6 References

DG92 DeWitt, D.J., Gray, J.: Parallel Database Systems: The Future of High Performance Database Systems. *Comm. ACM* 35 (6), 85-98, 1992

Gh90 Ghandeharizadeh, S.: Physical Database Design in Multiprocessor Database Systems. Ph.D. thesis, Univ. of Wisconsin-Madison, Sep. 1990

Li93 Linder, B.: Oracle Parallel RDBMS on Massively Parallel Systems. *Proc. PDIS 93*, 67-68, 1993

MN91 Mohan, C., Narang, I.: Recovery and Coherency-control Protocols for Fast Intersystem Page Transfer and Fine-Granularity Locking in a Shared Disks Transaction Environment. *Proc. 17th VLDB Conf.*, 193-207, 1991

PGK88 Patterson, D.A., Gibson, G., Katz, R.H.: A Case for Redundant Arrays of Inexpensive Disks (RAID). *Proc. ACM SIGMOD Conf.,* 109-116, 1988

Pi90 Pirahesh, H. et al.: Parallelism in Relational Data Base Systems: Architectural Issues and Design Approaches. In *Proc. 2nd Int.Symp. on Databases in Parallel and Distributed Systems*, 1990

Ra86 Rahm, E.: Primary Copy Synchronization for DB-Sharing. *Information Systems* 11 (4), 275-286, 1986

Ra91 Rahm, E.: Concurrency and Coherency Control in Database Sharing Systems, Techn. Report 3/91, Univ. Kaiserslautern, Dept. of Comp. Science, Dec. 1991

Ra93a Rahm, E.: Empirical Performance Evaluation of Concurrency and Coherency Control for Database Sharing Systems. *ACM Trans. on Database Systems* 18 (2), 333-377, 1993

Ra93b Rahm, E.: Parallel Query Processing in Shared Disk Database Systems. *Proc. 5th Int. Workshop on High Performance Transaction Systems (HPTS-5),* Asilomar, Sep. 1993 (Extended Abstract:: *ACM SIGMOD Record* 22 (4), Dec. 1993)

RM93 Rahm, E., Marek, R.: Analysis of Dynamic Load Balancing Strategies for Parallel Shared Nothing Database Systems. *Proc 19th VLDB Conf.*, 182-193, 1993

RM94 Rahm, E., Marek, R.: Dynamic Multi-Resource Load Balancing in Parallel Database Systems. Techn. Report, Univ. of Kaiserslautern, 1994

RMW93 Reiner, D., Miller, J., Wheat, D.: The Kendall Square Query Decomposer. *Proc. Spring CompCon*, 300-302, 1993 (also: Proc. PDIS93)

Se93 Selinger, P.: Predictions and Challenges for Database Systems in the Year 2000. *Proc 19th VLDB Conf.*, 667-675, 1993

SL91 Seeger, B., Larson, P.: Multi-Disk B-trees. *Proc. ACM SIGMOD Conf.*, 436-445, 1991

Va93 Valduriez, P.: Parallel Database Systems: Open Problems and New Issues. *Distr. and Parallel Databases* 1 (2), 137-165, 1993

Yu87 Yu, P.S. et al.: On Coupling Multi-systems through Data Sharing. *Proceedings of the IEEE* 75 (5), 573-587, 1987

utilized. This flexibility for dynamically controlling disk contention in multi-user mode is not supported by the SN architecture.

While we believe that disks constitute the most significant bottleneck resource for parallel query processing, in future work we will study additional bottleneck resources, in particular CPU, memory and network. Furthermore, we want to study parallel processing of other relational operators (e.g., joins, etc.) in SD systems. The impact of concurrency and coherency control on parallel query processing also needs further investigation.

5 Conclusions

We have presented a performance analysis of parallel scan processing in Shared Disk (SD) database systems. In contrast to Shared Nothing (SN), SD offers a high flexibility for scan processing because the number of subqueries is not predetermined by the degree of declustering (D) but can be chosen with respect to the query characteristics (relation scan, clustered index scan or non-clustered index scan, selectivity, etc.) as well as the current load situation (e.g., disk utilization, CPU utilization, etc.). Furthermore, the scan processors themselves can be selected dynamically to achieve load balancing.

However, even in single-user mode the effectiveness of intra-query parallelism can be reduced by disk contention between subqueries. We found that this problem primarily exists for clustered index scans where the relevant index and data pages typically reside on a single disk. Hence, clustered index scans are best processed sequentially unless the data of multiple disks needs to be accessed. In this case, the number of disks to be accessed determines the maximal degree of parallelism. On the other hand, parallel processing of relation scans permits optimal speedup in single-user mode by assigning the subqueries to disjoint sets of disks. This is easily feasible by choosing the degree of parallelism P such that P = k*D. Parallel processing of non-clustered index scans is also quite effective if the relation is declustered across a sufficiently large number of disks (e.g., D=n). Disk contention on the index cannot generally be avoided but is typically less significant for a larger number of data pages to be accessed[*]. A general observation is that physical declustering of relations and indices could effectively be used for parallel query processing indicating that SD database systems can make good use of disk arrays.

Multi-user mode inevitably leads to increased disk contention and therefore requires higher degrees of declustering if an effective intra-query parallelism is to be supported. Prefetching was found to be very effective for relation scans not only to improve response times, but also to reduce disk contention and to support smaller degrees of declustering, in particular in multi-user mode. Even for a high degree of declustering (e.g., D=4n), high arrival rates can lead to significant levels of disk contention and thus high response times for both complex queries and OLTP transactions. In such situations, we found it necessary to choose smaller degrees of intra-query parallelism to limit disk contention and response times degradations. In particular, the degree of scan parallelism should be chosen the smaller the higher the disks are

[*] Of course, selective index scans accessing only few data pages should be processed sequentially.

query arrival rate of 0.5 QPS did not cause any significant response time degradations for OLTP, this was no longer true for 1 QPS. In this case, OLTP response times deteriorate proportionally to the degree of scan parallelism due to increased disk contention. This shows that limiting the degree of intra-query parallelism is not only necessary for obtaining good throughput, but also for limiting the performance penalty for OLTP transactions that have to access the same disks. Furthermore, keeping OLTP response times acceptably small may require a lower degree of scan parallelism than the one minimizing query response time.

found that disk utilization must be rather high (> 50%) before varying the degree of scan parallelism has a significant impact on performance.

In our final experiment, we studied a heterogeneous workload consisting of relation scans and OLTP transactions. This experiment is based on the same configuration than before (n=16, D=64) but introduces disk contention between OLTP transactions and relation scans. Each OLTP transaction randomly accesses four data pages from the D disks. A fixed OLTP arrival rate was chosen such that it causes an average disk utilization of about 25%. In addition to this base load, we process relation scans with arrival rates of 0.5 QPS and 1 QPS. The resulting response times for different degrees of parallelism for the relation scans are shown in Fig. 9. For the queries (left diagram), we observe a similar response time behavior than for the homogeneous workload. In particular, for higher query arrival rate (disk contention) only a limited degree of scan parallelism proves useful. While P=8 achieved the best response time for 1 QPS and without OLTP load, the optimum is now achieved for P=4. This underlines that the degree of scan parallelism should be chosen according to the current disk utilization, irrespective of whether disk contention is due to concurrent OLTP transactions or other queries.

