The Microsoft Database Research Group

David Lomet Roger Barga Surajit Chaudhuri Paul Larson
Vivek Narasayya
Microsoft Research
One Microsoft Way, Bldg. 9
Redmond, WA 98052
http://www.research.microsoft.com/research/db

1 Overview

1.1 History

Microsoft's strategic interest in the database field dates from 1993 and the efforts of David Vaskevitch, who is now the Microsoft Vice President in charge of the database and transaction processing product development groups. David's vision was that the world would need millions of servers, and that this presented a wonderful opportunity to a company like Microsoft that sells software in high volume and at low prices. Database systems played an important role in Vaskevitch's vision, and, indeed, in Microsoft's current product plans. David began looking for premier database and transaction processing people in late 1993.

The scope of Vaskevitch's efforts included a desire for Microsoft to establish a database research group. Rick Rashid, Microsoft Research Vice President, collaborated with Vaskevitch in recruiting David Lomet from Digital's Cambridge Research Lab to initiate the Microsoft Database Research Group. Lomet joined Microsoft Research in January of 1995. Hence, Microsoft's Database Research Group is now a little over three and a half years old.

One person does not a group make. Recruiting efforts continued. Surajit Chaudhuri, a researcher from HP Labs joined the Database Group in February of 1996. Paul Larson, a professor from the University of Waterloo joined in May of that year. Vivek Narasayya was initially an intern as a graduate student from the University of Washington in the summer of 1996, officially joining the group in April of 1997. Roger Barga, the newest member of the group and a new Oregon Graduate Institute Ph.D., joined in December, 1997.

1.2 Microsoft's Research Environment

The Database Research Group benefits from attributes of our Microsoft environment that contribute to the effectiveness of research here. These apply to Microsoft Research in general, not only to database research.

Research Ambiance: Microsoft Research, and specifically Rick Rashid, believes that one very significant measure of the quality of our research efforts is publication of our results in the major conferences and journals of our fields. This stimulates us to measure our success against the best researchers in each area. The other significant measure of our research is transfer of technology to product groups, which focuses our attention on industrially relevant problem areas. Thus, professional impact and product relevance combine to motivate high quality industrial research.

Geography: Essentially all of Microsoft product development is located on a single campus in Redmond. This means that Microsoft's Redmond based researchers are only about a five-minute walk from any development team with which they are collaborating. This is an enormous plus that greatly facilitates both consulting activities and the transfer of research developed technology.

Developers: Microsoft has highly skilled software developers who are almost uniformly a pleasure to work with. In addition, they are very accepting of technology that furthers the success of their products. This product focus means that it is possible for the latest in research technology to very quickly find a home within Microsoft products. Having great developers means that going from research prototype to delivered product can happen at stunning speed.

Market Impact: Industrial researchers, like product developers, want their efforts to have impact. When a research idea gets into a Microsoft product, it can impact the way that millions of people use computers. This is heady stuff that can result in tremendous job satisfaction.

1.3 Database Research Activities

The challenge of industrial database research is to find the technical areas that leverage research skills to produce large leaps in technical capability. We deal with this with a combination of research projects and consulting activity.

Research Projects

The high quality of the Microsoft product developers makes choosing research projects particularly challenging. It requires both identifying long range opportunities and being able to deliver incremental progress on those challenges in a timely fashion- that is, a step ahead of our colleagues from the product groups. We have identified two areas in which it is clear that there are major opportunities with significant payoff if our efforts succeed. Both these areas leverage our painstakingly acquired knowledge of the areas and our research skills.

Self-tuning databases: The *AutoAdmin project* long-term goal is to make database systems self-administering and self-tuning in all their dimensions. Initially, the project is focusing on the physical database design problem (index and materialized view selection). Surajit Chaudhuri, Paul Larson, and Vivek Narasayya collaborate on the AutoAdmin project.

Robust applications: the *Phoenix project* long-term goal is to improve application availability and error handling robustness. Initially, the project is focusing on exploiting database recovery techniques to enable applications to survive system crashes. David Lomet and Roger Barga work on the Phoenix project.

Our research projects are described more fully in the next two sections.

Consulting Activities

In addition to our research projects, members of the Database Research Group also work directly with product development groups on more immediate technical problems. We view this type of internal technical consulting as an important part of our charter as an industrial research group. Consulting keeps us close to the current product activity and strengthens ties with the development teams. It not only is helpful to the developers, but also keeps us aware of and focused on real world problems.

