The COMFORT project
a comfortable way to better performance

ETH

Eidgenössische
Technische Hochschule
Zürich

Departement Informatik
Institut für
Informationssysteme

Gerhard Weikum
Christof Hasse
Axel Mönkeberg
Peter Zabback

The COMFORT Project:

A Comfortable Way to Better Performance

July 1990

137

Authors' address:

Informationssysteme
ETH-Zentrum
CH-8092 Zurich, Switzerland

e-mail: weikum@inf.ethz.ch

The COMFORT Project: A Comfortable Way to Better Performance

Gerhard Weikum, Christof Hasse, Axel Moenkeberg, Peter Zabback

Information Systems – Databases
Computer Science Department
ETH Zurich
CH-8092 Zurich, Switzerland

Abstract

This paper is an initial outline of the COMFORT project. The overall objective of COMFORT is to automate tuning decisions for transaction processing in database systems. The basic idea is to determine most tuning parameters and run-time strategies on the basis of a compile-time analysis of the transaction programs and statistical information about the data and the workload. The issues that are particularly addressed within this framework are: 1) workload-customized concurrency control and recovery protocols that exploit semantics yet do not incur much overhead, 2) exploiting intra-transaction parallelism in nested transactions, 3) intelligent data allocation and migration to optimize data access in a storage hierarchy consisting of large memory, disk-arrays, and optical disks, and 4) intelligent resource management to coordinate the policies for load control, CPU scheduling, and memory allocation with the transaction manager.

1. Scope and Objectives

COMFORT stands for Compile-time Performance Tuning. The overall objective of COMFORT is to automate tuning decisions for transaction processing in database systems. Our basic approach towards this objective is to determine most tuning parameters and run-time strategies on the basis of a compile-time analysis of the transaction programs and statistical information about the data and the workload. Compared to currently available database systems, COMFORT is supposed to

- extend the range of tuning opportunities, e.g., by introducing transaction-type-specific system parameters,
- eliminate, to a large extent, the need for a human tuning expert like a DBA,
- provide guidelines for building a database system that adapts itself to the workload without incurring much overhead, and,
- ultimately, remove (the need for) manual tuning tricks from (source-level) transaction programs and make application development easier.

Database system tuning is usually considered hard. Typically, systems provide the DBA with a number of tuning knobs, and that is about it. The default parameters do often not bear any relationship with any reasonable application, and there is no methodology for

finding the right settings other than experience. Compared to other areas such as logical database design, performance tuning is even more based on guesswork or the intuition of a database guru. Unfortunately, even if intuition were considered to be a viable solution, there would be far less gurus than performance-critical DBMS installations in the world. Therefore, the main goal of COMFORT is to develop heuristic guidelines and rules of thumb for "intelligent" tuning decisions, and to automate the application of such rules in a system environment.

Related Work

Previous work that is related to COMFORT can be found in both the database system (DBMS) and the operating system (OS) literature. In DBMSs, the idea of using "performance hints" for prefetching, buffer management, scheduling, or concurrency control has been around for quite some time. However, while extensible database systems such as Exodus and even relational database systems have introduced such hints, the actual directives must be provided by the database administrator, system programmers, or even application programmers. For example, DB2's "cursor stability" option allows programmers to relax strict two-phase locking [DB2], Ingres supports explicit preclaiming [RTIa], Exodus [Ca86a] and Bubba [Bo90, CFW90] provide hooks for overriding the LRU buffer replacement policy, and DASDBS offers various degrees of set-oriented disk I/O [Wei89a]. Other performance-enhancing heuristics such as intelligent prefetching and allocation strategies are typically hard-wired in the system code, and hardly adaptable to an application's special needs. Unlike these approaches, COMFORT is supposed to generate performance hints automatically at compile-time. Previous work on automatic DBMS tuning is scarce and has mostly concentrated on single issues such as prefetching [WZ86], adaptive clustering, or scheduling [HK88, HK89]. The research on query optimization, on the other hand, does usually not consider such low-level issues, despite their significant performance impact. COMFORT pursues an integrated approach to improve the overall system performance.

Unlike the DBMS area, there is ample work on adaptive and "self-optimizing" algorithms in the OS and distributed-systems literature (e.g., [BR76, BR89, De76, De80, RP81, ELZ86, Wo86]). However, it seems fair to say that none of the investigated approaches can be considered as a breakthrough in automating the tuning of a full-fledged system. In our opinion, the reason for the lack of system-wide solutions is the inherent unpredictability of OS workloads. Thus, there is usually no compile-time information available to the adaptive OS algorithms. In contrast, the main load of a DBMS often consists of predefined (sometimes called "canned") transaction programs, so that lots of compile-time hints can be passed to the DBMS run-time system. This in turn reduces the overhead of an adaptive system substantially. We believe that even though dynamically defined (sometimes misleadingly called "ad-hoc") decision-support transactions will play a bigger role, the relative predictability of a DBMS workload is an important asset in the COMFORT project. At least, DBMSs can understand the task they are optimizing, whereas OSs can at best analyze load statistics (e.g., [DI89])

Outline of the Paper

The rest of the paper is organized as follows. In Section 2, we discuss basic assumptions on our target environment. In Section 3, we elaborate on the COMFORT architecture. Then, Section 4, which is the main part of the paper, presents various individual research directions that are being pursued in the COMFORT framework, namely inter- and intra-transaction parallelism, intelligent data allocation and migration, and intelligent resource management in DBMSs. For each direction, we present a top-down view of the problem area, and discuss several concrete tuning problems in more detail, namely using latches in "compiled concurrency control protocols", parallel execution strategies for multi-key queries, incremental data-placement optimization, and conflict-driven load control. The paper is concluded with an outlook on our long-range goals.

2. Basic Assumptions

In this section, we discuss several assumptions on the software and hardware components of our target environment. These assumptions are based on our view of future trends in DBMS-centered computing systems. The purpose of this section is to make these assumptions explicit, in order to explain why we believe that COMFORT is a relevant and promising project.

2.1 Application and System Software

Based on the opinions of several senior researchers [Li85, Reu88, Sp87, Sv84], we expect the transaction concept to play an overriding role in the architecture of future application and system software. Transactions have proven to be a convenient means for coping with concurrency and various sorts of failures in a modular way that is transparent to the application programmer.

In traditional DBMS applications, the complexity of transaction programs varies from simple TP1 (i.e., debit/credit transactions) to interactive decision-support transactions that access lots of data. In either case, we expect that all transactions execute compiled code; that is, decision-support transaction programs are written as "stored procedures" in a 4GL, as opposed to "ad-hoc queries" on a pure SQL interface. This assumption is crucial in that it allows us to pass all performance-relevant transaction programs through an analysis step that extracts global tuning information. Thus, even though future DBMS applications will probably comprise an increasing variety of transaction types, it seems that the overall workload can still be characterized sufficiently concise to make COMFORT a viable approach.

For non-traditional DBMS applications such as office automation or CIM, there is a fairly broad consensus that the classical transaction concept is not adequate. Such evolving applications pose higher demands on the transaction management in that they deal with long-lived activities that require more flexible notions of isolation and atomicity. We believe that *open nested transactions* [Gr81, Tr83, BSW88] can serve as a well-founded basis for extended transaction facilities, since they incorporate semantic concurrency control as well as (sub-)transaction compensation rather than merely relying on vanilla

2PL and state-based undo. Our implementation experience with multi-level transactions, which are a special case of the open nested transaction paradigm, has further confirmed the viability of such a transaction model [Wei87a, Sch90a, Wei90].

The open nested transaction model seems to fit well with application systems that consist of a variety of cooperating tools. We assume that such tools cooperate with the DBMS by means of a well-defined collection of (open nested) transactions. For COMFORT, this means that while we cannot predict the behavior of end-users like engineers, there is still a good chance to analyze and exploit the characteristic properties of the workload that is generated by the involved tools. Note that it is not our goal to provide tuning for the application tools; rather we aim at making the DBMS more cooperative by tuning it to the workload characteristics of the application.

COMFORT is supposed to cover both traditional and non-traditional applications. While tuning is hard for conventional transaction-processing workloads such as reservation systems, it is even harder for advanced applications such as office automation or CIM. Such evolving applications pose higher demands on the tuning of the underlying database system, since it is even more crucial to exploit the application semantics in order to achieve the required performance. We plan to evaluate the results of the COMFORT project against performance-critical transactions in the area of banking and electronic stock-trading applications [PRS88] as well as call (or mail) routing and directory services in ISDN communication networks [PLZ89], on the one hand, and in an "active" office automation environment as well as for an MRP subsystem in the CIM area, on the other hand.

2.2 Hardware Technology

While COMFORT is supposed to provide automatic DBMS tuning capabilities on state-of-the-art commodity hardware, an important additional goal is to utilize recent and expected near-future technological advances. We believe that the following technologies will have a major impact on the architecture and performance of next-generation database systems (see also [Lag]):

- the availability of commercially viable shared-memory systems with several dozens of processors (e.g., [Seq]),
- fast progress in optical interconnect technology (e.g., FDDI [Th89]) as a communication basis for large-scale distributed systems,
- a trend towards disk-arrays [Kim86, PGK88, RB89], providing high bandwidth and improved seek times due to a high number of disk arms,
- safe RAM [Co89] and non-volatile disk caches [Sm85, Gro85],
- the advent of optical and magneto-optical disks as a mass storage device [Fu84], and
- increasing memory sizes with only minor improvements in memory latency [CKB89, GLV84, DeW84].