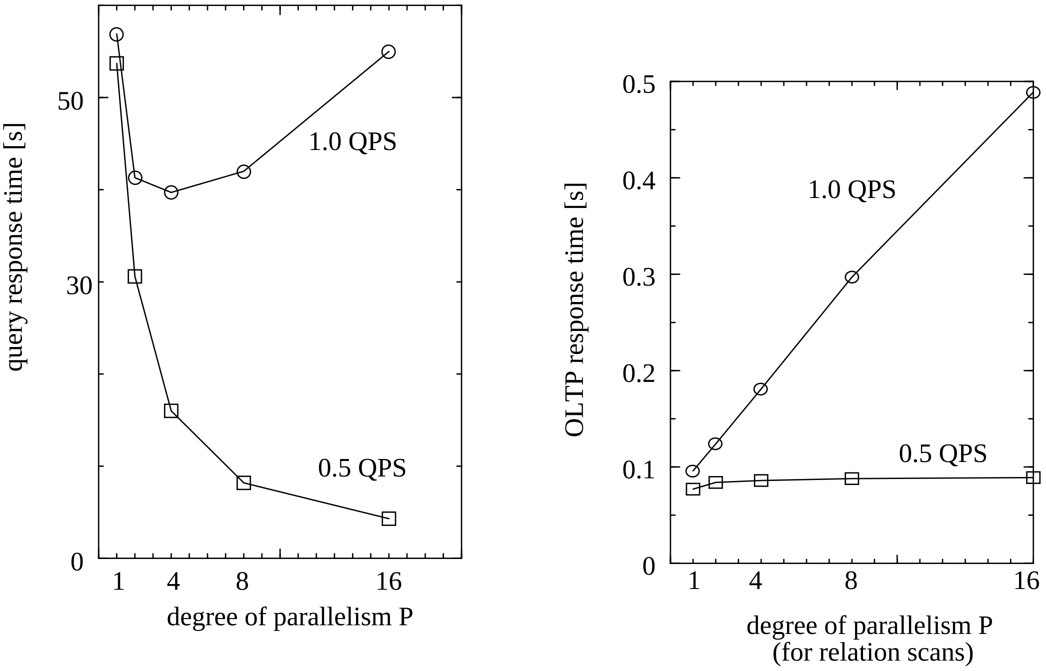


Fig. 9: Response times for mixed query/OLTP workload

OLTP response times (right diagram of Fig. 9) are very sensitive to the number of concurrent scan queries as well as the degree of intra-query parallelism. While a is increased proportionally to n. We now study the impact of the degree of parallelism P for different arrival rates and a fixed number of nodes and disks. This experiment is performed for relation scans using prefetching and a system of 16 nodes and 64 disks.

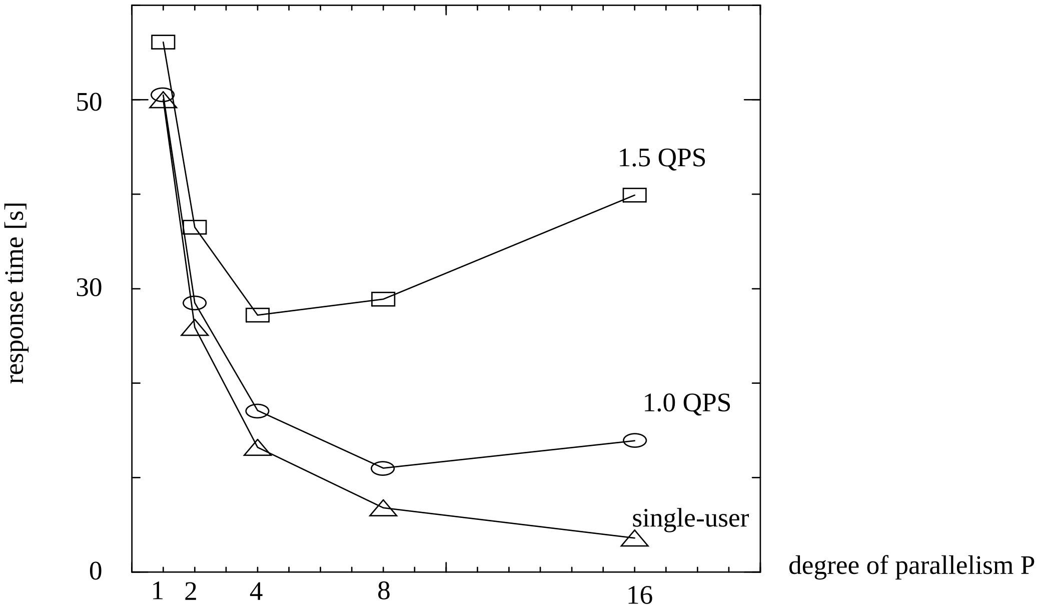
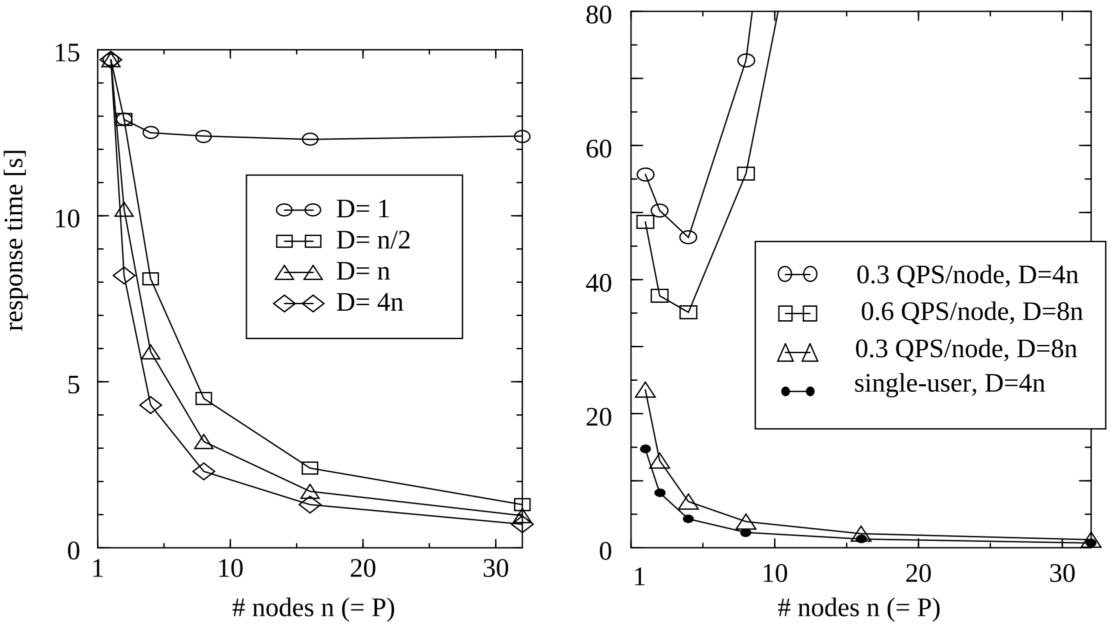