As a research group, we are exposed constantly to the activities and results produced by a wide range of research groups throughout the world. Our consulting activities enable us to bring research results (our own and others) to the attention of the product developers. Consulting activities have taken several forms, ranging from involvement in exploring designs, through prototyping and evaluation of potential solutions, all the way to contributing virtually ready-to-ship code.

Our consulting activities have had a measurable impact on several Microsoft products.

SQL Server: The SQL Server 7.0 release is a major enhancement to SQL Server. Large portions of the query processing and storage engine components were rewritten with greatly increased functionality, performance, extensibility, and modularization. Our research consulting was particularly helpful in the query optimizer for index usage, in the storage engine for support of record level locking, and for improved OLAP performance.

Windows NT: Windows NT is a modern and constantly evolving operating system intended for both desktop and server roles. Our consulting on its caching behavior and memory usage helped lead to important performance gains.

Internet Systems: Servicing the Internet is a new and highly important area, with substantial new systems being built. These systems have caching and memory usage that also were improved with the help of our consulting.

2 The AutoAdmin Project

2.1 The Opportunity

More than ever before, database systems are being used as integral components of a variety of enterprise and desktop applications. In order for such widespread use to be cost-effective, it is important that the cost of ownership of databases be low. Unfortunately, this is not true of today's commercial database systems. They require careful database administration and tuning for good performance. Furthermore, administration and system tuning are complex tasks that require considerable expertise. Automating and simplifying these tasks is crucial to making deployment of databases more affordable.

The AutoAdmin objective is to help develop a next generation database system that adapts automatically and gracefully to its environment. The vision is to have a system that consistently delivers high performance with little or no administration, regardless of changes in its load and environment. Monitoring and feedback are key techniques required to make a database management systems self-tuning. This means gathering information while the system is running and exploiting this information to adjust the system parameters to improve future performance. The time horizon for tuning can vary greatly: from a few milliseconds to days. An example of long term tuning is determining an appropriate set of index structures. An example with a shorter time horizon is the estimate of query execution time when results of previous executions can be exploited.

2.2 The Project

AutoAdmin is a long-term effort. In the short term, we have focussed our research project efforts on automating the task of picking an appropriate physical design for a database. Specifically, we have concentrated on the problem of identifying appropriate indexes and materialized views to optimize the performance of a database on a given workload.

2.2.1 Selecting Indexes and Materialized Views

The index selection problem has been studied since the early '70s and the importance of the problem is well recognized. Despite a long history of work in this area, there are few research prototypes and commercial products that are widely deployed. There are several difficulties in designing a robust industrial strength index selection tool.

- Workloads consist of arbitrarily complex SQL statements and change over time. This underscores the need to ensure that workloads can be tracked and that query complexities can be handled.

- Modern query processing systems exploit indexes in many more ways than were done in early relational systems, e.g. index only queries, index intersections, multi-column indexes. Hence, the design space is very large, making efficient search of this space essential.

- Query optimizers have specific characteristics that make the plans that they generate rather idiosyncratic. A particular physical design is not interesting unless the optimizer exercises its features. This underscores the need to ensure that the design tool is in step with the optimizer.

To the best of our knowledge, none of the past work has addressed these fundamental issues satisfactorily. Textbook solutions to the physical database design problem that take only semantic information such as uniqueness, reference constraints and rudimentary statistics to produce a database design perform poorly because they ignore valuable workload information. The class of tools that adopt an expert system like approach, like Rdb Expert, where the knowledge of good designs are encoded as rules, suffer from being disconnected from the query optimizer.

The AutoAdmin index selection technology that we have developed required identifying and prototyping new database server interfaces to permit the creation of hypothetical indexes. The creation of a hypothetical index requires efficiently gathering statistics on columns of the index. We exploit sampling techniques for this step [CMN98]. Two component tools were implemented that exploit these interfaces.

The **index analysis utility** [CN98] creates a set of hypothetical indexes and analyzes their impact with respect to varied workloads on the system. The analysis utility can be exploited by a variety of client tools.

We leveraged the index analysis utility to develop an **index tuning wizard** that iterates through the space of hypothetical indexes efficiently to propose a set of indexes appropriate for a given workload. Such a workload may be derived from a customer benchmark or obtained by logging the database server activity using available tools. For each choice of a set of hypothetical indexes, it uses the special database server interfaces to create the hypothetical indexes and evaluate their potential for performance enhancement with respect to the given workload. The index tuning wizard uses a novel search technique that filters out spurious indexes in an early stage and exploits characteristics of the relational query engine to reduce the cost of selecting indexes. For example, it takes into account index-only access. It also generates complex alternatives (e.g., multi-column indexes) from good simpler alternatives (e.g., single-column indexes) in a structured fashion. Technical details of this wizard may be found in [CN97].

