in: Proc. PARLE-92 (Parallel Architectures and Languages Europe), Paris, 1992, Lecture Notes in Computer Science 605, Springer-Verlag, pp. 295-310

Performance Evaluation of Parallel Transaction Processing in Shared Nothing Database Systems

Robert Marek
Erhard Rahm

University of Kaiserslautern
Dept. of Computer Science
6750 Kaiserslautern, GERMANY

Abstract. Complex and data-intensive database queries mandate parallel processing strategies to achieve sufficiently short response times. In praxis, parallel database processing is mostly based on so-called "shared nothing" architectures entailing a physical partitioning and allocation of the database among multiple processing nodes. We examine the performance of such architectures by using a detailed simulation system. We analyse response time performance of transactions and individual database queries in single-user as well as in multi-user mode. Furthermore, we study the throughput behavior for on-line transactions. Three workload types covering a wide range of commercial applications are used for performance evaluation: the debit-credit benchmark load, synthetically generated relational queries as well as real-life workloads represented by database traces.

1 Introduction

Increasing requirements on performance, availability and modular system growth demand distributed architectures for transaction and database processing. With respect to on-line transaction processing (OLTP), high transaction rates (subject to a response time constraint) for short transactions as debit-credit [An85, TPC89] is the primary performance objective. For complex database queries, providing short response times acceptable for on-line execution is the main performance challenge, in particular if such queries are executed on the same database than concurrent OLTP transactions. Since complex queries typically access large amounts of data or/and perform extensive computations, in general the response time goal can only be achieved by employing parallel query processing [Pi90]. Performance should scale with the number of nodes: ideally adding processing nodes linearly improves throughput for OLTP or response times for complex queries. Furthermore, viable solutions have to provide good cost-effectiveness which can be achieved by using a larger number of powerful microprocessors (instead of mainframes) for database processing. Typically, the price per MIPS (Million Instructions Per Second) is substantially lower for microprocessors than for mainframes.

To meet these requirements, research and system developments have concentrated on so-called *shared nothing* architectures [St86] for distributed and parallel database processing. Shared nothing systems consist of multiple functionally homogenous processing nodes or processing elements (PE). Each PE comprises one or more CPUs and a local main memory, and runs local copies of application and system software like operating system and database management system (DBMS). Cooperation between PE takes place by means of message passing. Typically, the PE are locally distributed so that a high-speed network can be used for communication. The characteristic feature of shared nothing systems is that the database is partitioned and distributed among all nodes so that every PE "owns" one partition. If a transaction needs to access data owned by another node, a sub-transaction is started at the respective owner PE to access the remote data. In this case, a distributed two-phase commit protocol [MLO86, ÖV91] is also to be executed to guarantee the all-or-nothing property [HR83] of the transaction. Existing shared nothing systems supporting parallel transaction processing include the products Tandem NonStop SQL [Ta89, EGKS90] and Teradata's DBC/1012 [Ne86] as well as several prototypes like Bubba [Bo90], Gamma [De90], EDS [WT91] and Prisma [Ke88]. With the exception of Tandem, these systems represent database machines (back-end systems) dedicated to database processing. The database operations or DML (Data Manipulation Language) statements submitted to the back-end system may originate directly from the end-user (ad-hoc queries) or from application programs running on workstations or mainframes. Some database machines (e.g. EDS) support the management of application programs (consisting of multiple DML statements) in the back-end system in the form of "stored procedures" that may be started by a single request.

With respect to parallel transaction processing, we can roughly distinguish between inter- and intra-transaction parallelism. *Inter-transaction parallelism* refers to the concurrent execution of multiple independent transactions on the same database. This kind of parallelism is already supported in centralized DBMS (multi-user mode), e.g. in order to overlap I/O delays to achieve acceptable system throughput. To improve response time, intra-transaction parallelism is needed either in the form of inter-DML or intra-DML parallelism. *Inter-DML parallelism* refers to the concurrent execution of different DML statements of the same transaction. The degree of parallelism obtainable by inter-DML parallelism, however, is limited by the number of database operations of the transaction as well as by precedence constraints between these operations. Currently, commercial DBMS do not support this kind of parallelism because the programmer would have to specify the DML dependencies using adequate language features. *Intra-DML parallelism* aims at parallel processing of a single DML statement based on a parallel execution plan generated by the DBMS query optimizer.

In shared nothing systems, parallel query processing is largely influenced by the chosen database allocation. To support response time improvements, the database should be allocated such that sub-queries on disjoint database portions can be processed in parallel on different PE (e.g. using a horizontal partitioning of relations). On the other hand, parallel query processing entails cooperation and communication overhead for initialization of sub-queries (or sub-transactions), for exchanging intermediate results and for two-phase-commit. While this overhead increases with the number of nodes, the response time improvements obtainable by increasing the degree of intra-transaction parallelism generally decrease. As a result, parallel query processing is useful only for a limited number of PE in general [Bo90]. Another problem is to find a database and workload allocation supporting both high transaction rates for OLTP transactions and a high potential for parallelizing complex queries. OLTP transactions should be processed locally as far as possible to limit the communication overhead reducing the attainable transaction rates. On the other hand, parallelizing complex queries introduces communication overhead in order to support short response times. Frequently, supporting both locality of reference (by clustering related data and transactions) and intra-transaction parallelism (by de-clustering data) are contradicting subgoals that may not be achievable at the same time.

We have developed a comprehensive simulation system to study basic performance trade-offs of shared nothing architectures. A distinctive feature of our approach is that we support three different workload types for performance evaluation: the debit-credit workload constituting the standard load in OLTP benchmarks [An85, TPC89], synthetically generated relational queries as well as real-life workloads represented by database traces. These workloads are used to investigate response time and throughput performance for inter- and intra-transaction parallelism. Former performance studies [DGS88, SD89] were restricted to a single workload type (e.g. relational queries) or concentrated on comparing the performance of alternative algorithms for parallel query processing (e.g. join strategies). Furthermore, most studies only considered intra-DML parallelism in single-user mode thereby ignoring inter-transaction parallelism. Since in reality multi-user mode is inevitable to support acceptable throughput and cost-effectiveness, inter-transaction parallelism should also be taken into account in performance evaluations. Other performance evaluations investigated concrete systems [EGKS90] or concentrated on specific aspects like the impact of concurrency control [CL89, JTK89]. We are not aware of any other study using real-life traces in the evaluation of parallel query processing strategies.