Fig. 8: Degree of parallelism vs. arrival rate (n=16, D=64)

Fig. 8 shows that for sequential processing (P=1) multi-user response times are only slightly higher than in single-user mode, but that the effectiveness of intra-query parallelism decreases with growing arrival rates. In single-user mode response times continuously improve with increasing degrees of parallelism and reach their minimum for P=16. For an arrival rate of 1 QPS and 1.5 QPS, on the other hand, the response time minimum is achieved for P=8 and P=4, respectively. Further increasing the degree of parallelism causes a response time degradation, in particular for the higher arrival rate 1.5 QPS. These results show that the optimal degree of scan parallelism depends on the current system state, in particular the level of disk contention. Under low disk contention (single-user mode or low arrival rates), intra-query parallelism is most effective and achieves good speedup values even for higher degrees of scan parallelism, e.g., P=n. However, the higher the disks are utilized due to inter-query parallelism the lower the optimal degree of intra-query parallelism becomes. Hence, there is a need for dynamically determining the degree of scan parallelism according to the current system and disk utilization. Note that such a dynamic query scheduling approach is feasible for Shared Disk, but not for Shared Nothing. Hence, Shared Disk is better able to limit disk contention in multi-user mode by reducing the degree of intra-query parallelism accordingly. However, we fers better response times than a 32-way parallel non-clustered index scan. On the other hand, the non-clustered index scan remains always better than a relation scan with prefetching (Fig. 6b) although the differences between the two approaches become smaller for larger degrees of parallelism.



a) Single-user mode

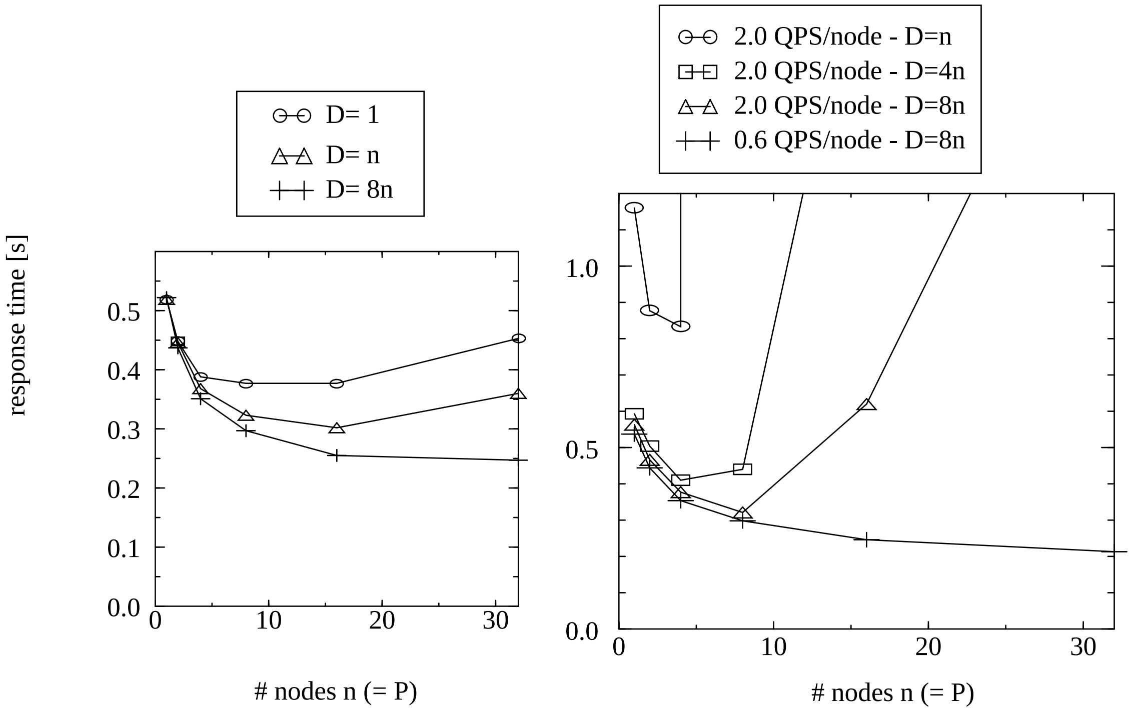
b) Multi-user mode

Fig. 7: Performance of non-clustered index scan

The multi-user results (Fig. 7b) illustrate that the high I/O requirements of non-clustered index scans allow for significantly lower throughput than clustered index scans. While we could support 0.6 QPS for up to 32 nodes and D=8n without problems for clustered index scans (Fig. 6b), this arrival rate causes significant disk contention for non-clustered index scans and cannot be supported for more than 8 nodes. Put differently, non-clustered index scans require a much higher degree of declustering to meet a certain throughput. Similarly as for relation scans (Fig. 5), in multi-user mode the effectiveness of intra-query parallelism is much smaller than in single-user mode. Increasing the degree of intra-query parallelism while increasing the workload proportionally with n, is only effective for comparatively low disk utilization, i.e., for low arrival rates or few processors.

4.4 The need for dynamic query scheduling

The experiments discussed so far always used the maximal degree of intra-query parallelism P=n. In combination with inter-query parallelism this caused a high level of disk contention for a larger number of processors even when the number of disks



b) Single-user mode

b) Multi-user mode

Fig. 6: Performance of clustered index scan

that selective clustered index scans should be processed sequentially to support high throughput. A small degree of intra-query parallelism may be useful if the data of multiple disks needs to be processed.

For *non-clustered index scans* prefetching is not employed since the result tuples may be spread over many disks. In single user-mode (Fig. 7a), the sequential response time for non-clustered index scan is about a factor 10 better than for the relation scan without prefetching (Fig. 4a) since we have to access about 10% of the data pages. In contrast to clustered index scans, parallel processing of non-clustered index scan is rather effective if the relation is declustered across at least n/2 disks (speedup of 15 for P=32). This is because accesses to the data pages are spread across all disks so that much smaller disk contention arises. However, in contrast to parallel relation scan processing disk contention cannot completely be eliminated even for D=n because of index accesses (not all leaf index pages could be cached). Furthermore, it cannot be excluded that subqueries have to access data pages on the same disks although the probability of this event becomes smaller with higher degrees of declustering. For these reasons, a declustering factor of 4n provides slightly better response times than D=n in single-user mode.
Note however, that a sequentially processed clustered index scan (Fig. 6a) still of-

4.3 Parallel processing of index scans

We now focus on the performance of parallel index scans in single- and multi-user mode. For our relation (125,000 tuples) we use a 3-level B^+ tree with 625 leaf pages. A range query with scan selectivity of 1% thus requires access to 2 higher-level index pages and 7 leaf pages. The number of additional accesses to data pages for the 1250 result tuples depends on whether a clustered or non-clustered index is used. For the clustered index scan, the tuples are stored in 125 consecutive data pages while up to 1250 different data pages may have to be accessed for the non-clustered index scan. For parallel index scan processing, we assume that the range condition on the index attribute can be decomposed into P smaller range conditions so that each subquery has to access the same number of tuples.