It seems that a number of file systems are already capable to benefit at least partially from such enhanced hardware (e.g., [DS89, Ga88, Ka89, Ru89]. However, database systems are typically not in a position to utilize novel hardware that rapidly. Note that "mature" relational systems such as Oracle, Ingres, and Sybase started taking some advantage of multi-processor systems only recently. The reason for the relatively slow progress of database systems is that their architecture is more complex than a file system and they provide more powerful and semantically richer functions. So the challenge obviously is to take into account both advanced technologies and (the semantics of) advanced applications. However, since OSs offer services to a much broader class of clients than just DBMS applications, it is likely that OSs can get faster returns from utilizing novel hardware. Therefore, one may argue that DBMSs will always lag behind OSs in taking advantage of hardware enhancements. Ultimately, this demonstrates the necessity for a better cooperation between the OS and the DBMS [Lag].

2.3 Integration with other Projects

Since we want to explore the tuning of traditional as well as non-traditional applications, we use the DASDBS prototype as the underlying database management system in the COMFORT project. DASDBS supports complex objects (i.e., nested relations and references) as well as complex transactions (i.e., multi-level transactions) [Sch90a]. Moreover, it already provides a variety of performance tuning hooks and can be extended relatively easily so as to fit with COMFORT. Using DASDBS as our run-time engine should, however, not rule out that the results of COMFORT can be transfered to other advanced DBMSs.

While COMFORT can be viewed as a fairly ambitious project by itself, it is also an essential part of the COSMOS project at ETH [Sch90b]. COSMOS, which stands for Cooperative System for the Management of Objects, can be viewed as the successor of DASDBS. It aims at a novel DBMS architecture that facilitates a better cooperation between the DBMS and its environment, including both the people that interact with the DBMS (i.e., the DBA, the application developer, and the end-user) and the tools and various subsystems that, together with the DBMS, constitute an application system. The issues that are addressed include: 1) coupling a complex-object-oriented DBMS with a variety of programming languages, 2) "objectifying" data for function shipping and data streaming in a heterogeneous DBMS environment, and 3) taking more advantage of OS and network-management facilities rather than reimplementing most functions in the DBMS. We expect COMFORT to make a contribution to the COSMOS project especially in the third area, i.e., improve the cooperation of DBMSs and OSs.

COMFORT has some ties also to the COCOON project* at ETH [SS89], which is the object-model & object-management part of COSMOS. Since COCOON is being implemented on top of DASDBS, it is obviously another candidate for verifying (and hopefully demonstrating) the automatic tuning capabilities of COMFORT.

* COCOON stands for COCOON ... Complex-Object-Orientation based on Nested Relations. The project is based on recursively nested relations.

3. COMFORT Architecture

3.1 The Big Picture

COMFORT can be viewed in different ways: as an expert system for database system performance tuning [Ab89], as some kind of automatic DBA, as a database system generator [Ca86b], or as a "context-sensitive" transaction program compiler. We think that the automatic DBA and the transaction program compiler views are the most appropriate ones, for we want to stress the fact that most tuning decisions should be made automaticallly at compile-time in the envisaged COMFORT environment.

The overall architecture of COMFORT is shown in Figure 1. COMFORT consists of two major components:

- the *Adaptive Data Manager*: a flexible run-time system that accepts all sorts of tuning parameters and hints, and
- the *Transaction Compiler*: a compile-time analyzer that makes intelligent tuning decisions.

The compile-time decisions are based on information about the database schema and other semantic invariants (including behavioral aspects), the access characteristics of the transaction programs, statistics about the data and the workload, and the underlying DBMS/OS and hardware architecture. This information is managed in a special area called the *Tuning Information Pool* (TIP), which may be viewed as an extension of the data dictionary. The Transaction Compiler extracts tuning tips from these metadata.

At run-time, tuning parameters may be re-adjusted based on more precise or more recent information. Ultimately, we envisage our run-time platform to provide a collection of self-optimzing algorithms that are guided by the hints of the Transaction Compiler. We plan to approach such a run-time architecture by enhancing DASDBS with adaptive strategies for its various modules.

The tuning decisions that are made by the Transaction Compiler affect the dynamic workload and, possibly, even the access characteristics of the compiled transactions. These dynamic modifications of the system behavior should in turn be reflected in the Tuning Information Pool, which is the main input for the compile-time decisions. Thus, COMFORT is actually a feedback loop, in which the stability of decisions is a critical issue that must be addressed in the project.

Changes of the workload such as adding a new transaction program to the application can raise the need for recompilation, for correctness or performance reasons. In this case, the challenge is to avoid recompiling all programs, that is, to make COMFORT work incrementally.

3.2 Module Cooperation

Even though COMFORT strives for automatic performance tuning at a global, i.e., system-wide level, we strongly believe in a modular DBMS architecture, an extensible tool-

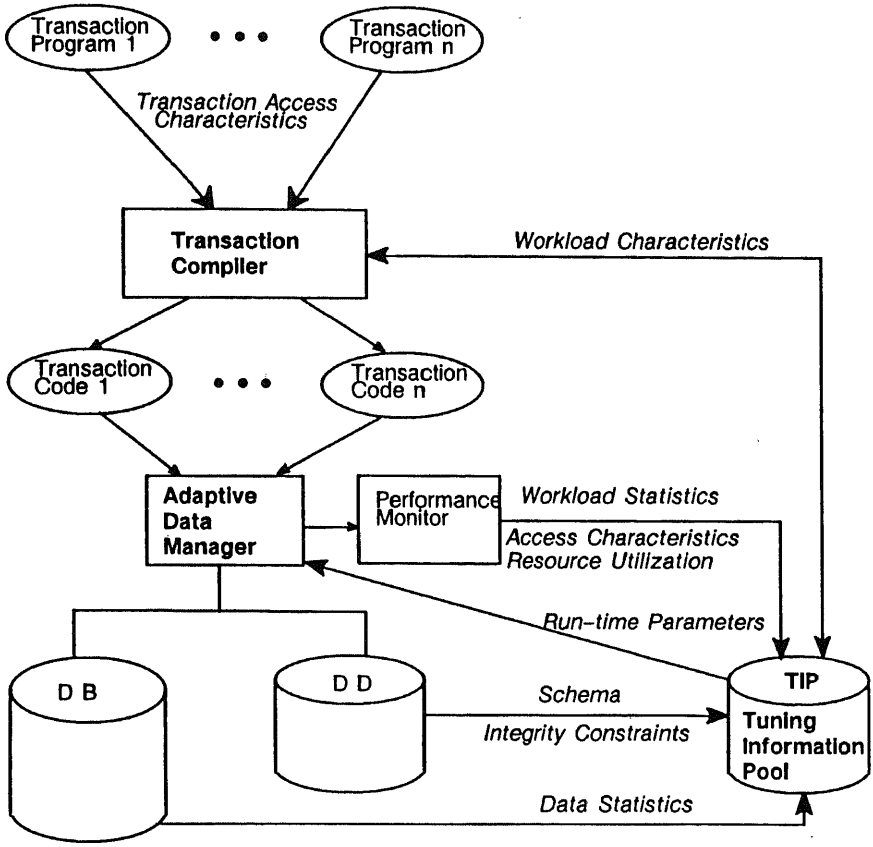

Figure 1: COMFORT Architecture

box architecture being the ideal [Bat88, Ca86b, Ha90, Sch90a]. Therefore, it is one of our goals to identify "tuning dependencies" between modules and organize our Adaptive Data Manager in such a way as to avoid unnecessary interaction between modules. Our initial approach to this architectural problem is illustrated in Figure 2.

For each main module, Figure 2 shows (a non-exhaustive list of) the most important *tuning parameters* (marked with a bullet), *load data* that characterize the workload independent of the tuning decisions or target system (marked with a downward arrow), the resulting (i.e., observed or estimated) *performance data* (marked with an upward arrow), and "*tuning dependencies*" from other modules. The performance data shown in Figure 2 are classified according to whether they can be obtained (or estimated) at compile-time (C) or run-time (R), and whether they are exact (E), statistical (S), or heuristic (H), i.e., "guesstimated" values. The load data that are annotated with a C are supposed to be determined by the Transaction Compiler or provided explicitly in an external "workload