The next section provides a survey of our simulation system. In section 3 we describe the workloads as well as the results of the conducted simulation experiments. Finally, we summarize the major findings of our investigation.

2 Simulation model

Our simulation system models the hardware and transaction processing logic of a generic shared nothing DBMS architecture. The system has been implemented using the discrete event simulation language DeNet [Li87, Li89] and encompasses more than 20.000 lines of source code. Our system consists of two main components: *workload generation* and *processing subsystem*. The first component generates the workload and assigns it to the PE of the processing subsystem where the actual transaction processing takes place. In this section, we summarize the implementation of both components; a more detailed description can be found in [Ma91]. Several architectural features and processing schemes have been chosen according to the ESPRIT effort EDS [WT91].

2.1 Workload generation and allocation

The database and workload model is of great importance to any database performance evaluation. In order to cover a wide range of applications we support three different workload types: debit-credit transactions, relational queries and real workloads derived from database traces. In all cases, we use the same database model which is based on four object granularities: database, partitions, pages and objects (e.g. records). The database is a collection of partitions that may be used to represent a file, a record type (relation) or an index structure. A partition consists of a number of database pages which in turn consist of a specific number of objects (records). The number of objects per page is determined by a blocking factor which can be specified on a per-partition basis. Differentiating between objects and pages is important in order to study the effect of clustering which aims at reducing the number of page accesses (disk I/Os) by storing related objects into the same page. Furthermore, concurrency control may now be performed on the page or object level. For relational queries, we additionally support specific index structures (clustered and non-clustered B^*-trees).
We employ a horizontal data distribution of partitions (relations) at the object level controlled by a relative distribution table. This table defines for every partition P_j and processing element PE_i which fragment of P_j is allocated to PE_i.

To keep the processing subsystem independent from the various workload generators we defined a uniform interface for representing transactions and queries that supports all workload types and parallelization forms. In this model, a transaction consists of a BOT step (begin of transaction), several DML statements and an EOT step (end of transaction). Each DML statement in turn consists of multiple object references, indicating the object and page identifications and the access mode (read or write).To achieve the mentioned independency between workload generation and processing subsystem, we determined the distribution and parallelization of DML statements already at load generation time, supporting both inter- and intra-DML parallelism. Special FORK-WAIT operations are used to specify distributed and parallel execution sequences within a transaction or a DML statement. The transaction and query representations at the interface to the processing subsystem roughly correspond to the execu-

tion plans in real systems that are generated during the compilation (optimization) of transaction programs and queries.
The parallelization forms and parameter settings used for the different workloads will be described in section 3.

The simulation system is an open queuing model and supports the definition of an individual arrival rate for each transaction type. Several strategies can be chosen for workload allocation, e.g. random routing or the use of a routing table. In the latter case, for every transaction type T_j and processing element PE_i it can be specified which percentage of type T_j will be assigned to PE_i.

2.2 Workload processing

The processing component models the execution of a workload on a shared nothing system with an arbitrary number of PE connected by a communication network. Each PE has access to private database and log files allocated on external storage devices (disks). Internally, each PE is represented by a transaction manager, a buffer manager, a concurrency control component, a communication manager and a CPU server. Figure 1 shows the main components of the processing subsystem.

The transaction manager controls the (distributed) execution of transactions. The maximal number of concurrent transactions per PE is controlled by a multiprogramming level. Newly arriving transactions must wait in an input queue until they can be served when this maximal degree of inter-transaction parallelism is already reached. The transaction processing starts

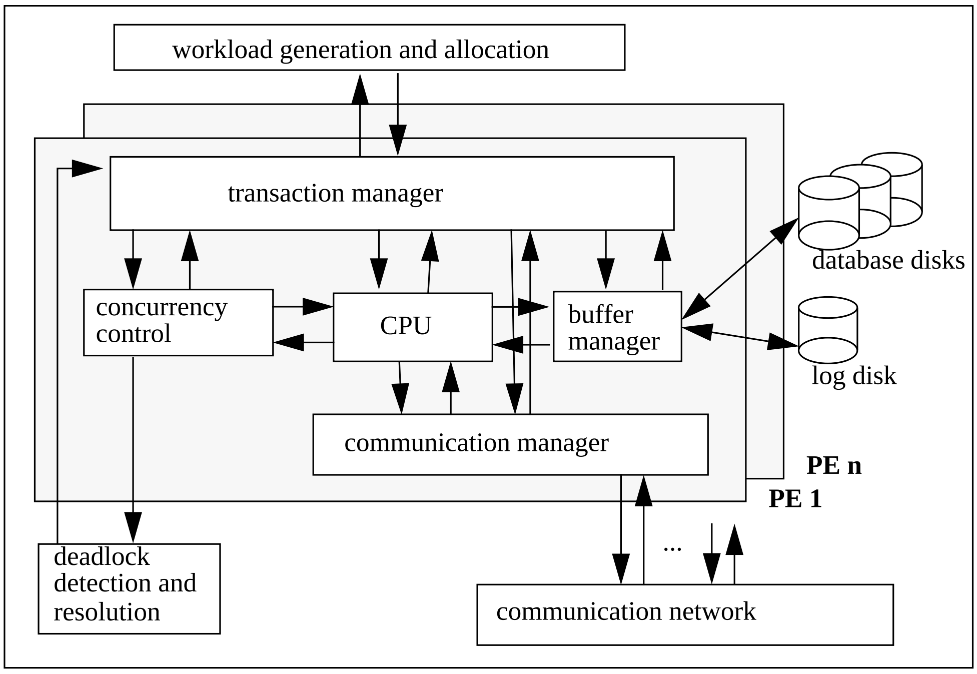

Figure 1: Gross structure of the simulation system.

with the BOT processing entailing the transaction initialization overhead. The processing of DML operations and object references is performed according to the execution structure determined by the workload generator indicating when remote requests and parallel sub-transactions have to be started. The EOT step triggers two-phase commit processing involving all

PE that have participated during execution of the respective transaction. We support the optimization proposed in [MLO86] where read-only sub-transactions only participate in the first commit phase.

CPU requests are served by a single CPU per PE. The average number of instructions per request can be defined separately for every request type (e.g. transaction initialization, DML and object reference processing, communication overhead, I/O overhead etc.).

