples will be too high and the sampling algorithm will be needlessly inefficient. Recently, working with Peter Haas and Arun Swami of the IBM Almaden Research Center we have investigated applying their asymptotically optimal *fixed-precision* stopping conditions to various schemes for sampling select-join queries. (A fixed-precision stopping condition is one such that the sampling terminates after a pre-specified accuracy has been achieved rather than after a pre-set number of samples.)

3. Sampling issues in parallel database systems.

   Parallel database systems are a natural place to employ sampling, since random sampling is an inherently parallel application (each sample is independent of the other samples.) However, care must be taken when randomly sampling in a multiprocessor. If one takes a global simple random sample from a file spread among the disks of a multiprocessor, then load imbalances will arise because of random fluctuations in the number of samples that come from each processor. The alternative, taking a fixed number of samples from each processor, achieves perfect load balancing but instead of returning a simple random sample returns a stratified random sample. In this project we have investigated the tradeoffs between using a global simple random and a stratified sample in multiprocessor sampling applications.

3.2.1 Recent Publications on Sampling

Haas, P., Naughton, J., Seshadri, S., and Swami, A., "Fixed-Precision Estimation of Join Selectivity," submitted for publication.

Lipton, R., Seshadri, S., Naughton, J., and Schneider, "Efficient sampling strategies for relational database operations," to appear in *Theoretical Computer Science*, 1993.

Seshadri, S., and Naughton, J., "Sampling Issues in Parallel Database Systems." In *Proceedings of the International Conference on Extending Database Technology*, Vienna, Austria, March 1992.

3.3 Error Propagation

The propagation of errors in the estimates of query optimizers may have devastating effects on the quality of the decisions of query optimizers, especially for large queries. Our work on the problem has focused on understanding how errors propagate in the result size of join queries and on identifying techniques for limiting such propagation. We have shown that, under certain assumptions, the maximum, average, and standard deviation of error all increase exponentially with the number of joins in a query. Limiting the propagation of the maximum error is the most reasonable approach for systems to take. We have studied various techniques that can be used to achieve that, but we have mostly concentrated on the use of histograms that approximate the frequency distribution of values in the join attributes of relations. For a restrictive class of queries, we have shown that, in the optimal histograms (those that reduce the maximum error the most), join attribute values should be placed in buckets based on their frequency of appearance in the relations and not on the values themselves, as is commonly done. For small queries, our results indicate that the optimal histograms strongly depend on the frequency distributions mentioned above. This implies that the common practice of maintaining accurate frequencies of the few most frequent attribute values may be far from optimal. For very large queries, however, we have shown that this practice becomes optimal. Our work on error propagation continues in the direction of obtaining similar results for arbitrary queries.

3.3.1 Recent Publications on Error Propagation

Y. Ioannidis and S. Christodoulakis, "Optimal Histograms for Limiting Worst-case Error Propagation in the Size of Join Results", to appear in *ACM-TODS*, 1993.

Y. Ioannidis and S. Christodoulakis, "On the Propagation of Errors in the Size of Join Results", In *Proc. of the 1991 ACM-SIGMOD Conference on the Management of Data*, pp. 268-277, Denver, CO, May 1991.

4 The Zeta Project

The Zeta project research focuses on issues in the implementation of "shared-nothing" parallel database systems. The PIs on the project are Mike Carey, David DeWitt, and Jeff Naughton, all of whom can be reached by `lastname@cs.wisc.edu`. This work is funded by IBM (through an IBM Research Initiation Award), NCR, and Tandem. In addition to developing better algorithms for basic query processing in parallel database systems, the Zeta project is concerned with second-generation parallel DBMS research issues such as concurrency control and resource scheduling.

Most parallel database machine research to date has addressed important but specialized aspects of the database system domain. For example, in some cases the workload is assumed to be a homogeneous stream of short transactions; in other cases, the workload is assumed to be a single-user sequence of complex join queries. Currently, little is known about how to harness the power of a large parallel database machine to service a dynamic mix of short transactions, ad-hoc queries, and complex batch queries. Providing the capability to service such a workload requires basic advances in concurrency control and resource scheduling.