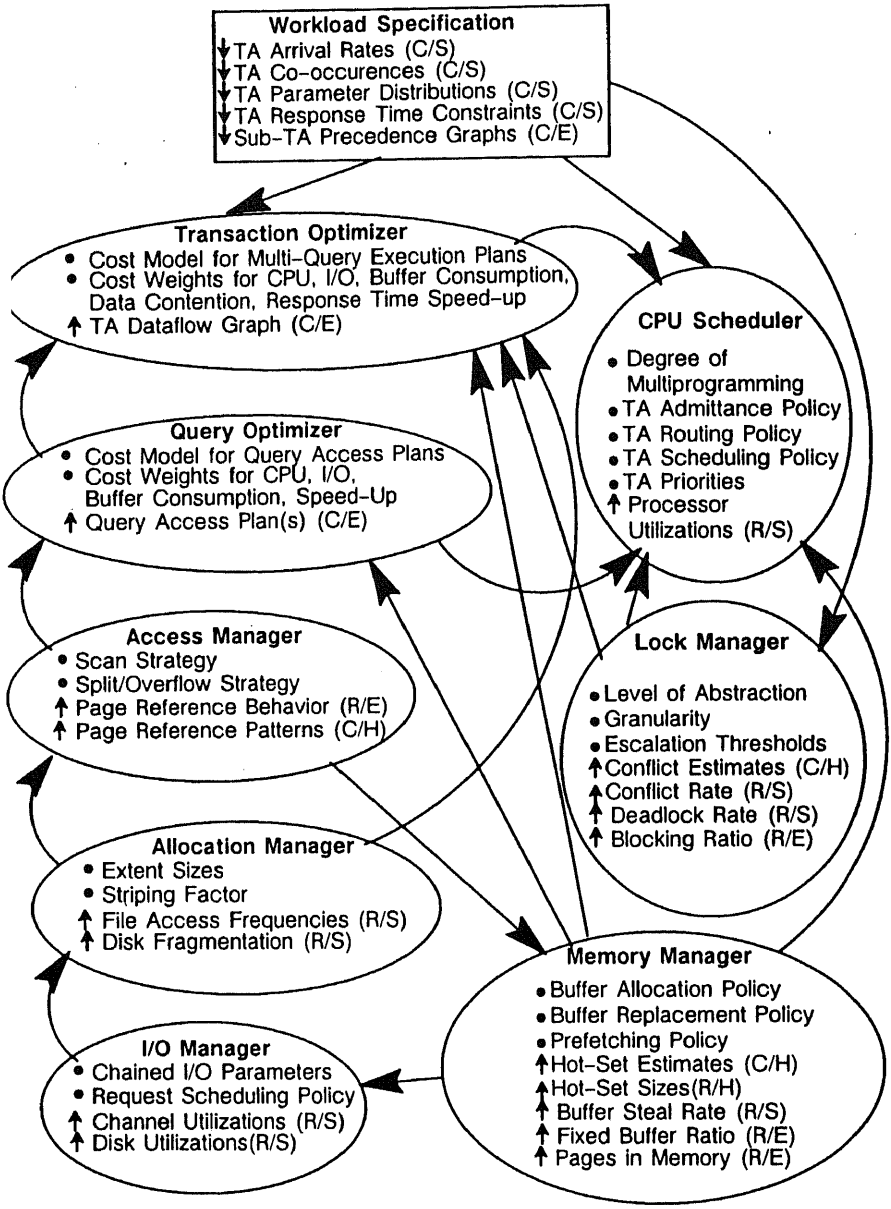

Figure 2: Tuning Dependencies between DBMS Modules

specification". While Figure 2 is obviously rather sketchy, it clearly demonstrates that modularizing performance tuning is a non-trivial problem.

Ideally, a "dependency" would be a function from a collection of performance data to the range of possible values of a particular tuning parameter. However, since there is no performance model that is able to predict the behavior of a complex DBMS, we plan to approximate the "dependency functions" coarsely.

4. Research Directions

COMFORT is supposed to address all performance-relevant issues of advanced database systems. Particular focus, however, will be on issues that are expected to become performance-limiting factors with a wider use of advanced technologies. Currently, we concentrate on the following issues:

- low-conflict low-overhead *parallelism* to utilize multi-processors effectively,
- intelligent *data allocation and migration* to optimize data access in a storage hierarchy consisting of main memory, disk-arrays, and optical disks, and
- intelligent *resource management* to coordinate the policies for load control, CPU scheduling, and memory allocation with the transaction manager.

We discuss our approaches to each of these issues in the following three subsections. In each subsection, we first present a general outline of the problem area, i.e., sketch a top-down view, and then illustrate our approach by discussing a concrete example problem in more detail, i.e., show a particular bottom-up view.

4.1 Parallelism

Parallelism in database systems aims at both increasing transaction throughput (*inter-transaction parallelism*) and decreasing transaction response time (*intra-transaction parallelism*). In addition, intra-transaction parallelism may also improve throughput, for locks are held for a shorter duration. Both directions are investigated in two complementary subprojects, described in the following.

4.1.1 Inter-Transaction Parallelism

Database systems can benefit from a parallel system only to the extent that data contention is not the performance-limiting factor. Therefore, we believe that concurrency control as a means for avoiding and controlling data contention will play a crucial role in systems with many processors. While data contention can also be reduced by carefully designing applications [St86, In88, RTIb, RTIc], we believe that eliminating concurrency-control hot spots should not be the responsibility of application developers. Similarly, the strict separation of online transactions and batch processing that is often dictated by data contention problems adds to the complexity of application design [Gr85]. To alleviate this sort of problems, semantic concurrency control techniques [Gar83, VV88] such as exploiting state-dependent commutativity [O'N86, We88] may prove useful and practically important in a wide area of applications.

Moving concurrency control to a higher level of abstraction, however, means that "short locks" or "latches" on pages are necessary in addition, in order to serialize high-level actions on the same low-level objects. Moreover, the locking information itself must be kept consistent in the face of parallel lock requests. Thus, we have to deal with at least the following three sources of data contention:

- high-level locks that are held for the duration of a transaction,
- low-level locks that are held (at most) for the duration of a (subtrans-)action, and
- control semaphores that are held for the duration of accessing DBMS-internal control information.

The third sort of data contention may in turn cause memory and bus contention and reduce cache effectivity in a shared-memory multi-processor architecture, unless the DBMS-internal structures and low-level algorithms are designed carefully [GT89].

Multi-level transactions [BBG89, BSW88, MGG86, Wei87a, Wei87b, Wei90], as a special case of open nested transactions, are a framework for designing and implementing low-conflict concurrency control methods which exploit the semantics of multiple levels of abstraction and can thus deal uniformly with the first two sources of data contention. In the OPERA project*, the concept of open nested transactions and the implementation techniques that we have developed so far are being extended towards workload-customized low-overhead "compiled concurrency control protocols", distributed systems, and "cooperative" transaction models [Jo88].

Example Problem: Compiled Concurrency Control Protocols

Even though there is largely consensus that multi-level transactions are a good conceptual basis for high-performance transaction processing, a major criticism has been the overhead of managing explicit "heavy-weight" subtransactions. As a response to this criticism, one goal of COMFORT is to avoid the overhead by distinguishing, at compile-time, cases which require full-fledged subtransactions from cases in which subtransactions can be implemented in a cheaper way, for example, by latching, i.e., short-term locking that does not check for deadlocks nor guarantees fairness.

Short Locks vs. Latches

As a performance enhancement for conventional record locking, it is suggested in [Mo89] that "latches" be used instead of "short locks" while a record operation accesses pages. The main differences are that latches are directly addressable, e.g., as an additional entry in the page header, rather than being stored in a hash table with dynamically allocated entries, and that latching does not check for deadlocks and disregards the (unlikely) possibility of starvation.

Following the direction of "light-weight" subtransactions, we can easily adopt the idea of storing lock control blocks in page headers (see also LC89]. However, turning off deadlock detection for subtransactions can not be so easily generalized to an advanced data-

* OPERA stands for Extending Open Nested Transactions.

base system. Unlike conventional record operations that usually access one data page and a number of index pages in a well-defined order, complex operations on complex objects may access many pages, in a less predictable order. Hence, the subtransactions are susceptible to page-level deadlocks, so that simple latching is not feasible. While it is possible to carefully design deadlock-free latching protocols for certain types of subtransactions, e.g., for B⁺-tree operations [ML89], such a manual approach seems to be too cumbersome for complex access patterns. Therefore, we plan to investigate conditions for partially compiling a multi-level locking protocol to a latch-based protocol.

Locking Levels, Granularities, and Protocols

The idea of compile-time optimization can be pushed even further by analyzing the access characteristics of transaction programs and determining the optimal granularity and abstraction level of concurrency control. This means that COMFORT would automatically pick the best method from alternatives such as 1) relation locking vs. tuple locking vs. field locking, and 2) page locking vs. tuple & index-entry locking vs. predicate locking. In both of these design dimensions, potential parallelism can be traded for reduced overhead [Wei87a, He89]. At run-time, compile-time decisions in favor of fine granularities or a high level of abstraction can be adjusted by setting appropriate lock escalation thresholds [DB2]. These thresholds should again be chosen automatically in the COMFORT architecture.

As a next step, the applicability of non-strict 2PL and non-2PL protocols (e.g., [Mo85, Bay86, SGA87] and the question of whether certain locks have to be set at all may be analyzed at compile-time. Transaction preanalysis for concurrency-control purposes was applied already in the SDD-1 system in the seventies [BSR80]. While it seems that this approach has almost been forgotten since then (with the exception of a few, mostly theoretical papers [LSW86, EW88]), we believe that its leverage increases with the level of abstraction on which locking is employed. This means that, for example, predicate locking could benefit more from a compile-time conflict analysis than page locking, e.g., by compiling predicate conflicts into scheduling directives. Our ultimate goal is to automatically generate "compiled concurrency control protocols" that combine the ease of system-provided transaction management with the efficiency of hand-coded critical sections.

4.1.2 Intra-Transaction Parallelism in Nested Transactions

Multi-level transactions as a special case of nested transactions have been intensively studied for high-performance inter-transaction parallelism. In the PLENTY subproject*, we investigate how this approach can be extended so as to handle also intra-transaction parallelism, i.e., parallel execution of multiple DBMS operations within the same transaction program and/or employing parallel algorithms for single DBMS operations. Such intra-transaction parallelism can be based on data partitioning as well as operation decomposition and pipelining [Bo90, Bue89a, Bue89b, DeW86, DGS88, Lo89, Reu86, Wi89].

* PLENTY stands for Parallel Execution of Nested Transactions on Plenty of Processors.

The main goal of PLENTY is to explore under which conditions the increased parallelism is worth the additional overhead [Bo88]. In particular, we want to gain a better understanding of how parallelism affects data contention in the three categories introduced in Section 4.1.1. On the one hand, parallelism may reduce data contention, because high-level locks are held for a shorter duration if the transaction response time can be effectively decreased. On the other hand, parallelism may increase data contention on short-term locks and control semaphores, simply because the load is potentially higher in a multi-processor system. Thus, it is an open question at what point parallelism reaches its limits.