For concurrency control, we employ distributed strict two-phase locking (long read and write locks). The local concurrency control manager in each PE controls all locks on the local partition. Locks may be requested either at the page or object level. A central deadlock detection scheme is used to detect global deadlocks and initiate transaction aborts to break cycles.

Database partitions can be kept memory-resident (to simulate main memory databases) or they can be allocated to a number of disks. Disks and disk controllers have explicitly been modelled as servers to capture I/O bottlenecks. Disks are accessed by the buffer manager component of the associated PE. The database buffer in main memory is managed according to a global LRU (Least Recently Used) replacement strategy. For update propagation to disk, either a FORCE or NOFORCE strategy [HR83] can be selected. FORCE requires to write out all pages modified by a transaction at EOT, while NOFORCE only incurs logging I/O. Logging is modelled by writing a single page per update transaction or sub-transaction to the local log file of the respective PE.

The communication network provides transmission of message packets of fixed size. Messages exceeding the packet size (e.g. large sets of result tuples) are disassembled into the required number of packets.

3 Simulation results

Using the described simulation system, we conducted a large number of simulation experiments with different workloads. Our performance evaluation concentrates on the influence of parallelism and the number of PE (scalability) on throughput and response time. With respect to scalability, the following performance metrics are essential for parallel systems [EGKS90, Bo90]:

- *Throughput scaleup* measures the throughput improvement as the number of PE and the database size are increased. For N PE, scaleup is defined as the quotient of the throughput for N PE and the throughput for 1 PE. This metric is especially important for OLTP workloads (e.g. debit-credit).
- *Response time speedup*, on the other hand, measures the improvement of complex query response times as more PE are added to execute the query. For N PE, the speedup is obtained by dividing the response time for a single PE by the response time result for parallel execution on N PE with the same database size.

Table 1 summarizes the major parameter settings used for all workloads. In general, we varied the number of nodes between 1 and 64; for the trace-driven experiments only up to 8 PE proved useful. The parameters for the I/O (disk) subsystem were chosen so that no bottlenecks occurred (sufficiently high number of disks and controllers). The duration of an I/O operation is composed of the controller service time, disk access time and transmission time. The parameter settings for the communication network have been chosen according to the EDS prototype [WT91].

In the following three subsections 3.1 to 3.3, we discuss the simulation results for our three workload types. Workload-specific parameter settings will also be described there.

parameters		settings
number of PE		1, 2, 4, 8, 16, 32, 64
CPU:	number of processors per PE	1
	processor capacity	30 MIPS
avg. #instructions	for BOT	25000
	for EOT	25000
	per object reference	5000 (Debit Credit: 25000)
	for message send/receive	5000
	per I/O	3000
buffer manager:	update strategy	NOFORCE
disk devices:	avg. controller service time	1 ms (per page)
	transmission time per page	0.4 ms
	avg. disk access time	15 ms (5 ms for log disks)
comm. network:	packet size	128 bytes data
	avg. packet transmission time	8 microsec

Table 1: General parameter settings.

3.1 Results for Debit-Credit workload

The debit-credit workload is completely homogeneous and consists of a single transaction type from a banking application [An85, TPC89]. Every transaction updates one ACCOUNT, BRANCH and TELLER record. Additionally, a record is inserted into a HISTORY file. There is a one-to-many relationship between BRANCH and TELLER and BRANCH and ACCOUNT specifying all accounts and tellers associated with a given branch. According to [An85, TPC89], 15% of the transactions access an ACCOUNT record of a different branch than the one where the transaction is processed.

Table 2 shows the major parameter settings for the debit-credit workload. The database size is chosen according to the throughput goal (number of PE) as required by the benchmark definition. The size of the HISTORY partition is immaterial for our purposes as every transaction adds a new record at the end of this sequential file. Each relation is equally (horizontally) distributed among all PE. In particular, each PE holds all TELLER, ACCOUNT and HISTORY records belonging to the PE's BRANCH records. Hence, when a transaction is assigned to the PE owning the corresponding BRANCH record, accesses to BRANCH, TELLER and HISTORY are local and inter-PE communication may only be required for up to 15% of the ACCOUNT accesses. Besides such an "intelligent" (BRANCH-based) transaction routing, we also investigated a random routing of transactions in order to study the influence of sub-optimal workload allocation strategies. Random routing is expected to cause much more inter-PE communication since both the BRANCH and the ACCOUNT record may be allocated on a remote PE. Additionally, more transactions are subject to a distributed two-phase commit. In our experiments the smaller relations (BRANCH, TELLER and HISTORY) were kept resident in main memory whereas ACCOUNT was allocated either on disks or in main memory. With the parameter settings from Table 1, the average path-length per transaction is 150.000 instructions (BOT, four object references, EOT) not including the overhead for communication and I/O.

parameters	settings
number of objects per partition	per PE: 100 BRANCH, 1000 TELLER and 10.000.000 ACCOUNT objects
blocking factor (number of objects per page)	1 (BRANCH), 10 (TELLER), 20 (ACCOUNT and HISTORY)
concurrency control	object level
storage allocation	BRANCH, TELLER, HISTORY: main memory resident ACCOUNT: disk or main memory resident
main memory buffer size	10 page frames (for ACCOUNT)
transaction routing	via BRANCH, RANDOM

Table 2: Parameter settings for debit-credit workload.

For debit-credit, the obtainable throughput (using inter-transaction parallelism) is of primary interest. Fig. 2 shows the achieved transaction rates (in transactions per second, TPS) as well as the corresponding throughput scaleup for system sizes varying between 1 and 64 PE. These results refer to a CPU utilization of 90% (based on simulation results). The experiments are designated as follows:

-	experiment	b*:	BRANCH-based transaction routing
-		r*:	random transaction routing
-		*d:	ACCOUNT allocated on disk
-		*m:	all relations main memory resident.