We first analyze the performance of *clustered index scans* (Fig. 6). In this case, we always use prefetching for data pages. The number of nodes n and the degree of parallelism are again varied from 1 to 32. In single-user mode, we study the following degrees of declustering: D=1, D=n/2 and D=n. In multi-user mode, we consider different arrival rates for a homogeneous load of clustered index scans only. Furthermore, the number of disks is up to 8 times higher than the number of processors. The index is always declustered across the same disks than the relation's data pages.

Let's first look at the single-user response times (Fig. 6a). Sequentially processing the clustered index scan achieves an average response time roughly 100-times better as for the relation scan with prefetching (due to the scan selectivity of 1%). However in contrast to the relation scan (Fig. 4b), intra-query parallelism is little useful for the clustered index scan not only for D=1, but also for D=n and even for D=8n. This is because in most cases the relevant index and data pages reside also on only one disk due to the clustering according to the index attribute (e.g., for D=32 we have about 390 data pages per disk compared to 125 relevant data pages). The small improvement of D=n over D=1 comes from the fact that the relevant pages for some queries may be on two instead of one disk (the probability of this case increases with D). D=8n offers a small improvement for n ≥ 16 since the data of multiple disks needs to be processed in this range. Still, compared to sequential processing only a speedup of 2 is achieved which is clearly not cost-effective.

Multi-user mode (Fig. 6b) leads to increased disk contention so that only modest arrival rates are attained if intra-query parallelism is used. For instance, an arrival rate of 2 QPS per node cannot be sustained for more than a few nodes even if we increase the number of disks proportionally with n (e.g., D=n or D=4n). This shows

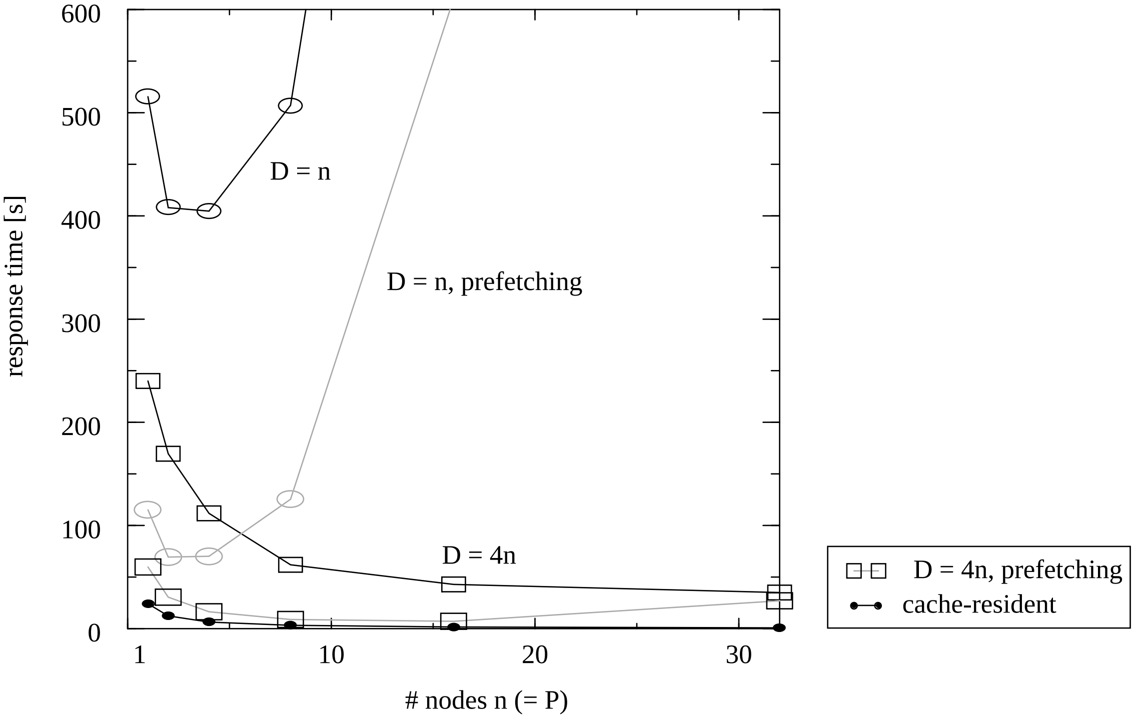
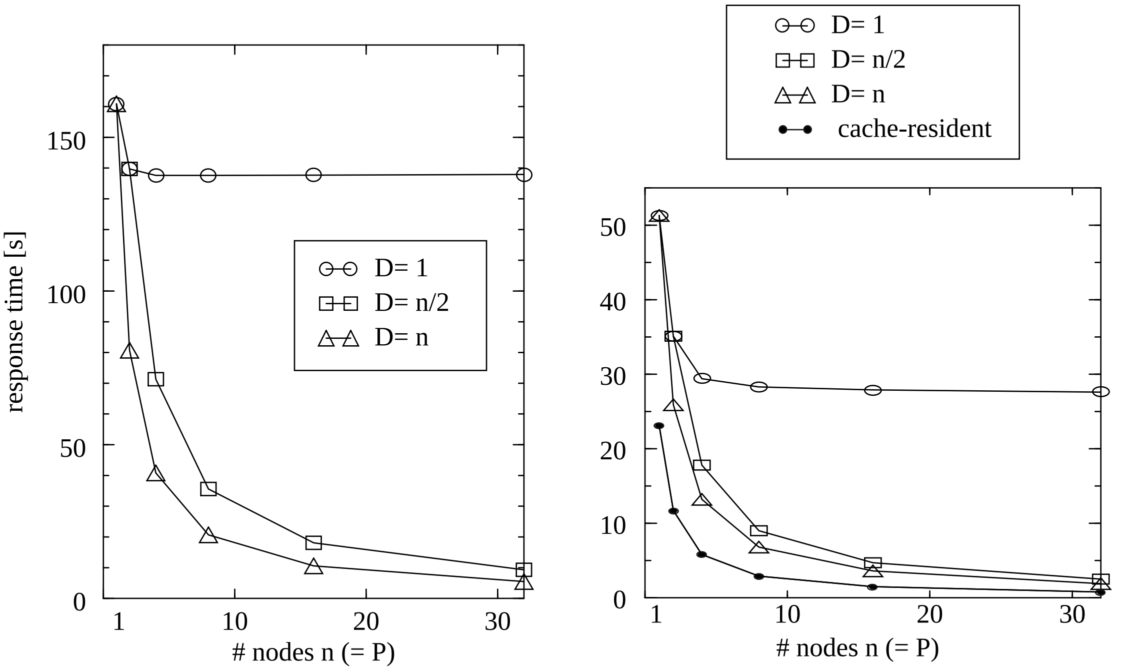