Our approach to trading off parallelism vs. run-time overhead is based on two steps. In the first step, a transaction is analyzed and *decomposed* into subtransactions according to the partitioning of data, the algorithmic precedences of the transaction program, and the knowledge of the DBMS-internal implementation of the invoked operations. This analysis step yields a nested transaction structure that reflects as much potential parallelism as possible. In the second step, the degree of parallelism in a nested transaction is *reduced* so as to take into account the run-time overhead, and an actual run-time execution structure is synthesized. This is done by combining subtransactions (e.g., siblings) for which a parallel execution would probably not improve the transaction's response time nor the overall transaction throughput.

Note that the sketched two-step procedure is supposed to be applied at the compile-time of a transaction program, in accordance with the overall approach of COMFORT. PLENTY strives for the automatic generation of an "optimal" set of subtransactions for transactions with data-intensive operations such as multi-key queries that evaluate multiple indexes, computing transitive closures, relational joins (with low-selectivity filters), or maintaining derived data on updates. The crucial point here is to come up with a (heuristic) cost model for the "reduction" of a nested transaction.

Further issues that must be addressed are 1) examining what applications and kinds of operations justify the effort of intra-transaction parallelism, 2) studying and developing algorithms for DBMS operations that can be effectively parallelized (e.g., for joins [SD89, MR89] or operations on complex objects [Du89, HHM86, HSS89, Ze89], aiming at both I/O and CPU parallelism, 3) the load balancing in assigning subtransactions to processors and distributing the I/O load across multiple disks, and 4) the effective utilization of and the requirements for OS support such as synchronization primitives, scheduling directives, or even transaction management facilities.

The PLENTY subproject is based on a Sequent Symmetry shared-memory multi-processor; however, we also want to investigate how our approach can be generalized to a distributed(-memory) system. As the DBMS platform, the DASDBS prototype is considered to be appropriate for the following reasons. DASDBS provides storage structures for nested relations that can be extended so as to take advantage of I/O parallelism. Both horizontal and vertical partitioning of (sets of) objects are possible, and even system-controlled redundancy can be incorporated. Moreover, DASDBS already supports multi-level transactions in a way that allows also parallel subtransactions within the same appli-

cation transaction. Thus, we can explore various forms of intra-transaction parallelism without having to reconsider synchronization issues. To achieve effective improvements with respect to response time, parallel subtransactions are executed as separate (light-weight) processes on a multi-processor.

Example Problem: Parallel Execution Strategies for Multi-Key Queries

We believe that multi-key queries like "Select * From R Where A_1 = $Value_1$ And ... And A_n = $Value_n$" are gaining practical importance for decision-support transactions. In state-of-the-art DBMSs, the only way of accelerating such queries is through B^+-tree-like indexes for (a subset of) the attributes A_1, ..., A_n [Schn83] (or, similarly, stored bitmaps for attribute values that occur frequently [O'N87]). We are investigating how multi-key queries can be accelerated on a shared-memory multi-processor system, making the following assumptions on the underlying storage model and transaction management:

- The relation R is declustered, i.e., spread across multiple disks, based on a partitioning of its primary-key range.
- B^+-tree indexes are defined for the attributes A_1, ..., A_k ($k \leq n$); all of them are non-unique indexes. For the sake of simplicity, we assume that the order A_1, ..., A_k reflects the order of index selectivities. That is, the index for A_1 is the most selective one, where selectivity is defined as $\frac{number\_of\_different\_Ai\_values\_in\_R}{number\_of\_tuples\_in\_R}$.
- For transactions that invoke multi-key queries, the "repeatable-read" property is guaranteed, i.e., phantoms are prevented, by predicate locks or index-key locks. These "long locks" are held until EOT. While accessing an index, "short locks" are acquired in order to serialize concurrent operations on the same B^+-tree. Such a multi-level concurrency control requires either 2PL on B^+-tree pages for the duration of the index access, or a customized tree locking protocol which may be implemented with latches [ML89].

Sequential Strategies

In sequential systems, common strategies for processing a multi-key query are the following:

(A) Use only the most selective index, i.e., the index for A_1, and filter the qualifying tuples for the "remainder predicate" A_2 = $Value_2$ And ... And A_n = $Value_n$.

(B) Use the j most selective indexes ($j \leq k$) such that the *estimated combined selectivity*, computed as the product of the selectivities of A_1, ..., A_j, is so low that accessing the qualifying tuples is cheaper, i.e., probably requires less page accesses than using an additional index. Then, compute the intersection of the pointer lists obtained from these j index searches, and finally filter the qualifying tuples for the "remainder predicate" A_{j+1} = $Value_{j+1}$ And ... And A_n = $Value_n$.

(C) Proceed as in strategy (B); however, compute the intersection of pointer lists after each index search and terminate the index phase as soon as the *effective combined selectivity*, i.e., number of tuples or different pages in the pointer-list intersection does not justify further index searches.

Note that the inflexible strategy (A) may be advantageous only in DBMSs that bundle together the use of an index and locking the accessed index key. Thus, using multiple indexes could result in excessive index locking. This is because all keys will be locked, even though, in principle, locking only one key would be sufficient as a superpredicate of the conjunctive predicate, and the key to be locked could even be chosen arbitrarily out of the accessed index keys.

Strategy (B) makes the implicit assumptions that the distribution of attribute values is uniform for each attribute and the distributions of different attributes are independent of each other. Thus, even though strategy (B) is still correct if distributions are skewed or attributes are correlated, its effectivity crucially depends on the data characteristics. In contrast, strategy (C) adapts itself to the effective selectivities but may require (slightly) more run-time overhead than (B).

Parallelization of Strategies

As for the parallel processing of a multi-key query, it seems that only strategy (B) can be directly decomposed into parallel execution steps. Obviously, this strategy allows accessing the j used indexes in parallel, and it bears potential parallelism in the computation of the pointer-list intersection.

While strategy (C) cannot be parallelized directly, it can be combined with strategy (B) so as to take into account the potential discrepancy of estimated and effective combined selectivities. The resulting hybrid strategy is illustrated in Figure 3, which shows the precedence graph of the necessary execution steps for a multi-key query. The main feature is that the parallel index phase of strategy (B) is repeated until the effective combined selectivity becomes sufficiently low; then the final filter phase is entered. In the i^{th} index phase, ν_i indexes are selected out of those indexes that have not been used in the previous i-1 index phases. These ν_i indexes can be accessed in parallel, and the retrieved pointer-lists are intersected with the result of the $i\text{-}1^{st}$ index phase. Furthermore, if response time is very critical, e.g., in a real-time system, prefetching the result pages of the i^{th} index phase while proceeding with the $i+1^{st}$ index phase may be worth the addtional I/O costs and buffer consumption [KWZ89].

Tuning Problems

At a first glance, it may seem that incorporating such a parallel execution strategy in a DBMS is more or less straightforward and does not impose any significant tuning problems. At a closer look, however, it becomes obvious that the selection of indexes in each of the index phases is a fairly challenging problem for a parallel query optimizer. The quality of the index-selection decision for the i^{th} index phase depends on at least the following factors:

- the estimated selectivity of an index,
- the effective combined selectivity of the intermediate conjunctive predicate before and after the i^{th} index phase, which in turn depends on the attribute-value distributions and the correlation of multiple attributes,

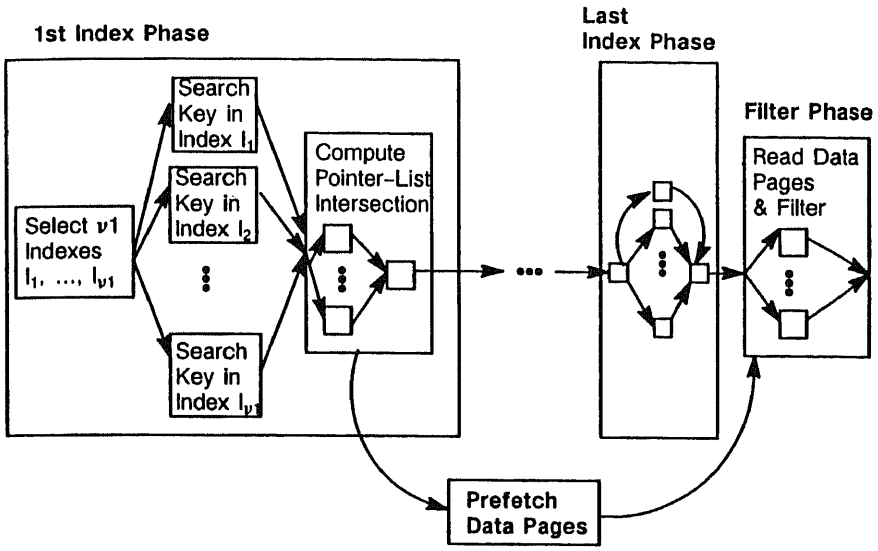

Figure 3: Parallel Execution Strategy for Multi-Key Queries

- the number of I/Os for the index access, which in turn depends on the height of the B⁺-tree and the index-specific buffer miss rate, i.e., ultimately, on the size and the access frequency of the index,
- the potential for parallel I/O among the selected indexes, which in turn depends on the disk placement of the indexes and the heat, i.e., overall I/O frequency of the various disks, and
- the contention for "short locks" on index pages, which in turn depends on the update frequencies of attributes.