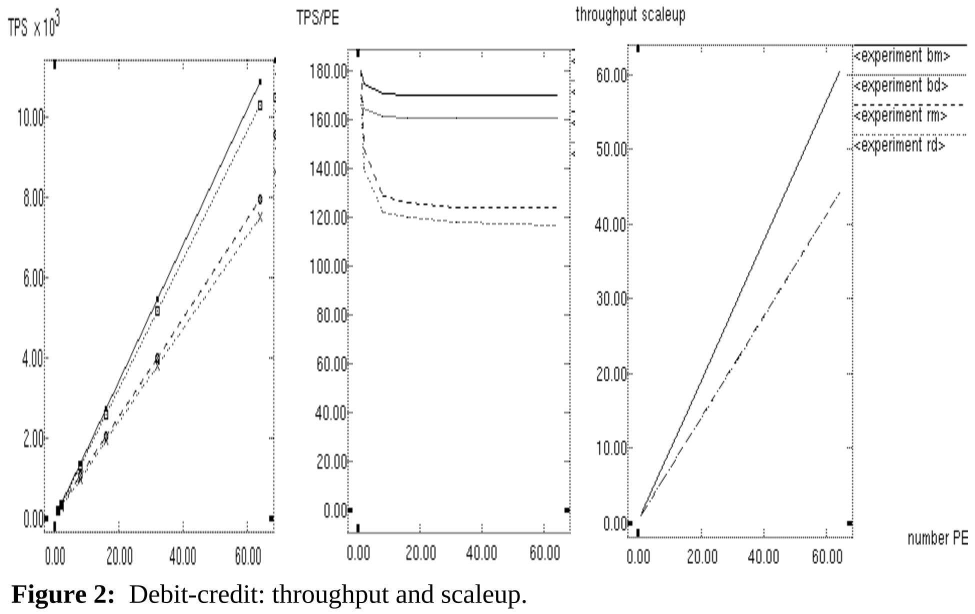

Figure 2: Debit-credit: throughput and scaleup.

The curves show that in all cases throughput could almost linearly be increased with the number of PE. However, with random routing the throughput increase takes place at a lower level due to the considerably higher communication overhead than for the BRANCH-based workload allocation. So only a throughput scaleup of 44 was reached for random routing compared to 60 for a BRANCH-based routing in the case of 64 PE. The almost linear throughput scaleup even for random routing was favored by the short size of this transaction. So the worst case in terms of messages per transactions was almost reached for 8 PE already so that additional PE did not cause any further deterioration. Random routing causes a maximum of 6.9 messages per transaction vs. 0.9 messages using BRANCH-based routing. However, regarding throughput per PE, the impact of communication on performance can be perceived more clearly. In any case throughput per PE actually decreases with growing system size, showing best performance using BRANCH-based routing. The storage allocation for ACCOUNT was less significant for throughput since the I/O overhead was small compared to the total pathlength. In fact, throughput scaleup is virtually the same for the disk- and the memory-based allocation.
A similarly ideal throughput behavior for debit-credit transactions has already been achieved in real shared nothing systems. For instance, Tandem demonstrated such performance characteristics for NonStop SQL on 2 to 32 processors (using a disk-based storage allocation and a BRANCH-based workload allocation) [Ta88]. Since performance predictions of the 256 PE EDS prototype state 12000 TPS running at 30% utilization [WT91], our simulation results (approx. 10000 TPS on a 64 PE machine running at 90% utilization) proved to be realistic.

Due to the increased number of remote requests, transaction response time for random routing was higher than for the BRANCH-based workload allocation. For debit-credit, the use of intra-transaction parallelism permitted only modest response time improvements. Intra-DML parallelism cannot be utilized since each DML statement accesses only a single record. Inter-DML parallelism, however, is applicable as all four DML statements can be simultaneously executed in principle. However, since we assumed a single CPU per PE and most DML statements of a transaction are executed on the same PE, this kind of parallelism did not yield large response time improvements, too. On the positive side, the use of inter-DML parallelism does not cause any extra messages for debit-credit so that the attainable transaction rates were the same as without employing intra-transaction parallelism.

3.2 Experiments using relational queries

With respect to the synthetically generated relational workloads, we restrict our considerations to the simple case of read operations on a single relation (no joins). Every transaction corresponds to a single SQL SELECT operation and is executed in parallel on all PE holding tuples of the corresponding relation (intra-DML parallelism). The transaction's local object references are basically determined by the relation's cardinality, the use of index structures, the selectivity of the (imaginary) selection predicate as well as the data allocation to PE. In our model, every distributed SELECT is followed by a merge statement covering the overhead for merging the qualifying tuples.

Table 3 shows the main parameter settings for this workload. We examine the response time impact of different access methods (relation scan vs. index scan) and predicate selectivities. In addition, multi-user experiments with different arrival rates were conducted. Like many existing DBS, our simulation system supports clustered and non-clustered B*-trees for indexing. In the case of a clustered index, the tuples of the relations themselves are stored in sort order permitting a substantial reduction of disk I/Os during query evaluation (in general, the amount of I/O savings corresponds to the blocking factor). The height of the B*-trees has been determined according to the relation size and the fan-out of index pages. In all experiments, the relation is uniformly distributed among all PE maximizing the potential for parallel

parameters		settings
relation	size	1.000.000 objects (tuples)
	blocking factor	10
	index type	no index, clustered index, non-clustered index
	storage allocation	disk
	allocation to PE	uniform
SELECT	selectivity	1 %, 10 %
	size of result tuples	100 bytes
	access method	sequential (relation scan) or index scan
	workload allocation	random (uniformly over all PE)
buffer size per PE		1000 pages

Table 3: Parameter settings for relational queries (SELECT on single relation)

query processing. To determine the communication overhead for result exchange, we specify the average size of qualifying tuples.

Fig. 3 plots response time and speedup results against the system size for different access methods in single-user mode and a selectivity of 1%. As expected, the use of a clustered index supports the best absolute response times while without index (relation scan) query execution takes the longest time. On the other hand, without index intra-DML parallelism permitted the highest absolute response time improvements by shortening the average execution time from about 30 minutes (1 PE) to 30 seconds for 64 PE. Speedup was linear over the entire range of number of nodes and almost optimal (factor 60 for 64 PE). Still, response time for 64 PE was higher than in the case with a clustered index on 1 PE. This illustrates, that in order to reduce query response time the use of an index may be more effective than employing parallel processing (Of course, not all queries can be supported by an adequate index. In particular, only one clustered index is possible per relation).

The ideal speedup behavior for relation scans was favored by the comparatively large size of the relation. In experiments with a relation size of 10.000 tuples, even for relation scans linear speedup was obtained only for up to 8 PE, while parallel processing on 64 PE merely resulted in a 35-fold response time improvement. This is because the communication overhead and delays per query increase proportionally with the number of PE, while the amount of useful work decreases (for 64 PE, each PE performs only 1/64 of all object references since the relation size remains constant). Even with a linear speedup, the absolute response time improvements decrease when adding more PE. Hence, the useful number of PE is also constrained by cost-effectiveness considerations (in Fig. 3, more than 16 or 32 PE only achieved comparatively small improvements).