Despite the fact that the AutoAdmin project is relatively young, we have been successful in impacting SQL Server. Its next release (SQL Server 7.0) will feature our index tuning wizard, which can be launched in a variety of ways to select a set of appropriate indexes for a workload [CN98-wp]. A workload may be provided externally or created using the SQL Server profiler. The index tuning wizard will be a significant contribution to the SQL Server focus on ease of administration.

As the next step in automating the physical design, we are currently expanding the physical design space to include selecting not only a set of indexes, but rather a set of indexes as well as materialized views. Indeed, an index can be seen as a very simple form of a materialized view: a project-only view. Other types of materialized views, for example, join views or aggregation views, have the potential to provide similar performance gains as indexes.

2.2.2 Query Processing with Materialized views

In order to extend the choice of physical design to materialized views, not only do we need to extend the index selection framework to include materialized views, but we also need to ensure that the query processing subsystem can support materialized views. Specifically, it requires solutions to the following two problems:

1. *Query transformation:* The query optimizer must be able to rewrite queries to exploit materialized views when it is beneficial to do so.

2. *View maintenance:* The system must automatically update all affected materialized views whenever base

tables are updated.

So far, our work on materialized views has focused on the query transformation problem. Most previous research on query transformation has made several simplifying assumptions: project-select-join queries and views only, set semantics (as opposed to multiset semantics), no knowledge about constraints like keys and foreign keys, and computing the query from a single view. Currently we have a running prototype system for query transformation. that handles a broader class of queries and views (project-select-join-group) with normal SQL semantics, exploits knowledge about keys and foreign keys, and considers transformations involving multiple views. SQL's multiset semantics adds a new dimension to the query transformation problem because we have to make sure not only that we get all the required rows but also with the right duplication factor.

Taking into account knowledge about keys and foreign keys turns out to be very important but also suprisingly complex. Consider two tables, Orders and Customers, where the Orders table contains a non-null foreign key CustomerNo referencing the primary key of Customers. Assume that we have a join view consisting of the (natural) join of Orders and Customers and consider a query referencing Orders only. Without knowledge about keys and foreign keys, we would have to conclude that the query cannot be computed from the join view. Taking into account just foreign keys, we can conclude that all required rows exist in the view but not necessarily with the right duplication factor. To guarantee that the duplication factor is correct, we have to take into account that the join is on a key.

Key dependencies generate functional dependencies that hold in the result of a query expression. These derived functional dependencies play an important role when deciding whether a group-by query can be computed from a group-by view.

Combinatorial explosion rears its ugly head as soon as one broadens the solution space to transformations that use multiple views. The key problem here is to come up with good heuristics for limiting the search without missing too many good solutions. The heuristics in the current prototype work well on small databases but we don't know how well they scale to large databases with hundreds or even thousands of tables and views.

References

CN97 Chaudhuri, S., Narasayya V. "An Efficient, Cost-Driven Index Selection Tool for Microsoft SQL Server." Proceedings of the 23rd VLDB Conference Athens, Greece, 1997, pages 146-155.

CMN98 Chaudhuri S., Motwani R., Narasayya V. "Random Sampling for Histogram Construction: How much is enough?" Proceedings of the ACM SIGMOD International Conference on Management of Data, 1998.

CN98 Chaudhuri, S., Narasayya V. "AutoAdmin "What-If" Index Analysis Utility." Proceedings of the ACM SIGMOD International Conference on Management of Data, 1998.

CN98-wp Chaudhuri S., Narasayya V. "Index Tuning Wizard for Microsoft SQL Server". Microsoft White Paper.

3 The Phoenix Project

3.1 The Opportunity

Dealing with errors or exceptions is a very large part of getting applications "right". Such errors and exceptions greatly increase the complexity of application programming. However, failures are not only an application programming problem but an operational and an availability problem as well. The Phoenix project is an effort to decrease application programming complexity, increase the availability of applications, and in many cases avoid the operational task of coping with an error.

System Crashes

Database systems recover the *database* to the last committed transaction. Incomplete transactions are aborted. While database state is recovered, the states of applications using the database, and their surrounding sessions are "blown away" (erased). This behavior results in longer *application* outages. Our intent is to reduce this period of unavailability by extending database recovery to include session and application states. This will also enable stateful applications to survive failures and continue execution.