Note, in particular, that it may sometimes be better not to use an index contrary to selectivity arguments, if one of the following conditions holds: 1) the buffer miss rate of the index is high and the disk(s) on which the index is located is heavily accessed, whereas the disk load for the R tuples is low and well balanced across multiple disk, or 2) the index is a concurrency-control hot spot, i.e., updated very frequently, whereas contention for (short) locks on data pages is evened out across many pages. We are currently building a testbed for quantifying these performance issues and exploring tuning rules for parallel multi-key queries. We further plan to extend our scope to multi-key queries using text indexes [DPS83] or customized multi-key access paths such as signatures [CS89].

4.2 Data Allocation and Migration

In currently available database systems, data allocation primarily aims at clustering data that is frequently accessed together in physical proximity on disk. Clustering is usually schema-driven [SPS87]; clustering on a per object basis can be achieved only through

clever use of the bulk loader. For the purpose of I/O load balancing, some DBMSs further support "striping" [SG86] or "declustering" [LKB87, Co88], i.e., spreading data collections across multiple disks, usually based on horizontal partitioning. Beyond these tuning opportunities, data placement is fairly straightforward today. Memory is usually too scarce to make entire data collections resident; so all data typically lives on disk, and simple "on-demand" fetching and LRU-like strategies are mostly employed for data migration between disk and memory and vice versa [TG84].

In future systems, larger memories will allow making very frequently used data memory-resident, and very infrequently used data will reside on optical disks rather than being off-line. Data migration within the storage hierarchy is likely to become a crucial issue then. For example, migration between fast-access disks and archival store such as optical disks should be fully automatic, driven by application-specific probabilities of when certain data will be reaccessed. Similarly, conventional buffer management should be enhanced to take into account the effects of hardware disk caches on the one hand, and to exploit the semantics of advanced applications on the other hand [CK89]. Novel evolving applications will shed a different light also on the clustering vs. declustering tradeoff, which is not really well understood even for classical applications. Finally, another tradeoff that needs thorough investigation is optimizing read I/Os vs. optimizing write I/Os, e.g., whether one should employ an extent-based or a log-based allocation strategy [OD89]. In any case, automatic online reorganization that operates in the background is highly desirable for maintaining the required performance level.

Example Problem: Incremental Optimization of Data Placement

An Adaptive Storage System

In the FIVE subproject*, we have implemented a low-level file system that provides flexible data-placement options, i.e., capabilities for both clustering and declustering. *Files* are simply (arbitrarily long) bytestrings, stored in a collection of *blocks*, as in Unix. Unlike Unix, files can be spread across multiple *volumes*, i.e., (logical) disk drives. Similar to [St88], space for a file is allocated in two-dimensional *regions*, where

- the "width" w of a region is the striping or declustering factor, i.e., the number of disks across which the region is spread, and
- the "height" h of a region is the clustering factor, i.e., the number of contiguous blocks (referred to as an *extent*) that are allocated on each of the w disks.

Figure 4 shows the space allocation for a file consisting of two 2x3 regions on three disks. The block numbers that are shown are the logical block numbers that are visible to the clients of FIVE.

When space is allocated, i.e., when a file is created or its physical space needs to be extended, FIVE accepts the following types of tuning hints:

* FIVE stands for File System with Adaptive Enhancements. As one might guess, FIVE is based on five concepts: volumes, blocks, files, extents, and regions. Also, if you've gotten a headache of a file-system problem, just take FIVE.

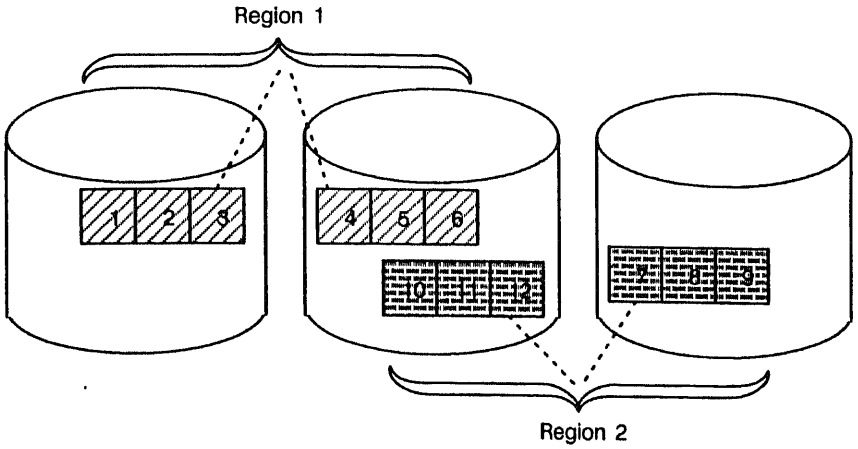

Figure 4: Space Allocation based on Two-dimensional Regions

- hints on the size and dimensions of regions,
- affinity hints, i.e., tentative block numbers near which space should be allocated, and
- anti-affinity hints, i.e., block numbers that identify disk drives that should not be used for the allocation.

Such hints can be either advisory tips or obligatory directives. For a particular allocation, FIVE can exploit both clustering and declustering through set-oriented I/O [Wei89a] for (nearly-)sequential disk accesses and parallel multi-disk I/O, respectively.

Tuning Problems

Initially, we restrict our scope to requests that access either logically contiguous portions of a file or an entire file, the latter being the dominant part of the workload. Hence, our optimization criterion for the placement of files is to minimize the average access time for reading an entire file under the constraint that the overall disk load should be roughly balanced, i.e., the utilization of the hottest disk should not exceed a certain threshold.

Obviously, this optimization problem is related to the data placement problem that has been addressed in the Bubba project for I/O load balancing [Co88]. There, the problem was to decluster a given set of files and place the resulting file fragments on a given number of disks such that the "accumulated heat" is approximately the same for each disk, where "accumulated heat" means the sum of the access frequencies of the fragments that are stored on a disk. The algorithm developed for Bubba assumes that the access frequencies and the optimum declustering factor of each file are given or have been determined by an analytic performance model, respectively. In contrast to this scenario, our problem is considerably harder in the following respects:

- File access frequencies are not known in advance. We may expect some client-provided hints at the creation-time of a file, or rely on statistical values, e.g., the average heat of all existing files of a particular type, or estimates derived from transaction arrival rates and an analysis of transaction programs. In addition, we can collect access-frequency information on a per file basis while transactions are being processed, and use this information for occasionally re-optimizing the space allocation of a file.
- As files are created and extended dynamically, we have to optimize data placement *incrementally*. Thus, "sub-optimal" allocations must be acceptable, too, e.g., because of disk fragmentation.
- From the above two points it follows that reorganization, i.e., reallocation of file regions, is a must. In accordance with the COMFORT philosophy, such reorganizations should
  - be initiated automatically (see [Sa89] for an approach to automatic reorganization in the Unix system),
  - be able to process files incrementally, i.e., reorganize individual files or divide bigger reorganizations into multiple steps, each of which can be executed as a separate transaction, and
  - operate online in background mode, i.e., run concurrently with foreground transactions, but probably at a lower priority [Wei89b].

  Such automatic reorganizations involve continuous self-monitoring (i.e., collecting information about file access frequencies, co-access frequencies for multiple files, co-access frequencies for multiple blocks of the same file, etc.), and carefully trading off the expected benefits of a reorganization vs. the costs of the reorganization itself [TF82].

To improve our understanding of the problems and tradeoffs in this incremental data-placement optimization, we are about to start a series of performance experiments with the tuning parameters of FIVE, using workload characteristics of an office document filing system [ZPD90].

4.3 Resource Management

The fact that future transaction-processing systems will be provided with much more MIPS and memory than what is typically available today does not mean that resource management will be any simpler. Rather we are convinced that, even with increasing resources per workload unit, resource management will become more sophisticated for two main reasons. First, the resource appetite of complex applications is likely to increase quite significantly. Secondly, more available resources may open up benefits of smarter algorithms for CPU load control and memory management, whereas vanilla algorithms like FCFS scheduling and LRU buffer replacement can hardly be outperformed in a situation with very limited resources. In addition, the data availability itself has to be considered as another valuable resource in a highly parallel environment. On the other hand, the large amount of resources that need to be managed may justify more overhead for the resource management itself, e.g., dedicating a processor to scheduling decisions.

In the RAPIDS subproject*, we want to develop an approach toward "intelligent" resource management, which coordinates the policies for managing different types of resources so as to maximize transaction throughput. For example, an intelligent scheduler should dispatch a complex transaction with particular memory requirements only if the chances are good that a certain amount of memory is indeed available [CD85, CJL89]. As another example, consider a long-running, batch-like transaction that has low priority, but holds a large number of locks or a lot of memory. It may be a wise scheduling decision to give this transaction a very high priority or vary its priority dynamically, in order to complete the transaction and release its locks as soon as possible [Dei89]. A heuristic rule for choosing the priority could be based on the probability that other transactions are blocked.

Example Problem: Conflict-driven Load Control for the Avoidance of Data-Contention Thrashing

Load Control in DBMSs is necessary for dealing with overload situations, that is, preventing the system from thrashing. Without load control, load peaks can lead to a disastrous deterioration of both transaction throughput and response time. Such thrashing situations or "black hole phenomena" (as response time approaches infinity) occur because of resource contention, especially with regard to memory, or data contention. In the following, we consider only data-contention thrashing (*DC thrashing*), that is, performance deterioration caused by excessive lock conflicts [BHG87, TGS85, ACL87].