Fig. 5: Multi-user performance of relation scan

We observe that for D=n, response times are several times higher than in single-user mode (Fig. 4) due to disk waits. Parallel scan processing only allows for very modest response time improvements for 2-4 processors (speedup of 1.25). More nodes lead to significantly aggravated disk contention because we increase both the degree of inter-query (arrival rate) and the degree of intra-query parallelism linearly with n. As a result, the load can no longer be processed for more than 8 nodes and D=n. As Fig. 5 shows the disk bottleneck is largely removed for our arrival rate if we decluster the relation across 4 times as many disks as there are processors (D=4n). While prefetching cannot prevent the disk bottleneck for D=n and more than 8 nodes, it allows for substantially improved response times (factor 5). Furthermore, for up to 4 processors its response times for D=n are better than without prefetching and the four-fold number of disks! For D=4n, prefetching allows us to approach the optimal response times of the cache-resident case. These results demonstrate that multi-user mode requires substantially higher degrees of declustering than single-user mode to keep response times acceptable and to achieve linear throughput increase. Furthermore, prefetching is even more valuable in multi-user mode to keep disk contention low and to limit the number of disks.

nificant response time improvement for P>2. On the other hand, having one disk per subquery (D=n) avoids any disk contention for relation scan in single-user mode allowing optimal response time speedup. A declustering across n/2 disks is significantly better than D=1, but still suffers from disk contention in particular for smaller degrees of parallelism (P ≤ 8).



a) no prefetching

b) prefetching

Fig. 4: Single-user performance of relation scan

Prefetching (Fig. 4b) is very effective for both sequential and parallel processing of relation scans. Not only response times are significantly reduced, but also disk utilization (55% for P=1). This lowers disk contention and supports smaller degrees of declustering. Even for D=1, response times can be improved for up to 4 nodes and a speedup of 1.7 is achieved. Response times for D=n/2 are almost as good as for D=n thus permitting the use of fewer disks.

For the *multi-user experiment* (Fig. 5) we study a homogeneous workload of relation scans on the same relation. The arrival rate is increased proportionally to the number of nodes because we want to support both short response times as well as linear throughput increase. We used an arrival rate of 0.07 queries per second (QPS) per node resulting in a CPU utilization of about 30%. We found that this arrival rate cannot be processed if we have fewer disks than processors (D < n) due to disk over-utilization. The response time results in Fig. 5 refer to the cases of D=n and D=4n and with or without prefetching. For comparison, we again show the results for a cache-resident relation (no disk I/O).

Configuration	settings	Database/Queries	settings
number of nodes (n)	1, 2, 4, 8, 16, 32	**relation :**	(100 MB)
#processors per node (m)	1	#tuples	125.000
CPU speed per processor	30 MIPS	tuple size	800 B
		blocking factor	10 (data), 200 (index)
avg. no. of instructions:		index type	clustered / non-cl. B$^+$-tree
BOT	25000	storage allocation	disk
EOT	25000	degree of declustering D	varied
I/O initialization	3000		
scan object reference	1000		
send short message (128 B)	1000	**scan queries:**	
receive short message	1000	scan type	relation scan / clustered index scan / non-clustered index scan
send long message (page)	5000		
receive long message	5000	scan selectivity	1.0 %
		no. of result tuples	1250
buffer manager:		size of result tuples	800 B
page size	8 KB	arrival rate	single-user, multi-user (varied)
buffer size per node	500 pages	query placement	random (uniformly over all nodes)
disk devices:		scan parallelism P	varied
controller service time	1 ms (per page)		
# prefetch pages	8 pages		
avg. disk access time (1 page)	11 ms		
avg. disk access time (prefetching)	18 ms		
cache size	1000 pages		
communication bandwidth	20 MB/s		

Fig. 3: System configuration, database and query profile.

4.2 Parallel processing of relation scan

We first study the performance of relation scans in *single-user mode* for the cases without prefetching (Fig. 4a) and with prefetching of pages into the disk cache (Fig. 4b). We vary the number of nodes n from 1 to 32 and use one subquery per node (i.e., P=n) since we assumed a single processor per node. Three cases are considered for declustering the input relation. A degree of declustering D=1 refers to the case where the entire relation is stored on a single disk, while D=n assumes a declustering of the relation across n (=P) disks. D=n/2 assumes two processors per disk for n ≥ 2 (1 disk for n =1). For comparison purposes, we have also shown in Fig. 4b the results where the entire relation fits into the disk cache (or is kept in a solid-state disk).

The results for the cache-resident case show that response times are indeed dominated by disk access times. This is particularly true without prefetching (Fig. 4a) where response times are up to a factor 5 (for D=1) higher than with prefetching. The results show that storing the entire relation on a single disk (D=1) makes parallel scan processing useless since disk utilization is already 85% for sequential scan processing (P =1). Increasing the number of subqueries improves the CPU-related response time portion, but completely overloads the disk preventing any sig-

4 Performance Analysis

Our experiments concentrate on the impact of disk contention on the performance of parallel scan processing in SD database systems. For this purpose, we study the relationship between the degree of declustering D and the degree of parallelism P for both single-user and multi-user mode as well as for relation scans, clustered and non-clustered index scans. We additionally investigate the use of prefetching pages to improve performance. Furthermore, we show that the SD architecture allows us to control disk contention in multi-user mode by a dynamic query scheduling approach that determines the degree of scan parallelism based on the current system state.

In the next subsection, we provide an overview of the parameter settings used in the experiments. Afterwards, we analyze the performance of parallel relation scans (4.2) and index scans (4.3) for different values of D and P in single- and multi-user mode. Finally, we describe experiments for homogeneous and heterogeneous workloads showing the need for dynamically determining the degree of parallelism P based on the current disk contention.

4.1 Simulation Parameter Settings

Fig. 3 shows the major database, query and configuration parameters with their settings. Most parameters are self-explanatory, some will be discussed when presenting the simulation results. The scan queries used in our experiments access a 100 MB relation with 125.000 tuples. In the case of index scans, only 1% of the tuples is accessed (scan selectivity). Relation scans also generate a result set of 1250 tuples, but must access the entire relation. The number of processing nodes is varied between 1 and 32.

The duration of an I/O operation is composed of the controller service time, disk access time and transmission time. For sequential I/Os (e.g. relation scans, clustered index scans), prefetching can be chosen resulting in an average access time of 18 ms for 8 pages rather than 8*11 ms if the pages were read one by one. For message and page transfers we assume a communication bandwidth of 20 MB/s and that no bottlenecks occur in the network. This assumption is justified by the comparatively small bandwidth requirements of our load as well as by the fact that we focus on disk contention in this study.

To capture the behavior of OLTP-style transactions, we provide a workload similar to the debit-credit benchmark. Each OLTP transaction randomly accesses four data pages from the same disks accessed by the scan queries.

the GLA for most of the objects to be referenced. For OLTP transactions, such an affinity-based routing can be implemented by a table indicating for each transaction type the preferred nodes. Furthermore, the lock overhead can be spread among all nodes in contrast to a centralized locking scheme.

For page-level locking, coherency control can efficiently be combined with the locking protocol by extending the global lock tables with information (e.g., time stamps) to detect invalid page copies. We have implemented such an on-request invalidation approach since it allows us to detect obsolete pages during lock request processing without extra communication. To propagate updates in the system, we assume that each node acts as the "owner" for the database partition for which it holds the GLA. The owner is responsible of providing other nodes with the most recent version of pages of its partition and for eventually writing updated pages to disk. With this approach, an updated page is transferred to the owner at transaction commit if it has been modified at another node. This page transfer can be combined with the message needed for releasing the write lock. Similarly, page transfers from the owner to another node are combined with the message to grant a lock [Ra86, Ra91].