In the case of index scans, the number of object references per PE is considerably lower than for a relation scan resulting in a much lower potential for parallel query processing. This was particularly the case for clustered indexes causing an unfavorable ratio between communication overhead and useful work and modest speedup values. For both index types, no substantial improvements were obtained any more when increasing the number of nodes from 32 to 64. However, in the case of a non-clustered index even a super-linear throughput could be obtained for 2-32 PE ! This interesting behavior was due to the fact that the aggregate buffer size grows with the number of PE while the database size remains constant. As a result, for a non-clustered index hit ratios increased with the number of nodes resulting in a response time improvement that was more significant than the communication delays for up to 32 nodes. In the case of a clustered index or without an index, the incresed buffer size could not be utilized

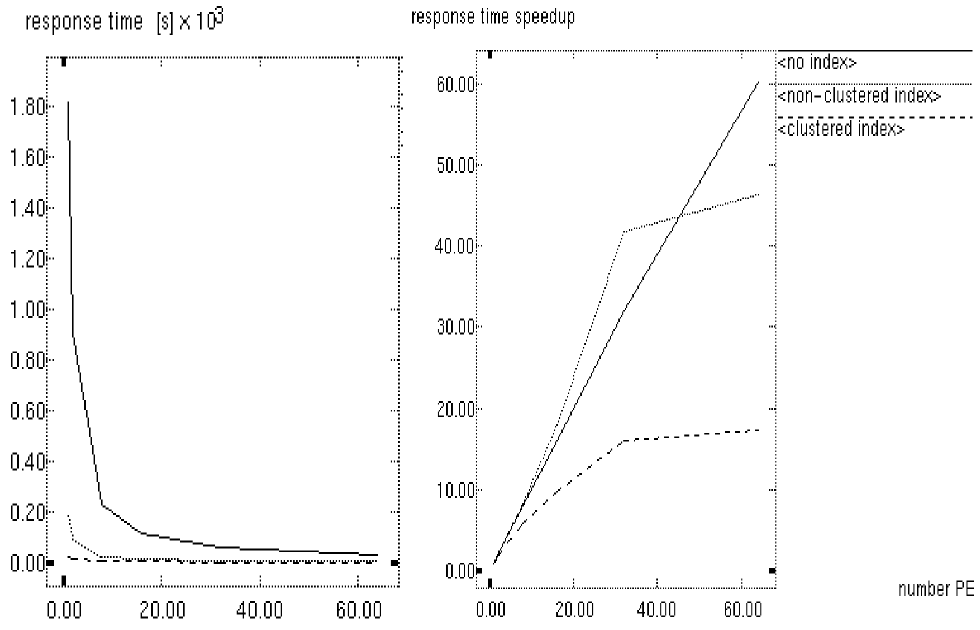

Figure 3: Influence of access method on response time and speedup (selectivity 1%).

since in these cases pages are sequentially processed. Due to the blocking factor 10, hit ratios

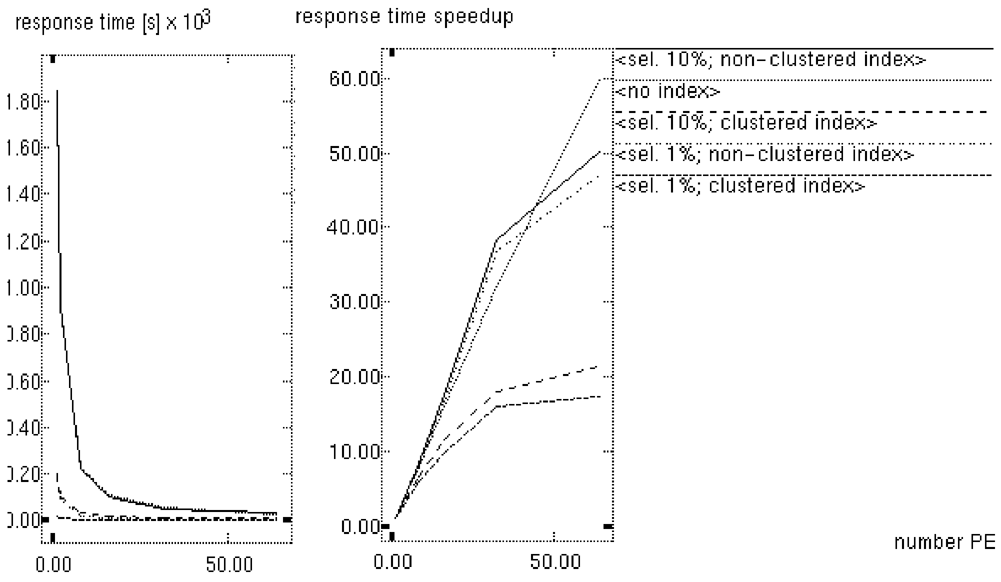

Fig. 4: Influence of selectivity and access method on response time and speedup (1 million tuples).

were always about 90% irrespective of the buffer size.

Further experiments were conducted to evaluate the influence of different query selectivities. In Fig. 4 we have additionally shown two curves for index scans and 10% selectivity. For such a high selectivity, the use of a non-clustered index did not prove useful and resulted in virtually the same response times than without index. Response times with a clustered index closely match the results for a non-clustered index and 1% selectivity. The speedup values for 10% selectivity show the same shape than for 1% but are slightly higher due to the increased number of object references per query improving the potential for intra-DML parallelism.