Processing a mix of small transactions and complex, on-line queries poses several challenging problems. One problem is that complex queries often require many fine granularity locks (or alternatively, fewer, coarser granularity locks) and hold them for long periods of time, impeding the progress of short transactions. The current commercial solution is "browse mode" execution, which allows complex queries to obtain approximate answers by reading inconsistent data. Clearly, this is not sufficient for all applications. For conventional database management systems, multiversion concurrency control algorithms are an alternative solution, and simulations indicate that they may indeed be effective. We are exploring the use of multiversion locking algorithms in the context of a parallel database machine.

An important related problem, also beyond the current state of the art, is how to schedule the many physical resources of a parallel database machine at runtime. In particular, decisions made by the query optimizer are based on cost estimates, which in turn are based on assumptions about the amount of available memory and the number of available processors. In fact, neither can truly be known until query execution time, long after the query optimizer has generated and compiled a plan for executing the query. But what if the system is incrementally presented with a workload containing many queries of varying sizes and complexity? Runtime scheduling issues we are considering include how much memory to allocate to the operators of each query and, for intermediate query results, over how many processors to decluster the results in order to parallelize the execution of subsequent query operators. Moreover, some queries in the workload may be more important than others, and should thus be given higher priority. Lastly, replicated data provided by chained-declustering provides an opportunity to use dynamic load balancing to correct load imbalances caused by skewed or hot data. We are investigating adaptive algorithms for making the runtime scheduling decisions needed to effectively cope with complex query mixes.

4.1 Recent Publications from Zeta

Bober, P., and Carey, M., "Multiversion Query Locking,", *Proc. of the 18th Int'l. Conf. on Very Large Databases*, Vancouver, BC, Canada, August 1992.

Bober, P., and Carey, M., "On Mixing Queries and Transactions via Multiversion Locking," Proc. of the 8th Int'l. Conf. on Data Engineering, Phoenix, AZ, February 1992.

Brown, K., Carey, M., DeWitt, D., Mehta, M., and Naughton, J., "Resource Allocation and Scheduling for Mixed Database Workloads," Computer Sciences Technical Report No. 1095, University of Wisconsin-Madison, July 1992.

DeWitt, D., Lieuwen, D., and Mehta, M., "Pointer-based Join Techniques for Object-Oriented Databases", in *Parallel and Distributed Information Systems '93*, January 1993.

DeWitt, D., and Gray, J., "Parallel Database Systems: The Future of Database Processing or a Passing Fad," *Communications of the ACM*, June, 1992.

DeWitt, D., Naughton, J., and Schneider, D., "Parallel Sorting on a Shared-Nothing Architecture using Probabilistic Splitting," in *Parallel and Distributed Information Systems '91*, December 1991.

DeWitt, D., Naughton, J., and Burger, J., "Nested loops revisited," in *Parallel and Distributed Information Systems '93*, January 1993.

D. DeWitt, J. Naughton, and D. Schneider, "A Comparison of Non-Equijoin Algorithms", in *Proceedings of the 17th International Conference on Very Large Databases*, August 1991.

D. DeWitt, J. Naughton, and D. Schneider, "Parallel External Sorting using Probabilistic Splitting", in *Proceedings of the Symposium on Parallel and Distributed Information Systems*, December 1991.

D. DeWitt, J. Naughton, D. Schneider, and S. Seshadri, "Practical Skew Handling in Parallel Joins", in *Proceedings of the 18th International Conference on Very Large Databases*, August, 1992.

Hsiao, H., and DeWitt, D., "A Performance Study of Three High Availability Data Replication Strategies", in *Parallel and Distributed Information Systems '91*, December 1991.

Ghandeharizadeh, S., DeWitt, D., and Qureshi, W., "A Performance Evaluation of Alternative Multi-Attribute Declustering Strategies," *Proc. of the ACM SIGMOD Int'l. Conf. on Management of Data*, San Diego, CA, June 1992.

M. Mehta and D. DeWitt, "Dynamic Memory Allocation for Multiple-Query Workloads", submitted for publication.

Shatdal, A., and Naughton, J., "Using shared virtual memory for parallel join processing," submitted for publication.

Srinivasan, V., and Carey, M., "Compensation-Based On-Line Query Processing," *Proc. of the ACM SIGMOD Int'l. Conf. on Management of Data*, San Diego, CA, June 1992.