The Need for Conflict-Driven Load Control

In both the literature and currently available DBMSs, the only mechanism for the avoidance of DC thrashing is to limit the *degree of multiprogramming (DMP)*, that is, the maximum number of transactions that are concurrently active in the DBMS. The DMP is set by the database administrator as a system startup parameter, and usually remains constant until the system is shut down. Unfortunately, this vanilla approach to DBMS load control has several severe shortcomings:

- There are applications for which the DMP is an extremely sensitive parameter. That is, the band of reasonable settings of the DMP is sometimes very narrow [JTK89]. If the DMP is set too low, then the available hardware resources are underutilized and the transaction load is not processed fast enough. If the DMP is set too high, then the system is susceptible to DC thrashing.
- The optimum DMP is highly dependent on the types of transactions in the workload. Obviously, the number of concurrent transactions that a DMBS can accept without suffering DC thrashing is higher for short read-only transactions than for long update transactions. Since real-life applications often consist of a fairly broad mix of transactions, globally limiting the DMP regardless of the transaction type seems to be a sledgehammer solution.
- High-performance DBMS applications such as reservation systems typically circumvent the problem of dealing with a variety of transaction types. In such systems, it is

* RAPIDS stands for "Resource and Processor Management based on Intelligent DBMS Scheduling". Note that, for the experienced river runner, rapids are an opportunity for rapid progress; for the unexperienced, they bear the risk of capsizing.

common practice to allow only "designed transactions" to access "hot" data, i.e., frequently used data [In88]. For example, a rule of thumb in online transaction processing is that queries should not access more than ten records [In88]. In practice, this implies that only such queries are allowed that can use a unique index or a non-unique index with very high selectivity. A related rule of thumb is to split long transactions into multiple short ones [In88], sometimes even at the expense of consistency [GS84, RTIc, Ora]. The "nice" property of such constraints and "transaction design rules" is that they produce a more or less homogeneous workload of "rifle-shot" transactions. The flip side of the coin is that this convenient workload characteristic is achieved through a shotgun marriage of DBMS performance tuning and database application design. Such an approach is clearly incompatible with the paradigm of data independence, i.e., separation of functionality and performance.

Our conclusion from these points of critique is that DBMS load control should not be based on simply setting the global DMP. We rather argue that automatic load control for DC thrashing avoidance can be better accomplished if one uses some notion of conflict rate as the control variable.The DMP is a control variable also for resource contention; however, the conflict rate directly measures data contention. So, to avoid DC thrashing, one should actually pay direct attention to the conflict rate. In particular, there is evidence that limiting the conflict rate rather than the DMP yields satisfactory performance regardless of the transaction types in the workload.

A conflict-driven approach to load control could have the following advantages:

+ It is a step towards automating the following tuning decisions: What is the *"right" number* of transactions to be admitted to the system, and what is the *"right" mix* of transactions to be admitted to the system?
+ It could, to some extent, replace guesswork and intuition in DBMS tuning. Thus, it would help all those unlucky DBMS installations that do not benefit from the experience and wisdom of one of the few DBMS performance gurus in the world.
+ Finally, it could shed new light also on the undesirable union of system tuning and application design.

Our Approach

In the RAPIDS project, we pursue such a conflict-driven approach to DBMS load control, which consists of the following steps:

(1) Since there are various definitions of conflict rate, we investigate which definition works best as a load control variable. Possible definitions are, for example, $\frac{\#Lock\_Waits}{\#Lock\_Requests}$ in a particular time interval, or $\frac{\#Locks\_Held}{\#Locks\_Held\_by\_Nonblocked\_TAs}$ at a particular point of time, or $\frac{\#Lock\_Waits}{\#Lock\_Requests}$ for those transactions that are in the system at a particular point of time. Our expectation is that there exits at least one defintion of conflict rate such that its value reflects the current (DC) load level regardless of the number and types of active transactions.

(2) We investigate beyond which value of the conflict rate DC thrashing occurs. Our expectation is that there exists a "natural constant" like a *critical conflict rate*, which can be experimentally determined. This expectation is supported by the analytic results of Tay [TGS85].

(3) We are implementing an initial DBMS scheduling algorithm that admits new arriving transactions only if the current conflict rate is uncritical.

(4) We are developing various extensions to the initial scheduling algorithm. For example, it may be wise to admit new transactions in non-FCFS order based on the transaction types or further compile-time knowledge about the transactions' access characteristics.

(5) Along the same lines, transaction priorities could be incorporated [AG89, Bu89] and dynamically adjusted according to the current conflict rate and the "lock consumption" of active transactions, i.e., the number and duration of locks held. For example, it seems to be advantageous to give the highest priority to the transaction that holds the highest number of locks [Ham90].

(6) We plan to extend the scope of our approach by considering also resource-contention thrashing, especially the problem of overloading memory resulting in excessive buffer replacements [CD85, CJL89].

(7) Ultimately, we want to take advantage of compile-time knowledge about the access characteristics of transactions. Such transaction profiles could be used for making run-time scheduling decisions both less expensive in terms of overhead and more effective in terms of the quality of the decisions. For example, compile-time predictions of a transaction's lock requests and its hot sets with respect to page references can serve as an estimate of the relative increase of the overall conflict rate and buffer steal rate if the transaction were admitted to the system.

5. Outlook

COMFORT is an approach towards automating the performance tuning of advanced DBMSs. The approach is based on two main components:

- the Transaction Compiler, which is supposed to analyze transaction programs so that many tuning parameters can be determined at compile-time, and
- the Adaptive Data Manager, which provides the necessary flexible run-time engine for dynamic fine-tuning and overwriting of inappropriate compile-time decisions.

While this paper has mostly presented half-baked ideas rather than proven results, its main contribution is that it sets up a framework for automatic DBMS tuning. Moreover, we have discussed initial approaches to several performance problems in various directions of DBMS research.

The ideal solution to COMFORT would be a self-optimizing and self-adapting system that does not need any performance-tuning input other than the information gathered from the compilation and execution of the transactions. Such a solution would, of course, affect

the overall architecture of the underlying data management system quite drastically. Given our experience with the development of DASDBS, it is therefore intriguing to view a novel next-generation database system as the ultimate long-range goal of the project. We believe that understanding performance tradeoffs and system tuning is an ideal training area for building such a system. So, while we are going to use DASDBS as our initial system platform, we may eventually find ourselves shooting for a complete next-generation system in the long run. In fact, the COSMOS project in which COMFORT is embedded is an approach towards such a novel cooperative DBMS architecture.

References

[Ab89] Abramowicz, K., Knowledge-based Tools for the Performance Optimization of Object-oriented Database Systems (in German), 3rd GI Conf. on Database Systems in Office Automation, Engineering, and Scientific Applications, 1989, Springer, IFB 204

[ACL87] Agrawal, R., Carey, M.J., Livny, M., Concurrency Control Performance Modeling: Alternatives and Implications, ACM TODS Vol.12 No.4, 1987

[AG89] Abbott, R., Garcia-Molina, H., Scheduling Real-Time Transactions with Disk Resident Data, VLDB Conf., 1989

[Bat88] Batory, D.S., Barnett, J.R., Garza, J.F., Smith, K.P., Tsukuda, K., Twichell, B.C., Wise, T.E., GENESIS: An Extensible Database Management System, IEEE Transactions on Software Engineering Vol.14 No.11, 1988

[Bay86] Bayer, R., Consistency of Transactions and Random Batch, ACM TODS Vol.11 No.4, 1986

[Bo88] Boral, H., Parallelism and Data Management, 3rd Int. Conf. on Data and Knowledge Bases, 1988

[Bo90] Boral, H., Alexander, W., Clay, L., Copeland, G., Danforth, S., Franklin, M., Hart, B., Smith, M., Valduriez, P., Prototyping Bubba: A Highly Parallel Database System, IEEE Transactions on Knowledge and Data Engineering Vol.2 No.1, 1990

[Bu89] Buchmann, A.P., McCarthy, D.R., Hsu, M., Dayal, U., Time-Critical Database Scheduling: A Framework for Integrating Real-Time Scheduling and Concurrency Control, IEEE Data Engineering Conf., 1989

[Bue89a] Bueltzingsloewen, G.v., Iochpe, C., Liedtke, R.-P., Kramer, R., Schryro, M., Dittrich, K.R., Lockemann, P.C., Design and Implementation of KARDAMOM – A Set-oriented Data Flow Database Machine, Int. Workshop on Database Machines, 1989, Springer, LNCS 368

[Bue89b] Bueltzingsloewen, G.v., Optimizing SQL Queries for Parallel Execution, ACM SIGMOD Record Vol.18 No.4, 1989

[BBG89] Beeri, C., Bernstein, P.A., Goodman, N., A Model for Concurrency in Nested Transactions Systems, Journal of the ACM Vol.36 No.1, 1989

[BHG87] Bernstein, P.A., Hadzilacos, V., Goodman, N., Concurrency Control and Recovery in Database Systems, Addison-Wesley, 1987

[BR76] Blevins, P.R., Ramamoorthy, C.V., Aspects of a Dynamically Adaptive Operating System, IEEE Transactions on Computers Vol.25 No.7, 1976

[BR89] Bhargava, B., Riedl, J., A Model for Adaptable Systems for Transaction Processing, IEEE Transactions on Knowledge and Data Engineering Vol.1 No.4, 1989

[BSR80] Bernstein, P.A., Shipman, D.W., Rothnie, J.B., Concurrency Control in a System for Distributed Databases (SDD-1), ACM TODS Vol.5 No.1, 1980

[BSW88] Beeri, C., Schek, H.-J., and Weikum, G., Multi-Level Transaction Management, Theoretical Art or Practical Need?, 1st Int. Conf. on Extending Database Technology, 1988, Springer, LNCS 303

[Ca86a] Carey, M.J., DeWitt, D.J., Richardson, J.E., Shekita, E.J., Object and File Management in the EXODUS Extensible Database System, VLDB Conf., 1986