Finally, we examine the influence of multi-user mode (inter-transaction parallelism) on the effectiveness of intra-DML parallelism. For this purpose, we increased query arrival rates linearly with the number of nodes. Fig. 5 shows the resulting response time results for different arrival rates per PE and contrasts them with the single-user values (use of a clustered index and selectivity of 1% are assumed). We observe that response times increasingly deteriorate with growing arrival rates thereby considerably reducing the improvements obtained by intra-DML parallelism. For 0.2 TPS per PE, response times could be improved by intra-DML parallelism only for up to 16 PE, while increasing the number of PE further caused a steep response time increase. This behavior is due to the fact that communication overhead grows quadratically with the number of PE thereby increasingly causing CPU waits (higher CPU contention). The quadratic effect comes from the fact that the number of concurrent queries increases with the arrival rates and thus with the number of PE and because the communication overhead per query grows with the number of PE (intra-DML parallelism). As a result, in multi-user mode only for a restricted number of nodes could response times be improved

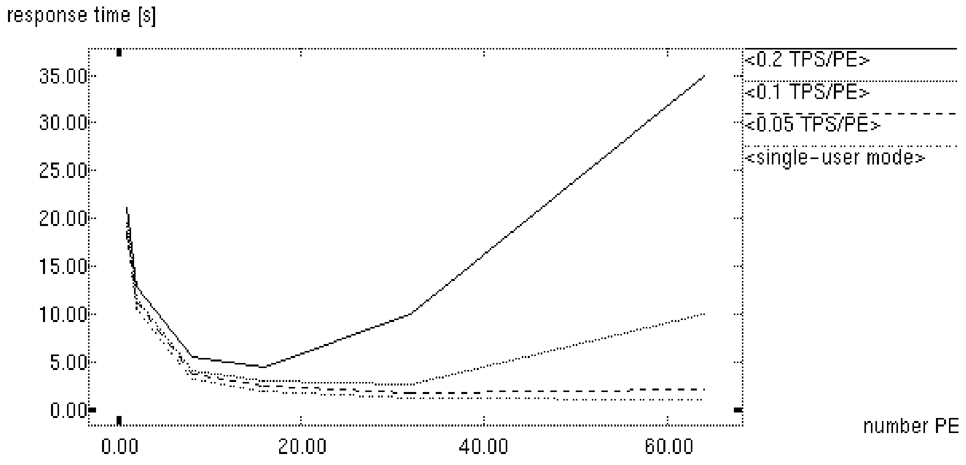

Figure 5: Influence of arrival rate on response time (use of intra-DML parallelism).

by intra-DML parallelism. Even in this range, speedup was considerably smaller compared to single-user mode. Furthermore, the communication overhead introduced by inter-DML parallelism significantly limits the attainable throughput and prevents a linear throughput increase with the number of PE.

Multiple CPUs per PE can help to reduce the average CPU waiting times so that the negative effect of multi-user mode may be less pronounced in this case. However, in the presence of update transactions lock contention can further limit the effectiveness of intra-DML parallelism.

3.3 Experiments using real-life workloads

The available traces were obtained from real-life database applications using a non-relational DBMS. Here we only consider results for the largest of these traces called DOA consisting of about 17.500 transactions and more than one million database references. Since the workload is dominated by read accesses (merely 1.6% of all accesses are writes), lock contention is expected to be low in the case of inter-transaction parallelism. The trace represents a "mixed" workload with a majority of shorter OLTP transactions and a few ad-hoc queries. While the largest query performs more than 11.000 database accesses, a transaction references 58 pages on average. Database accesses are spread over 13 database files and a total of about 66,000 different pages; the total database consists of about one million pages. The reference matrix in Table 4 depicts the relative access distribution of the 12 transaction types in the workload against the 13 database partitions. Transaction types and database partitions are ordered according to their number of references. The matrix value for transaction type x and partition y indicates which percentage of the total number of page references are caused by transactions of type x on pages of partition y (e.g. 9.1% of all references are issued by transaction type TT1 against partition P1). Below the reference matrix, for every partition the relative size is specified (in % of the total database size). Furthermore, it is indicated which fraction of a partition has been referenced during the trace period.

	P1	P2	P3	P4	P5	P6	P7	P8	P9	P10	P11	P12	P13	Total
TT1	9.1	3.5	3.3		5.0	0.9	0.4	0.1				0.0		22.3
TT2	7.5	6.9	0.4	2.6	0.0	0.5	0.8	1.0	0.3	0.2	0.0			20.3
TT3	6.4	1.3	2.8	0.0	2.6	0.2	0.7	0.1	1.1	0.4		0.0	0.0	15.6
TT4	0.0	3.4	0.3	6.8			0.6	0.4			0.0			11.6
TT5	3.1	4.1	0.4		0.0		0.5	0.0						8.2
TT6	2.4	2.5	0.6		0.7		0.9	0.3						7.4
TT7	1.3		2.6			2.3	0.1							6.2
TT8	0.3	2.3	0.2		0.0		0.1							2.9
TT9	0.0	1.4	0.0					1.1						2.6
TT10	0.3	0.1	0.3			1.0	0.1					0.0		1.8
TT11		0.9						0.2						1.1
TT12		0.1												0.1
Total	30.3	26.6	11.0	9.4	8.3	4.9	4.1	3.3	1.4	0.6	0.0	0.0	0.0	100.0
partition size (%)	31.3	6.3	8.3	17.8	1.0	20.8	2.6	7.3	2.6	1.3	0.8	0.0	0.0	100.0
% referenced	11.1	16.6	8.0	2.5	18.1	1.5	9.5	4.4	5.2	2.7	0.2	13.5	5.0	6.9

Table 4: Relative reference matrix of DOA transaction load.

The table shows that the major areas are accessed by almost every transaction type and that the important transaction types access all major areas. This means that in the distributed case the database and workload cannot generally be assigned such that transactions of different nodes operate on disjoint database partitions. Furthermore, access distribution between different partitions and within the partitions is clearly non-uniform. For instance, almost 27% of all references are directed against partition P2 which accounts only for about 6% of all pages. Furthermore almost 17% of the pages in P2 have been referenced during the trace period, while the corresponding share is merely about 7% for the entire database. Access frequencies to individual pages also differ largely within a partition (more than 4500 references are directed to the most frequently accessed page).

In our experiments, we used a data allocation such that each PE has to process about the same number of database accesses. Furthermore, a table-driven workload allocation was employed aiming at an approximately uniform PE utilization. Additionally, the algorithm to determine the database and workload allocations tried to find assignments minimizing the number of remote database accesses [Ra86].

Inter-DML parallelism could not be used for the trace-driven simulations since we have no information about precedence constraints between DML statements. To support intra-transaction parallelism we therefore use intra-DML parallelism by processing the object references of a DML statement in parallel if the respective objects are allocated to different nodes. On average, 6.5 objects are referenced per DML statement indicating a comparatively small potential for intra-DML parallelism partially influenced by the use of a non-relational DBMS. On the other hand, even with a relational DBMS the average number of accesses per database operation may be similarly low for applications with a high share of OLTP transactions that are typically supported by appropriate index structures.

Since DOA contains several transaction types of different size, transaction response time is no adequate performance metric any more.Therefore, we determined the execution time for so-called "units of processing" (U) rather than for transactions. Every object reference constitutes such a unit of processing as well as the BOT and EOT steps.