Srinivasan, V., and Carey, M., "Performance of On-Line Index Construction Algorithms," *Proc. of the Int'l. Conf. on Extending Database Technology*, Vienna, Austria, March 1992.

5 The SHORE Persistent Object Repository Project

Over the past five years, significant progress has been made on software tools for a variety of applications. However, these tools still rely largely on untyped, unstructured Unix files for storing and exchanging data, as relational database systems have failed to meet their functionality and performance requirements. Recent first-generation object-oriented database products are an improvement, but they are typically closed, single-language systems (e.g., persistent C++) that do not support heterogeneous application environments, let alone the high-performance parallel computer systems that will host the next generation of software tools.

This DARPA project aims to develop a system called SHORE (Scalable Heterogeneous Object REpository), a high-performance, scalable, persistent object system. The PIs on the SHORE project are Mike Carey, David DeWitt, Jeff Naughton, and Marv Solomon. SHORE is being designed to support a wide variety of performance-critical applications and tools, such as hardware and software CAD systems, persistent programming languages, and hypertext systems; our eventual goal is to totally displace Unix files in such application areas. One component of our strategy for migrating applications from files to objects is the provision of a graceful migration path, allowing legacy tools (for example, existing compilers in the case of software CAD) to access SHORE objects through the standard Unix file interface. In addition, SHORE will offer multiple levels of service in areas such as transaction management, thus allowing clients to select an appropriate (i.e., sufficient yet affordable) level of service. SHORE is being targeted at a wide range of hardware environments, from individual workstations to heterogeneous client/server systems up to large-scale multicomputers.

A fundamental goal of SHORE is to act as a repository for persistent objects that can be accessed and updated from programs that are written in multiple programming languages and run on different hardware platforms. As a result, the SHORE type system is a descendent of the IDL type system of OMG's Common Object Request Broker; it is also related to the ODL type system proposal from ODMG (the Object Database Management Group, a consortium of object-oriented database system vendors). The type system for SHORE objects, known as the SHORE Data Language (SDL), includes the usual set of primitive data types and a set of type constructors that include structures, discriminated unions, arrays, and interfaces (similar to C++ classes). SHORE objects can also participate in relationships with one another. Relationships are modeled using typed pointers as well as sets, bags, and sequences of such pointers; support is provided for referential integrity maintenance (e.g., inverse relationships). By design, SHORE avoids the "everything is an object" approach to complex objects; SHORE objects may contain arbitrarily complex data structures as part of their internal structure. The world of persistent objects managed by SHORE will reside in a Unix-like namespace, involving a hierarchical directory structure, extended to include various additional kinds of system objects (including named top-level SHORE objects, pools of "lighter weight" objects, modules of types, and a new kind of link).

The SHORE design consists of three main layers. At the bottom is an untyped storage management layer, providing facilities for I/O, disk space management, locking, recovery, indexing, and cache consistency. The next layer is a language-independent layer that implements the SDL type system and handles the details of typed persistent (i.e., SDL) object management. This layer provides a common set of facilities for dealing with heterogeneous language access, including support for translating between object representations. The topmost layer of SHORE consists of a set of language bindings, one per supported language, that provides a more desirable (language-appropriate) view of SHORE for application programmers. The SHORE runtime system will consist of peer SHORE processes, one per machine, each managing any local disks and providing inter-transaction caching of both local and remote data for its client applications. The implementation of persistence will involve a unique mix of object-faulting (for SHORE objects) and virtual memory-mapping (for objects that are large and/or that contain complex data structures).

We plan to implement versions of SHORE for client/server workstation environments and for a Paragon multicomputer running OSF. Each version presents important research and engineering challenges. Common to both are challenges pertaining to the SDL data model and Unix file compatibility, schemes for fast inter- and intra-object navigation, effective handling of heterogeneity, and the identification of the "right" range of service levels for SHORE clients. Additional client/server challenges include the design of strategies for object clustering and for distributed object caching and replication. Multicomputer-related challenges include understanding the tradeoffs between object clustering and declustering for storing complex objects on parallel disks and identifying opportunities (and appropriate interfaces) for exploiting large-scale parallelism in an object store.