Logical Errors

Transactions abort for logical errors as well as crashes. Aborting transactions in these cases means undoing back to transaction start. In the future we would like to extend database style recovery to support partial rollback as a result of application errors, where the rollback resets not only database state (already supported by savepoints) but also application state. This is compensation, of the multi-level transaction form, that includes application state.

3.2 The Project

In Phoenix, we have focused first on application availability and persistence.

Redo Recovery Technology

We have explored technology that exploits new database redo recovery technology [LT95] to enable applications to persist across system crashes, i.e. to provide for the recovery of application state as well as database state [L97,L98]. This permits applications to safely maintain state across multiple transactions. While forms of program persistence are not new, logging and checkpointing costs to realize persistence have been high. The techniques developed within Phoenix substantially reduce these normal execution costs by enabling logical logging, which reduces logging costs. These techniques exploit the database system's cache management and recovery mechanisms. While there remains an extra system cost for application persistence, it is much lower than in the past. Phoenix continues the trend of expending system resources to conserve more expensive and error-prone human resources.

Because this work exploits the database system's recovery mechanism, our approach requires the database system to wrap an application to capture its interactions with other system components and log its state changes. Hence, our focus has been on database applications, particularly those that are close to the database system. This permits simple robust applications such as database stored procedures.

With further extension, robust client/server database applications can be provided as well [LW98]. Further evolution of these techniques should enable the masking of system failures involving application subcomponents from higher level components in a more distributed environment, such as distributed transaction processing or workflow.

ODBC Persistent Sessions Prototype

Our initial systems effort avoids the difficulty involved with making substantial changes to the internals of the database recovery system by focusing on ODBC session availability. The term ODBC stands for "open database connectivity" – a technology based on an ANSI/ISO standard that allows applications to access multiple third-party databases. ODBC exploits a general-purpose call level interface (CLI) that uses SQL as its standard for accessing data. Our goal is to provide persistent server sessions to client systems that support ODBC, sessions that can survive a system crash without the client application being aware of the outage, except perhaps for timing considerations.

When a client application requests information from a database, the request goes to an ODBC driver, which is a database system specific program that actually accesses the database. The ODBC driver translates the request so that the database server can recognize and respond to it. The server provides the requested data to the ODBC driver, which then translates the data into a form the ODBC client application will recognize.

To provide ODBC session availability, we have introduced a generic Phoenix ODBC driver, one that will work with any database. Our Phoenix driver intercepts *every* application interaction with the database system by means of ODBC, as only these interactions can change ODBC session state. It essentially wraps itself around any database specific ODBC driver. The Phoenix driver intercepts application requests going to the database server, logging statements that alter session context, and carefully rewriting selected SQL statements to force the creation of persistent database tables that capture application state. It then passes the request on to the native ODBC driver. Replies are returned from the server to the native ODBC driver. The Phoenix ODBC driver intercepts the native ODBC driver responses to the client application, variously caching, filtering, and reshaping the result set, and synchronizing with the application state materialized on the database server.

When the Phoenix driver detects that the database server has failed, it periodically probes for the status of the server. Once the server has recovered, the Phoenix driver reestablishes a connection to the database system, then issues a series of calls to the database server in order to reinstall ODBC session context and verify that all application state that was materialized on the server was recovered by the database's recovery mechanisms. The Phoenix driver will identify the application's last completed request, ask the database to re-send the result set if necessary, and reissue any incomplete or interrupted requests.

When using our generic Phoenix ODBC driver, an application programmer does not need to deal directly with server failures. Indeed, a user of the application, end user or other software, may not even be aware that a server crash has occurred, except for some delay. Moreover, all the logic for recovering an ODBC session is localized in the Phoenix driver and can be used by any application to enhance ODBC session availability, without having to modify the application program, the database specific ODBC driver, or the database server.

References

LT95 Lomet, D. and Tuttle, M. Redo Recovery after System Crashes. VLDB Conference (Sept. 1995) Zurich, Switzerland 457-468.

L97 Lomet, D.B. Application Recovery: Advances toward an Elusive Goal. Workshop on High Performance Transaction Systems (HPTS97) Asilomar, CA (September, 1997)

L98 Lomet, D.B. Persistent Applications Using Generalized Redo Recovery. International Conference on Data Engineering, Orlando, FL (Feb. 1998) 154-163.

LW98 Lomet, D.B. and Weikum, G. Efficient Transparent Application Recovery in Client-Server Information Systems. ACM SIGMOD Conference, Seattle, WA (June 1998)(*best paper award*).