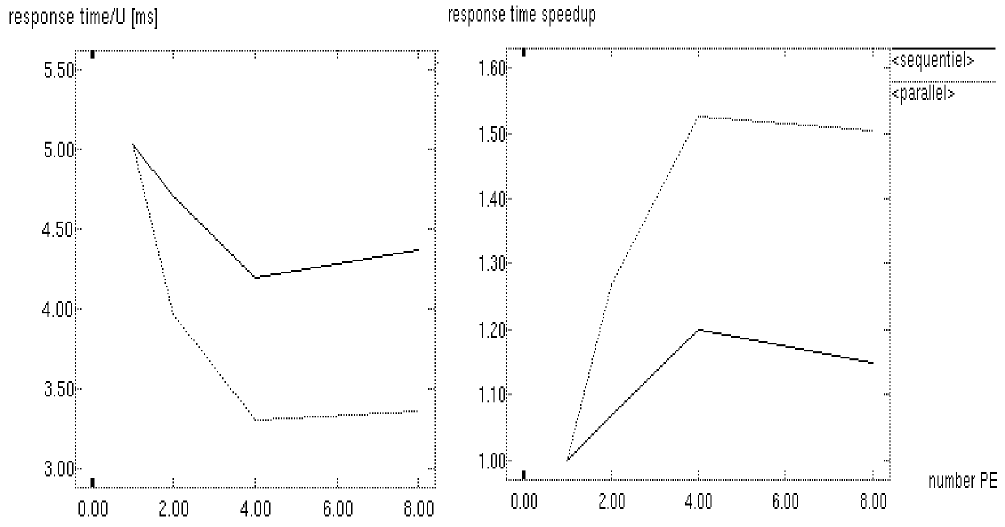

Figure 6: Response time and speedup in single-user mode (DOA).

Fig. 6 shows the response time and speedup results for DOA in single-user mode for up to 8 PE. In order to estimate the impact of intra-DML parallelism on response time, we have also shown the results for sequential DML processing (not employing intra-DML parallelism). Intra-DML parallelism permitted only small response time improvements for up to 4 PE (speedup factor 1.5); additional PE resulted in an increased response time. Surprisingly, even with a sequential DML processing a response time improvement could be obtained for up to 4 PE (by about 20%) despite the communication delay for remote database accesses and two-phase commit. This was again because the aggregate buffer size increases with the number of nodes permitting a significant reduction of the I/O delay. Since this effect also happened for intra-DML parallelism, the actual response time improvements due to parallel query processing correspond to the difference to the results for sequential processing.

One reason for the modest response time improvements is the small potential for intra-DML parallelism in the workload. In addition, the number of remote requests per operation increased with the number of nodes without resulting in a corresponding increase of intra-DML parallelism. This is because a transaction's object references may be spread over the entire database and processing may thus involve the majority of PE while intra-DML parallelism is limited by the number of references per DML statement. Another problem typical for real-life applications was that the suboperations of a DML statement differ widely in the number of their object references. Since the execution time is determined by the slowest suboperation, the benefit of intra-DML parallelism is further reduced.

Finally, we analyse the response time impact of multi-user mode for DOA. The results in Fig. 7 were obtained for arrival rates of 1-32 TPS per PE and with the use of intra-DML parallelism. Increasing inter-transaction parallelism caused an even higher reduction of the effectiveness of intra-DML parallelism than for the relational workload (Fig. 5). Only for comparatively low arrival rates (system underload) slight response time improvements could be achieved for 2 to 4 PE. For higher arrival rates the increased CPU contention (communication overhead) caused higher response times than for 1 PE and this effect became worse with an increasing number of PE. Another problem for the DOA workload was that it was not possible to achieve a similarly uniform CPU utilization in all nodes as for the synthetically generated (homogeneous) workloads.

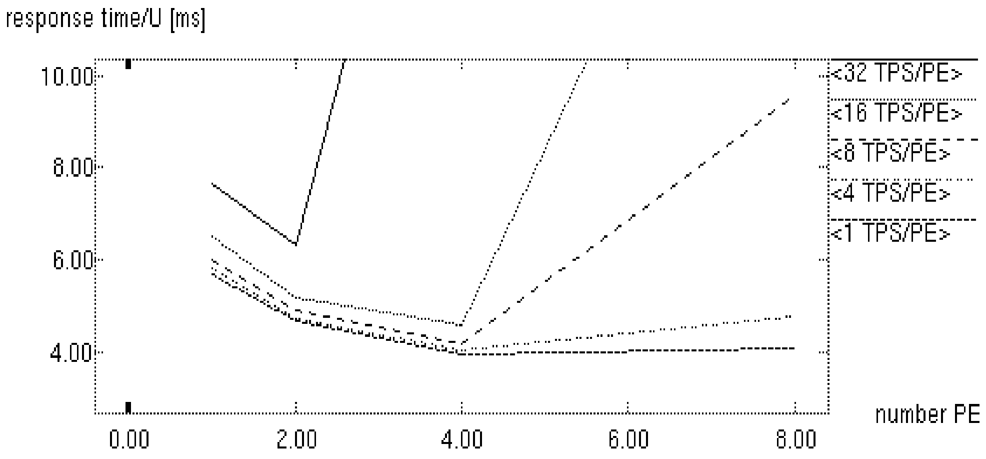

Fig. 7: DOA response times with inter-transaction and intra-DML parallelism

Our results indicate some of the problems that need to be addressed for real-life applications with both OLTP transactions and complex queries. In particular finding appropriate workload and database allocations is difficult for workloads such as DOA, especially if a larger number of PE should be utilized. The use of intra-DML parallelism reduces OLTP throughput due to the extra communication overhead for parallelism, while the effectiveness of intra-transaction parallelism is also impaired by inter-transaction parallelism due to increased resource contention. Further performance problems are possible due to lock conflicts between OLTP transactions and complex queries unless special concurrency control schemes (e.g. multiversion concurrency control) are supported that allow a processing of read-only transactions without locking.

4 Summary

We have presented a performance evaluation of shared nothing architectures for distributed and parallel transaction processing using a detailed simulation system. Different workload types were considered representing a wide spectrum of database applications. The focus has been on studying the effectiveness of intra-DML parallelism in single-user as well as in multi-user mode (inter-transaction parallelism). To evaluate scalability we used response time speedup and throughput scaleup as our primary performance metrics. In addition, we investigated workload allocation aspects and the influence of storage (index) structures for parallel query processing.