To support query processing, we have implemented a hierarchical version of this protocol with relation- and page-level locking[*]. Relation-level locking is used for relation scans and larger index scans because page-level locking could cause an extreme overhead in these cases. Page-level locking is applied for selective queries accessing only few pages. Relation locks are acquired by the coordinator before the subqueries are started and released after the end of all subqueries. In the lock grant message for a relation lock, the global lock manager indicates all pages of the relation for which an invalidation is feasible at the nodes where the query is to be executed. The respective pages are immediately removed from the buffers and requested from the owner during the execution of the subqueries.

[*] Since the GLA for a relation can be partitioned among several nodes in our implementation, we in fact support the additional lock granularity of a relation fragment, consisting of all tuples/pages of a relation for which one node holds the GLA. To simplify the description, we assume here that the GLA for each relation (and for each index) is assigned to only one node.

results. Locks may be requested either by the coordinator before starting the subqueries or by the individual subqueries. Similarly, all locks may be released by the coordinator or by the individual subqueries (see below).

The number of CPUs per node and their capacity (in MIPS) are provided as simulation parameters. The average number of instructions per request can be defined separately for every request type. To accurately model the cost of transaction/query processing, CPU service is requested for all major steps, in particular for transaction initialization (BOT), object accesses in main memory, I/O overhead, communication overhead, and commit processing. The communication network models transmission of "long" messages (page transfers) and "short" messages (e.g., global lock request). Query result sets are disassembled into the required number of messages (long or short).

The database buffer in main memory is managed according to a LRU replacement strategy and a no-force update strategy with asynchronous disk writes. The buffer manager closely cooperates with the concurrency control component to implement coherency control (see below).

Database partitions (relations, indices) can be declustered across several disks as discussed above. Disks and disk controllers have explicitly been modelled as servers to capture potential I/O bottlenecks. Furthermore, disk controllers can have a LRU disk cache. The disk controllers also provide a prefetching mechanism to support sequential access patterns. If prefetching is selected, a disk cache miss causes multiple succeeding pages to be read from disk and allocated into the disk cache. Sequentially reading multiple pages is only slightly slower than reading a single page, but avoids the disk accesses for the prefetched pages when they are referenced later on. The number of pages to be read per prefetch I/O is specified by a simulation parameter and can be chosen per query type.

Concurrency and coherency control

For concurrency and coherency control, we have implemented a primary copy locking (PCL) scheme [Ra86] because this scheme has performed best in a comprehensive, trace-driven performance study of several concurrency/coherency control schemes [Ra93a]. PCL partitions the global lock authority (GLA) for the database among all processing nodes so that each node handles all global lock requests for one database partition. Hence, communication is only required for those lock requests belonging to the partition of a remote node. With this scheme, a large portion of the locks can locally be processed by assigning a transaction to the node holding tered at the page level across an arbitrary number of disks. Declustering of relations is straight-forward. If B is the number of pages per disk (B = relation size in pages / declustering factor D), we simply assign the first B pages to the first disk, pages B+1 to 2B to the second disk and so on. Indices are not partitioned as in SN systems, but have the same structure as in centralized DBMS (only one root page, etc.). Each index level is separately declustered across D disks similarly to relation declustering (the root page is on a single disk, of course).

We support heterogeneous (multi-class) workloads consisting of several query and transaction types. Queries correspond to transactions with a single database operation (e.g., SQL statement). We support the following scan query types: relation scan, clustered index scan and non-clustered index scan. We also support the debit-credit benchmark workload (TPC-B) and the use of real-life database traces. The simulation system is an open queuing model and allows definition of an individual arrival rate for each transaction and query type.

Workload allocation takes place at two levels. First, each incoming transaction or query is assigned to one processing node acting as the coordinator for the transaction/ query. For this placement we support different strategies, in particular random allocation. Furthermore, we can allocate transaction and query types to a subset of the processing nodes allowing us to assign OLTP transactions and complex queries to disjoint sets of nodes. The second form of workload allocation deals with the assignment of suboperations to processors for parallel query processing. These assignment can be made statically (e.g. random) or dynamically based on the current processor utilization. The number of subqueries (degree of intra-query parallelism) can also be chosen statically or dynamically, e.g., based on the current disk utilization. Details are provided in the next section.

Processing model

Each processing node of the Shared Disk system is represented by a transaction and query manager, CPU servers, a communication manager, a buffer manager, and a concurrency/coherency control component (Fig. 2). The transaction and query manager controls the execution of transactions and queries. The maximal number of concurrent transactions and (sub)queries (inter-transaction parallelism) per node is controlled by a multiprogramming level. Newly arriving transactions and queries must wait in an input queue until they can be served when this maximal degree of inter-transaction parallelism is already reached. Parallel query processing entails starting all subqueries, executing the individual subqueries and merging their

3 Simulation model

For the present study, we have implemented a comprehensive simulation model of a Shared Disk database architecture. The gross structure of this simulation system is depicted in Fig. 2. In the following, we briefly describe the used database and workload models as well as the processing model. Furthermore, we outline the implemented strategy for concurrency/coherency control. The simulation system is highly parameterized. In Section 4.1, we will provide an overview of the major parameters and their settings used in this study.