[Ca86b] Carey, M.J., DeWitt, D.J., Frank, D., Graefe, G., Muralikrishna, M., Richardson, J.E., Shekita, E.J., The Architecture of the Exodus Extensible Database System, 1st Int. Workshop on Object-oriented Database Systems, 1986

[Co88] Copeland, G., Alexander, W., Boughter, E., Keller, T., Data Placement in Bubba, ACM SIGMOD Conf., 1988

[Co89] Copeland, G., Keller, T., Krishnamurrthy, R., Smith, M., The Case For Safe RAM, VLDB Conf., 1989

[CD85] Chou, H.-T., DeWitt, D., An Evaluation of Buffer Management Strategies for Relational Database Systems, VLDB Conf., 1985

[CFW90] Copeland, G., Franklin, M., Weikum, G., Uniform Object Management, 2nd Int. Conf. on Extending Database Technology, 1990

[CJL89] Carey, M.J., Jauhari, R., Livny, M., Priority in DBMS Resource Scheduling, VLDB Conf., 1989

[CK89] Chang, E., Katz, R., Exploiting Inheritance and Structure Semantics for Effective Clustering and Buffering in an Object-Oriented DBMS, ACM SIGMOD Conf., 1989

[CKB89] Cohen, E.I., King, G.M., Brady, J.T., Storage Hierarchies, IBM Systems Journal Vol.28 No.1, 1989

[CS89] Chang, W.W., Schek, H.-J., A Signature Access Method for the Starburst Database System, VLDB Conf., 1989

[Dei89] Deitel, H.M., An Introduction to Operating Systems, Chapter 10: Job and Processor Scheduling, 2nd ed., Addison-Wesley, 1989

[De76] Denning, P.J., Kahn, K.C., Leroudier, J., Potier, D., Suri, R., Optimal Multiprogramming, Acta Informatica Vol.7, 1976

[De80] Denning, P.J., Working Sets Past and Present, IEEE Transactions on Software Engineering Vol.6 No.1, 1980

[DeW84] DeWitt, D.J., Katz, R.H., Olken, F., Shapiro, L.D., Stonebraker, M., Wood, D., Implementation Techniques for Main Memory Database Systems, ACM SIGMOD Conf., 1984

[DeW86] DeWitt, D.J., Gerber, R.H., Graefe, G., Heytens, M.L., Kumar, K.B., Muralikrishna, M., GAMMA - A High Performance Dataflow Database Machine, VLDB Conf., 1986

[Du89] Duppel, N., Gugel, D., Reuter, A., Schiele, G., Zeller, H., Progress Report #4 of PROSPECT, University of Stuttgart, 1989

[DB2] Cheng, J.M., Loosley, C.R., Shibamiya, A., Worthington, P.S., IBM Database 2 Performance: Design, Implementation, and Tuning, IBM Systems Journal Vol.23 No.2 (Special Issue on DB2), 1984

[DGS88] DeWitt, D.J., Ghandeharizadeh, S., Schneider, D., A Performance Analysis of the Gamma Database Machine, ACM SIGMOD Conf., 1988

[DI89] Devarakonda, M.V., Iyer, R.K., Predictability of Process Resource Usage: A Measurement-Based Study on UNIX, IEEE Transactions on Software Engineering Vol.15 No.12, 1989

[DPS83] Dadam, P., Pistor, P., Schek, H.-J., A Predicate-Oriented Locking Approach for Integrated Information Systems, IFIP Congress, 1983

[DS89] Dibble, P.C., Scott, M.L., Beyond Striping: The Bridge Multiprocessor File System, ACM Computer Architecture News Vol.17 No.5, 1989

[ELZ86] Eager, D.L., Lazowska, E.D., Zahorjan, J., Adaptive Load Sharing in Homogeneous Distributed Systems, IEEE Transactions on Software Engineering Vol.12 No.5, 1986

[EW88] Eich, M.H., Wells, D.L., Database Concurrency Control Using Data Flow Graphs, ACM TODS Vol.13 No.2, 1988

[Fu84] Fujitani, L., Laser Optical Disk: The Coming Revolution in On-Line Storage, Communications of the ACM vol.27 No.6, 1984

[Ga88] Gait, J., The Optical File Cabinet: A Random-Access File system for Write-Once Optical Disks, IEEE Computer Vol.21 No.6, 1988

[Gar83] Garcia-Molina, H., Using Semantic Knowledge for Transaction Processing in a Distributed Database, ACM TODS Vol.8 No.2, 1983

[Gr81] Gray, J., The Transaction Concept: Virtues and Limitations, VLDB Conf., 1981

[Gr85] Gray, J., Good, B., Gawlick, D., Homan, P., Sammer, H., One Thousand Transactions Per Second, IEEE COMPCON, 1985

[Gro85] Grossman, C.P., Cache-DASD Storage Design for Improving System Performance, IBM Systems Journal Vol.24 No.3, 1985

[GLV84] Garcia-Molina, H., Lipton, R.J., Valdes, J., A Massive Memory Machine, IEEE Transactions on Computers Vol.33 No.5, 1984

[GS84] Gifford, D., Spector, A., The TWA Reservation System, Communications of the ACM Vol.27 No.7, 1984

[GT89] Graefe, G., Thakkar, S.S., Tuning a Parallel Database System on a Shared-Memory Multiprocessor, Manuscript, submitted for publication

[Ha90] Haas, L.M., Chang, W., Lohman, G.M., McPherson, J., Wilms, P.F., Lapis, G., Pirahesh, H., Carey, M., Shekita, E., Starburst Mid-Flight: As the Dust Clears, IEEE Transactions on Knowledge and Data Engineering Vol.2 No.1, 1990

[Ham90] Hammer, M., Simulation of an Intelligent DBMS Scheduler (in German), M.Sc. Thesis, ETH Zurich, 1990

[He89] Herrmann, U., Dadam, P., Kuespert, K., Schlageter, G., Locking Disjoint, Non-recursive Complex Objects through Object-Specific and Query-Specific Lock Graphs (in German), 3rd GI Conf. on Database Systems in Office Automation, Engineering, and Scientific Applications, 1989, Springer, IFB 204

[HHM86] Haerder, T., Huebel, C., Mitschang, B., Use of Inherent Parallelism in Database Operations, CONPAR 86, Springer, LNCS 237

[HK88] Hudson, S., King, R., An Adaptive Derived Data Manager for Distributed Databases, 2nd Int. Workshop on Object-Oriented Database Systems, 1988, Springer, LNCS 334

[HK89] Hudson, S., King, R., Cactis: A Self-Adaptive, Concurrent Implementation of an Object-Oriented Database Management System, ACM TODS Vol.14 No.3, 1989

[HSS89] Haerder, T., Schoening, H., Sikeler, A., Parallel Query Evaluation: A New Approach to Complex Object Processing, IEEE Database Engineering Vol.12 No.1, 1989

[In88] Inmon, W.H., Optimizing Performance in DB2 Software, Prentice-Hall, 1988

[Jo88] Joseph, J., Thatte, S., Thompson, C., Wells, D., Special Report on the 1988 Object-Oriented Database Workshop, Section 3: Panel on Transactions for Cooperative Design Work, ACM SIGMOD Record Vol.18 No.3, 1989

[JTK89] Jenq, B.P., Twichell, B., Keller, T., Locking Performance in a Shared Nothing Parallel Database Machine, IEEE Data Engineering Conf., 1989

[Ka89] R.H. Katz, J.K. Ousterhout, D.A. Patterson, P. Chen, A. Chervenak, R. Drewes G. Gibson, E. Lee, K. Lutz, E. Miller, and M. Rosenblum, A Project on High Performance I/O Subsystems, ACM Computer Architecture News Vol.17 No.5, 1989

[Kim86] Kim, M.Y., Synchronized Disk Interleaving, IEEE Transactions on Computers Vol.35 No.11, 1986

[KWZ89] Kratzer, K., Wedekind, H., Zoerntlein, G., Prefetching - A Performance Analysis, Technical Report, University of Erlangen, 1989

[Lag] Future Directions in DBMS Research, The Laguna Beach Participants, ACM SIGMOD Record Vol.18 No.1, 1989

[Li85] Liskov, B., The Argus Language and System, in: Paul, M., Siegert, H.J. (eds.), Distributed Systems, Springer, LNCS 190, 1985

[Lo89] Lorie, R., Daudenarde, J., Hallmark, G., Stamos, J., Young, H., Adding Intra-Transaction Parallelism to an Existing DBMS: Early Experience, IEEE Data Engineering Vol.12 No.1, 1989

[LC89] Lehman, T.J., Carey, M.J., A Concurrency Control Algorithm for Memory-Resident Database Systems, Int. Conf. on Foundations of Data Organization, 1989, Springer, LNCS 367

[LKB87] Livny, M., Khoshafian, S., Boral, H., Multi-Disk Management Algorithms, ACM SIGMETRICS Conference, 1987

[LSW86] Lausen, G., Soisalon-Soininen, E., Widmayer, P., Towards Online Schedulers Based on Pre-Analysis Locking, 1st Int. Conf. on Database Theory, 1986, Springer, LNCS 243

[Mo85] Mohan, C., Fussell, D., Kedem, Z.M., Silberschatz, A., Lock Conversion in Non-Two-Phase Locking Protocols, IEEE Transactions on Software Engineering Vol.11 No.1, 1985

[Mo89] Mohan, C., Haderle, D., Lindsay, B., Pirahesh, H., Schwarz, P., ARIES: A Transaction Recovery Method Supporting Fine-Granularity Locking and Partial Rollbacks Using Write-Ahead Logging, IBM Research Report RJ6649, San Jose, 1989