For OLTP workloads like debit-credit, we have shown that it is generally easy to achieve linear throughput scaleup. This simple workload often used in DBMS and OLTP benchmarks permits an ideal partitioning of the database and workload so that the communication overhead per transaction remains small and almost independent of the number of nodes. Although a random assignment of transactions results in a higher number of messages, an almost linear throughput scaleup (at a lower level) can be achieved even in this case. A (modest) response time improvement is possible for debit-credit without introducing extra messages by employing inter-DML parallelism.

For complex relational queries, an almost linear response time speedup could be obtained in single-user mode by employing intra-DML parallelism. However, this is generally possible only until a certain number of nodes since the ratio between communication overhead an useful work per subquery deteriorates when the number of processing elements grows. The greatest improvements by employing intra-DML parallelism were observed for queries with high response times in the central case (1 PE). On the other hand, only a modest potential for parallelization is given for queries with a high selectivity or when an index can be used for query evaluation. The effectiveness of intra-DML parallelism was substantially lower in multi-user mode where CPU capacity is more constrained and the communication overhead introduced by parallel query processing increasingly causes resource contention. As a result, response time improvements were much smaller than in single-user mode and could be achieved only for a more restricted number of nodes. Furthermore, the high communication overhead for parallel query processing significantly reduced transaction rates and prevented a linear throughput growth.

For the real-life workloads represented by database traces, response time improvements by using intra-DML parallelism were even lower. This was in part because the number of database accesses per DML statement was comparatively small, but also because of the difficulty to find a "good" database and workload allocation for such applications. The real workloads consist of multiple transaction types and database files with a highly non-uniform and heterogeneous reference behavior. In particular, the majority of references is by a few dominating transaction types and on few database files. These reference characteristics allowed an appropriate database and workload allocation only for a small number of nodes; when increasing the number of nodes further, the communication requirements per transaction increase without enabling a higher degree of intra-DML parallelism. More research is needed to determine database and workload allocation strategies for mixed applications for which both a largely local processing of OLTP transactions as well as intra-DML parallelism for complex queries is to be supported.

References

An85 Anonymous et al.: A Measure of Transaction Processing Power. *Datamation*, 112-118 (April 1985).

Bo90 Boral, H. et al.: Prototyping Bubba: A Highly Parallel Database System. *IEEE Trans. on Knowledge and Data Engineering 2,* 1, 4-24 (1990).

CL89 Carey, M.J., Livny, M.: Parallelism and Concurrency Control Performance in Distributed Database Machines. *Proc. ACM SIGMOD Conf.* (1989).

De90 DeWitt, D.J. et al. 1990. The Gamma Database Machine Project. *IEEE Trans. on Knowledge and Data Engineering 2* ,1, 44-62 (1990).

DGS88 DeWitt, D.J., Ghandeharizadeh, S., Schneider, D.A.: A Performance Analysis of the Gamma Database Machine. *Proc. ACM SIGMOD Conf.* (1988).

EGKS90 Englert, S., Gray, J., Kocher, T., Shath, P.: A Benchmark of NonStop SQL Release 2 Demonstrating Near-Linear Speedup and Scale-Up on Large Databases. *Proc. ACM SIGMETRICS Conf.*, 245-246 (1990).

HR83 Härder, T., Reuter, A.: Principles of Transaction-Oriented Database Recovery. *ACM Computing Surveys 15* (4), 287-317 (1983).

JTK89 Jenq, B.P., Twichell, B., Keller, T.: Locking Performance in a Shared Nothing Parallel Database Machine. *IEEE Trans. on Knowledge and Data Engineering 1* (4) (1989).

Ke88 Kersten, M. et al.: A Distributed, Main-Memory Database Machine. In: Database Machines and Knowledge Base Machines (*Proc. 5th Int. Workshop on Database Machines, 1987*), North-Holland, 353-369 (1988).

Li87 Livny, M.: DeLab - A Simulation Laboratory. *Proc. Winter Simulation Conf.*, 486-494 (1987).

Li89 Livny, M.: DeNet Users's Guide, Version 1.5, Computer Science Department, University of Wisconsin, Madison (1989).

Ma91 Marek, R.: Simulation of a Shared-Nothing System for Parallel Query Processing. Master's Thesis (in German), Univ. Kaiserslautern, Dept. of Comp. Science (1991).

MLO86 Mohan, C., Lindsay, B., Obermarck, R.: Transaction Management in the R* Distributed Database Management System. *ACM Trans. on Database System 11* (4), 378-396 (1986).

Ne86 Neches, P.M.: The Anatomy of a Database Computer - Revisited. *Proc. IEEE CompCon Spring Conf.*, 374-377 (1986).

ÖV91 Özsu, M.T., Valduriez, P.: Principles of Distributed Database Systems. Prentice Hall (1991).

Pi90 Pirahesh, H. et al.: Parallelism in Relational Data Base Systems: Architectural Issues and Design Approaches. In *Proc. 2nd Int.Symposium on Databases in Parallel and Distributed Systems* , IEEE Computer Society Press (1990).

Ra86 Rahm, E.: Algorithms for efficient load control in multi-system DBMS. *Angewandte Informatik* 4/86 (in German), 161-169 (1986).

SD89 Schneider, D.A., DeWitt, D.J.: A Performance Evaluation of Four Parallel Join Algorithms in a Shared-Nothing Multiprocessor Environment. *Proc. ACM SIGMOD Conf.,* 110-121 (1989)

St86 Stonebraker, M.: The Case for Shared Nothing. *IEEE Database Engineering 9* (1), 4-9 (1986).

Ta88 The Tandem Performance Group: A Benchmark of NonStop SQL on the Debit Credit Transaction. *Proc. ACM SIGMOD Conf.*, 337-341 (1988).

Ta89 The Tandem Database Group: NonStop SQL, A Distributed, High-Performance, High-Availability Implementation of SQL. In Lecture Notes in Computer Science 359, Springer-Verlag, 60-104 (1989).

TPC89 Transaction Processing Performance Council (TPC): TPC Benchmark A. (1989).

WT91 Watson, P., Townsend, P.: The EDS Parallel Relational Database System. In: "Parallel Database Systems", LNCS 503, Springer-Verlag, 149-168 (1991).