5.1 Recent Publications from SHORE

Carey, M., DeWitt, D., and Naughton, J., "The OO7 Benchmark," submitted for publication.

Franklin, M., Carey, M., and Livny, M., "Global Memory Management in Client-Server DBMS Architectures," *Proc. of the 18th Int'l. Conf. on Very Large Databases*, Vancouver, BC, Canada, August 1992.

Franklin, M., and Carey, M., "Client-Server Caching Revisited," *Proc. of the Int'l. Workshop on Distributed Object Management*, Edmonton, Canada, August 1992.

Franklin, M., Zwilling, M., Tan, C., Carey, M., and DeWitt, D., "Crash Recovery in Client-Server EXODUS," *Proc. of the ACM SIGMOD Int'l. Conf. on Management of Data*, San Diego, CA, June 1992.

Lieuwen, D. and DeWitt, D., "A transformation-based approach to optimizing loops in database programming languages," *Proc. of the ACM SIGMOD Int'l. Conf. on Management of Data*, San Diego, CA, June 1992.

Tsangaris, M., and Naughton, J., "A comparison of object clustering techniques," In *Proceedings of the ACM SIGMOD Symposium on Management of Data*, San Diego, California, June 1992.

White, S., and DeWitt, D., "A performance study of object faulting and pointer swizzling strategies", in *Proc. of the 18th Int'l. Conf. on Very Large Databases*, Vancouver, BC, Canada, August 1992.

Yong, V., and Naughton, J., "Storage reclamation and reorganization in client-server persistent object stores," submitted for publication.

6 Real-Time Database Systems

As discussed elsewhere, database systems are being called upon to provide database management services for a variety of new application environments. Some of these environments have a time-critical nature, where some of the tasks in the workload must be completed within a time limit (after which time the point of completing the task either diminishes or altogether disappears). We have an ongoing research effort that is addressing issues that arise in the design of a DBMS that is capable of meeting the challenges posed by such time-critical database applications. The PI's on this project are Mike Carey carey@cs.wisc.edu) and Miron Livny (miron@cs.wisc.edu). We have been working largely in the context of what we call "firm" real-time environments, where a transaction that does not complete by its deadline is no longer worth completing.

One topic that we (as well as researchers elsewhere) have worked on is the design and evaluation of transaction scheduling algorithms for a real-time DBMS. This portion of our effort has led to the design of several concurrency control algorithms that are tailored for real-time databases; these algorithms are optimistic in nature, as our performance studies have indicated that a firm real-time environment is one in which optimistic methods have promise (perhaps surprisingly, but for good reason, as they make scheduling decisions at the point where it is fairly clear how well conflicting transactions are doing towards meeting their time constraints). We have also evaluated a broad range of alternative solutions for real-time DBMS concurrency control.

Recently, we have looked at ways to ensure that a real-time DBMS will provide both timely and fair service to transactions in mixed workloads. The natural tendency is for a real-time scheduling algorithm to favor short tasks over long tasks if the short ones have nearer-term deadlines, and this tendency can lead to very poor performance for long real-time tasks. We have developed an approach, based on setting "virtual deadlines" for long tasks, that eliminates this scheduling bias. We are currently studying the problem of managing system memory in the presence of mixed workloads that contain tasks – such as large queries – with significant memory demands. One very recent outcome of this work is a family of hash join algorithms that are capable of adapting to the amount of buffer memory currently available for use in computing the join.

6.1 Recent Publications on Real-Time DBMS

Pang, H., Carey, M., and Livny, M., "Partially Preemptible Hash Joins," submitted for publication.

Pang, H., Livny, M., and Carey, M., "Transaction Scheduling in Multiclass Real-Time Database Systems," *Proc. of the 13th Real-Time Systems Symposium*, Phoenix, AZ, December 1992.

Haritsa, J., Livny, M., and Carey, M., "Value-Based Scheduling in Real-Time Database Systems," *VLDB Journal*, to appear.

Haritsa, J., Carey, M., and Livny, M., "Data Access Scheduling in Firm Real-Time Database Systems," *Journal of Real-Time Systems*, Vol. 4, No. 3, September 1992.