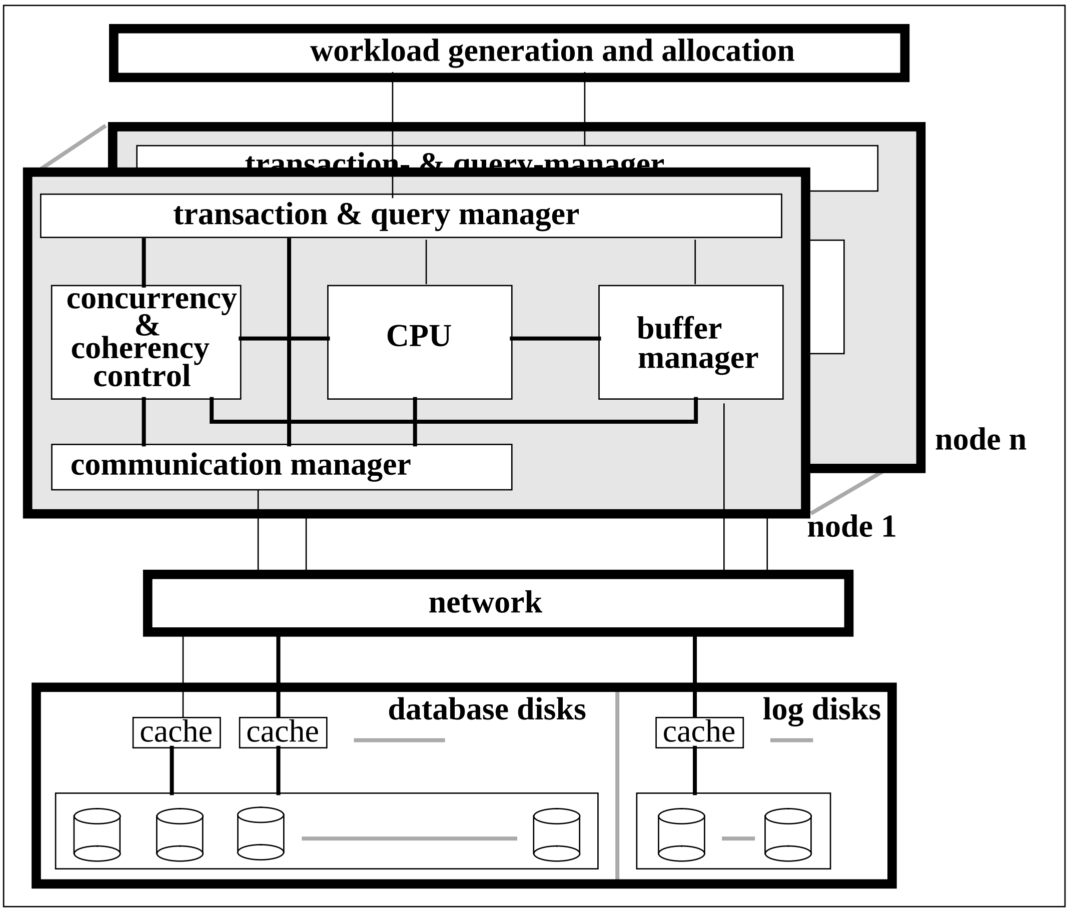

workload generation and allocation
transaction- & query-manager
transaction & query manager
concurrency & coherency control
CPU
buffer manager
communication manager
node n
node 1
network
database disks
log disks
cache
cache
cache

Fig. 2: Structure of the simulation system

Database and workload model

The database is modeled as a set of partitions. A partition may be used to represent a relation, a relation fragment or an index structure. It consists of a number of database pages which in turn consist of a specific number of objects (tuples, index entries). The number of objects per page is determined by a blocking factor which can be specified on a per-partition basis. Each relation can have associated clustered or non-clustered B^+-tree indices. Relations and indices can be physically declussets. Finally, a correct and efficient concurrency/coherency control approach is difficult to support with such a two-layered query processing scheme*.

We will therefore assume an integrated approach in this paper where the DBMS instances fully support parallel query processing.

* DB2 Parallel Query Server is initially restricted to read-only queries and operates on an asynchronously updated database copy so that no concurrency/coherency control problems exist.

Intra-query parallelism may be implemented by an additional layer on top of a conventional SD DBMS. This layer is responsible for decomposing a query into several subqueries that are submitted to and independently processed by different DBMS instances on multiple nodes. The extra layer also has to merge the results of the individual subqueries and to perform some post-processing if necessary, e.g., for computation of aggregates, etc..

For instance, assume the following SQL query on relation R:

```
SELECT B, MAX (C)
FROM R
WHERE C > 30.000
GROUP BY B
```

If R is declustered on attribute A as above, this query could be decomposed in up to 100 subqueries with the i-th subquery being

```
SELECT B, MAX (C)
FROM R
WHERE C > 30.000 AND A >= :min-i AND A <= :max-i
GROUP BY B
```

In this query *min-i* and *max-i* represent the lower and upper bound for A in the i-th partition as defined by the range partitioning. By extending the WHERE clause according to the range partitioning, it is guaranteed that each subquery works on disjoint set of disks so that disk contention is avoided if the subqueries are processed by a relation (partition) scan. Some post-processing is required to determine the global maximum per group from the subqueries' local maxima.

Such an approach has been implemented on top of Oracle's SD system for the KSR1 system [RMW93] and is also adopted in the DB2-based S/390 Parallel Query Server. It is relatively easy to implement since little changes are necessary in the underlying DBMS. However, performance problems may occur if the local DBMS instances optimize and process the individual subqueries independently from the decomposition layer responsible for intra-query parallelism. For example, if the local DBMS decide for the above query to use an index scan on attribute C rather than a relation scan, the chosen decomposition may suffer from disk contention on the index disk(s). For more complex queries, it is also conceivable that the local DBMS generate different execution plans for their subqueries so that execution skew may be introduced. Furthermore, since the query results are managed outside the DBMS (merging, sorting, etc.), a high overhead can be introduced for large result clustered index for sequentially processed index scans without problems. Sequential index scans incur minimal communication overhead and are therefore optimal for very selective queries (e.g., exact match queries on unique attribute). However, there may be index scans (e.g., for range queries) that need intra-query parallelism to achieve sufficiently short response times. With a physical declustering, this entails the danger that subqueries may have to access the same disks thereby causing disk contention. Concurrent access to higher-level index pages (root page and second-level pages) is expected to be less problematic since these pages can be cached in main memory or the disk caches. However, disk contention can arise for access to different index leaf pages and data pages stored on the same disk. The impact of disk contention for data pages is also expected to depend on whether a clustered or non-clustered index is being used. Our performance analysis will study these aspects in more detail.

Logical partitioning has the advantage that the DBMS knows the value distribution on disk for the partitioning attribute A. This is useful to restrict scan queries on A to a subset of the disks even without using an index. Furthermore, queries on A can easily be parallelized according to the partitioning function without introducing disk contention. For example, assume that the following range partitioning on A is used for allocating a relation to 100 disks:

A: (1 - 10,000; 10,001 - 20,000; 20,001 - 30,000; ...; 990,001 - 1,000,000).

A range query requesting tuples with A values between 70,000 and 220,000 can be processed by 15 (5, 3, 1) parallel subqueries each accessing 1 (3, 5, 15) of the 100 disks. If there is an index for A, the index scan can similarly be parallelized into 1-15 subqueries. To avoid contention for the index, it could also be partitioned into D subindices similarly as in SN systems[*].

Scan queries on different attributes than A cannot take advantage of the logical partitioning. They are similarly processed than with a physical declustering. Hence, parallel index scans for such queries may also suffer from disk contention between subqueries. A general disadvantage of logical partitioning is that it is difficult to define for the database administrator (DBA), in particular for range partitioning. A physical declustering, on the other hand, may only require specification of the degree of declustering D.

[*] Note however that the increased flexibility of the SD architecture regarding scan parallelism and selection of scan processors is preserved.

2 Parallel scan processing

To support parallel query processing, we assume that relations and index structures (B+ trees) can be declustered across several disks according to a physical or logical partitioning strategy. Physical partitioning operates on physical distribution granules like blocks or block sets and can be implemented outside the DBMS, e.g., within a disk array [PGK88]. Such an approach supports I/O parallelism for large read operations, but can cause performance problems in combination with processing parallelism. This is because if the DBMS has no information on the physical data allocation (declustering) it may not be possible to split a query into parallel subqueries so that these subqueries do not access the same disks. Logical partitioning, on the other hand, uses logical database objects like tuples as distribution granules and is typically defined by a partitioning function (e.g., range or hash) on a partitioning attribute (e.g., primary key). DB2 permits a logical range partitioning of relations across several disks, while Oracle supports physical declustering and hash partitioning. Typically, the database allocation in SN systems is also based on a logical range or hash partitioning.