[MGG86] Moss, J.E.B., Griffeth, N.D., Graham, M.H., Abstraction in Recovery Management, ACM SIGMOD Conf., 1986

[ML89] Mohan, C., Levine, F., ARIES/IM: An Efficient and High Concurrency Index Management Method Using Write-Ahead Logging, IBM Research Report RJ6846, San Jose, 1989

[MR89] Murphy, M.C., Rotem, D., Effective Resource Utilization for Multiprocessor Join Execution, VLDB Conf., 1989

[O'N86] O'Neil, P.E., The Escrow Transactional Method, ACM TODS Vol.11 No.4, 1986

[O'N87] O'Neil, P.E., Model 204 Architecture and Performance, 2nd Int. Workshop on High Performance Transaction Systems, 1987, Springer, LNCS 359

[Ora] Oracle RDBMS Database Administrator's Guide Version 6.0, Chapter 12: Consistency and Concurrency, Oracle Corp., 1988

[OD89] Ousterhout, J., Douglis, F., Beating the I/O Bottleneck: A Case for Log-Structured File Systems, ACM Operating Systems Review Vol.23 No.1, 1989

[PGK88] Patterson, D.A., Gibson, G., Katz, R.H., A Case for Redundant Arrays of Inexpensive Disks (RAID), ACM SIGMOD Conf., 1988

[PLZ89] Plattner, B., Lanz, C., Zogg, A., X.500: Standards for Directory Services in Communication Systems (in German), Technical Report #106, Computer Science Dept., ETH Zurich, 1989

[PRS88] Peinl, P., Reuter, A., Sammer, H., High Contention in a Stock Trading Database: A Case Study, ACM SIGMOD Conf., 1988

[Reu86] Reuter, A., Duppel, N., Peinl, P., Schiele, G., Zeller, H., An Outlook on PROSPECT, Technical Report, University of Stuttgart, 1986

[Reu88] Reuter, A., Distributed Database Systems: State-of-the-Art and Current Trends (in German), GI Annual Conference, 1988, Springer, IFB 187

[Ru89] Rubsam, K.G., MVS Data Services, IBM Systems Journal Vol.28 No.1, 1989

[RB89] Reddy, A.L., Banerjee, P., An Evaluation of Multiple-Disk I/O Systems, IEEE Transactions on Computers Vol.38 No.12, 1989

[RP81] Reiner, D., Pinkerton, T., A Method for Adaptive Performance Improvement of Operating Systems, ACM SIGMETRICS Conf., 1981

[RTIa] Relational Technology Inc., The INGRES Locking System, Ingres (Rel.5) Technical Note # 31

[RTIb] Relational Technology Inc., Collision Detection on Updates to a Single Record, Ingres (Rel.5) Technical Note #36

[RTIc] Relational Technology Inc., Improving Concurrent User Performance, Ingres (Rel.5) Technical Notes #38 and #49

[Sa89] Samadi, B., TUNEX: A Knowledge-Based System for Performance Tuning of the UNIX Operating System, IEEE Transactions on Software Engineering Vol.15 No.7, 1989

[Sch90a] Schek, H.-J., Paul, H.-B., Scholl, M.H., Weikum, G., The DASDBS Project: Objectives, Experiences, and Future Prospects, IEEE Transactions on Knowledge and Data Engineering Vol.2 No.1, 1990

[Sch90b] Schek, H.-J., Scholl, M.H., Weikum, G., From the KERNEL to the COSMOS: The Database Research Group at ETH Zurich, Technical Report, Computer Science Dept., ETH Zurich, 1990

[Schn83] Schnell, P., Implementation and Engineering of a Production-Oriented DBMS, IFIP Congress, 1983, North-Holland

[Seq] Symmetry Technical Summary, Sequent Computer Systems Inc., 1987

[Sm85] Smith, A.J., Disk Cache - Miss Ration Analysis and Design Considerations, ACM TOCS Vol.3 No.3, 1985

[Sp87] Spector, A., Eppinger, J., Daniels, D., Draves, R., Bloch, J., Duchamp, D., Pausch, R., Thompson, D., High Performance Distributed Transaction Processing in a General Purpose Computing Environment, 2nd Int. Workshop on High Performance Transaction Systems, 1987, Springer, LNCS 359

[St86] Stonebraker, M., The Case for Shared Nothing, IEEE Database Engineering Vol.9 No.1, 1986

[St88] Stonebraker, M., Katz, R., Patterson, D., Ousterhout, J., The Design of XPRS, VLDB Conf., 1988

[Sv84] Svobodova, L., File Servers for Network-Based Distributed Systems, ACM Computing Surveys Vol.16 No.4, 1984

[SD89] Schneider, D.A., DeWitt, D.J., A Performance Evaluation of Four Parallel Join Algorithms in a Shared-Nothing Multiprocessor Environment, ACM SIGMOD Conf., 1989

[SG86] Salem, K., Garcia-Molina, H., Disk Striping, IEEE Data Engineering Conf., 1986

[SGA87] Salem, K., Garcia-Molina, H., Alonso, R., Altruistic Locking: A Strategy for Coping with Long Lived Transactions, 2nd Int. Workshop on High Performance Transaction Systems, 1987, Springer, LNCS 359

[SPS87] Scholl, M.H., Paul, H.-B., Schek, H.-J., Supporting Flat Relations by a Nested Relational Kernel, VLDB Conf., 1987

[SS89] Scholl, M.H., Schek, H.-J., A Relational Object Model, Manuscript, ETH Zurich, 1989

[Th89] Thurber, K.J., Getting a Handle on FDDI, Data Communications, Special Issue on LAN Strategies, June 1989

[Tr83] Traiger, I.L., Trends in Systems Aspects of Database Management, 2nd Int. Conf. on Databases, 1983, Wiley Heyden

[TF82] Teorey, T.J., Fry, J.P., Design of Database Structures, Prentice-Hall, 1982

[TG84] Teng, J.Z., Gumaer, R.A., Managing IBM Database 2 Buffers to maximize performance, IBM Systems Journal Vol.23 No.2, 1984

[TGS85] Tay, Y., Goodman, N., Suri, R., Locking Performance in Centralized Databases, ACM TODS Vol.10 No.4, 1985

[VV88] Vianu, V., Vossen, G., Conceptual Level Concurrency Control of Relational Update Transactions, 2nd Int. Conf. on Database Theory, 1988, Springer, LNCS 326

[We88] Weihl, W.E., Commutativity-based Concurrency Control for Abstract Data Types, IEEE Transactions on Computers Vol.37 No.12, 1988

[Wei87a] Weikum, G., Principles and Realization Strategies of Multi-Level Transaction Management, Technical Report, TH Darmstadt, 1987, accepted for ACM TODS

[Wei87b] Weikum, G., Enhancing Concurrency in Layered Systems, 2nd Int. Workshop on High Performance Transaction Systems, 1987, Springer, LNCS 359

[Wei89a] Weikum, G., Set-Oriented Disk Access to Large Complex Objects, IEEE Data Engineering Conf., 1989

[Wei89b] Weikum, G., Clustering vs. Concurrency: A Framework and a Case Study, MCC Technical Report ACA-ST-096-89, Austin, 1989

[Wei90] Weikum, G., Hasse, C., Broessler, P., Muth, P., Multi-Level Recovery, ACM PODS Conf., 1990

[Wi89] Wilschut, A.N., Grefen, P.W.P.J., Apers, P.M.G., Kersten, M.L., Implementing PRISMA/DB in an OOPL, Int. Workshop on Database Machines, 1989, Springer, LNCS 368

[Wo86] Woodside, C.M., Controllability of Computer Performance Tradeoffs Obtained Using Controlled-Share Queue Schedulers, IEEE Transactions on Software Engineering Vol.12 No.10, 1986

[WZ86] Wedekind, H., Zoerntlein, G., Prefetching in Realtime Database Applications, ACM SIGMOD Conf., 1986

[Ze89] Zeller, H., Parallelizing Complex-Object Queries by Hash Joins (in German),3rd GI Conf. on Database Systems in Office Automation, Engineering, and Scientific Applications, 1989

[ZPD90] Zabback, P., Paul, H.-B., Deppisch, U., Office Documents on a Database Kernel - Filing, Retrieval, and Archiving, Int. Conf. on Office Information Systems, 1990

Gelbe Berichte des Departements Informatik

124	C. Pfister	The Graphics Editor Condor The Layout System Pedro
125	R. Crelier	OP2: A Portable Oberon Compiler
126	C. Szyperski	Network Communication in the Oberon Environment
127	H. Mössenböck	Coco/R: A Generator for Fast Compiler Front-Ends (vergriffen)
128	B. Sanders	Eliminating the Substitution Axiom from UNITY Logic
129	L. Pfau	GFT: A Tool for Data Management in the UNIX Environment
130	S. Bondeli	Divide and Conquer: A Parallel Algorithm for the Solution of a Tridiagonal Linear System
131	H. Goorhuis N. Gürman M. Montigel L. Thalmann	Neuronale Netze und regelbasierte Systeme: Ein hybrider Ansatz
132	B. Jiang	Design, Analysis, and Evaluation of Algorithms for Computing Partial Transitive Closures in Databases
133	J. Templ	SPARC-Oberon User´s Guide
134	G. Wong	An Appr5oach on How to Combine Object Recognition Methods
135	C. Wieland	Two Explanation Facilities for the Deductive Database Management System DeDEx
136	H.-J. Schek M.H Scholl G. Weikum	From the KERNEL to the COSMOS: The Database Research Group at ETH Zürich