To make *physical declustering* useful for parallel query processing in SD systems we assume that the DBMS at least knows the degree of declustering D and the disks holding partitions for a particular relation. These prerequisites make it easy to support parallel processing of relation scans without disk contention between subqueries. For a degree of declustering D this is possible for different degrees of parallelism P by choosing P such that

$$P * k = D,$$

where k is the number of disks to be processed per subquery. For instance, if we have D=100 we may process a relation scan with P = 1, 2, 4, 5, 10, 20, 25, 50 or 100 subqueries without disk contention between subqueries. Furthermore, each subquery processes the same number of disks (k) so that data skew can largely be avoided for equally sized partitions. CPU contention between subqueries is also avoided if each subquery is assigned to a different processor which is feasible as long as P does not exceed the number of processors $n*m$. The degree of declustering D should at least be high enough to support sufficiently short response time for a relation scan in single-user mode. As we will see, multi-user mode may require to have higher degrees of declustering, or degrees of scan parallelism P smaller than D.

A physical declustering of index structures is useful to support high I/O rates and thus inter-query/transaction parallelism (multi-user mode) [SL91]. SD can use a de-

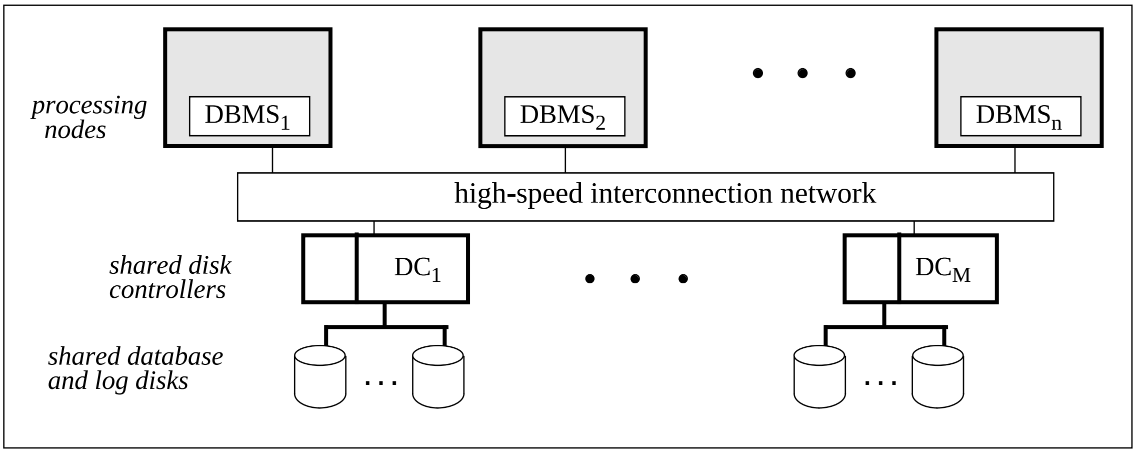


Figure 1: Shared Disk architecture

The nodes are assumed to be locally "clustered", i.e., they reside in one machine room. Furthermore, each node can access all disks as required for Shared Disk systems. All messages including I/O requests and data pages are exchanged across a high-speed and scalable interconnection network (e.g., hypercube). The main memory of each disk controller is used as a shared disk cache (DC). Each processing node runs private copies of the SD DBMS, operating system, and application software. Of course, the DBMSs' support the extensions needed for SD, in particular a global concurrency and coherency control protocol. Furthermore, parallel processing of scan queries is supported.

The remainder of this paper is organized as follows. The next section briefly discusses different alternatives for data allocation and parallel scan processing for SD. Section 3 provides an overview of the simulation model and the implemented approaches for concurrency/coherency control. In Section 4 we present and analyze simulation experiments for various system and workload configurations to study the impact of disk contention for the different scan query types. In particular, we analyze single-user as well as multi-user experiments with homogeneous and heterogeneous (query/OLTP) workloads. The major findings of this investigation are summarized in Section 5.

minimize the communication overhead and to support high transaction rates. For complex queries, on the other hand, a parallel processing on multiple nodes can be performed to achieve short response times. For these queries, we have the flexibility to base the degree of scan parallelism not only on parameters like relation size or query type, but also on the current system utilization. In particular, it may be advisable to choose a smaller degree of scan parallelism under high load in order to limit the communication overhead and the number of concurrent subqueries. Furthermore, complex queries can be assigned to less loaded nodes to achieve dynamic load balancing. In addition, it may be useful to assign OLTP transactions and complex queries to disjoint sets of nodes in order to minimize CPU and memory contention between these workload types.

However, SD bears the potential problem of disk contention that may outweigh the expected benefits discussed so far. Disk contention can already be introduced in single-user mode if concurrent subqueries of the same query are accessing the same disks. This problem can particularly be pronounced for parallel index scans because it may not be possible to prevent that multiple subqueries access index and data pages on the same disks. Hence, it is unclear to what degree it makes sense employing parallel index scans for SD[*]. The disk contention problem is aggravated in multi-user mode when multiple independent queries/transactions are accessing the shared disks. Note however, that disk contention in multi-user mode is not a SD-specific problem but is very difficult to deal with for SN as well [RM94].

To investigate the impact of disk contention on parallel query processing in more detail, we have implemented a detailed simulation system of a parallel SD database system. This model is used to study the relationship between the degree of declustering and the degree of processing parallelism for scan processing. The analysis is made for the three major types of scan queries: relation scan (table scan), clustered index scan, and non-clustered index scan. Furthermore, we study the usefulness of disk caches and prefetching for limiting disk contention. Finally, we show the usefulness to control disk contention in multi-user mode by dynamically choosing the degree of scan parallelism according to the current disk utilization (which is not feasible for SN).

Fig. 1 shows the SD architecture assumed in this paper. There are *n* processing nodes each consisting of *m* CPUs and local main memory. The processing nodes are loosely coupled, i.e., they communicate by message passing across a network.

[*] Note that Oracle 7.1 only supports parallel relation scans.

Yu87, MN91]. On the other hand, SN requires communication for distributed query processing, commit processing and global deadlock detection.

The differences between SN and SD with respect to the database allocation have far-reaching consequences for parallel query processing [Ra93b]. This is particularly the case for scan operations that operate on base relations*. In SN systems, a scan operation on relation R typically has to be processed by all nodes to which a partition of R has been assigned**. Hence, the degree of scan parallelism and thus the associated communication overhead are already determined by the largely static database allocation. Furthermore, there is no choice of which nodes should process a scan operation. As a result, SN does not support dynamic load balancing for scan, the most significant relational operator. SD, on the other hand, permits us to dynamically choose the degree of scan parallelism as well as the scan processors since each processor can access the entire relation R. Of course, R must be declustered across multiple disks to support I/O parallelism. In contrast to SN however, SD offers the flexibility to choose a degree of processing parallelism different from the degree of I/O parallelism.

This flexibility of the SD architecture is already significant for parallel query processing in single-user mode. This is because different scan operations on R have their response time minimum for different degrees of parallelism. For instance, a selective index scan accessing only one tuple is best processed on a single processor, while a relation scan accessing all tuples may require 100 processors to provide sufficiently short response times. SN requires to statically choose the degree of declustering and thus the degree of scan parallelism for an average load profile [Gh90]. If both scan queries of our example are processed with equal probability, the relation would thus have to be partitioned among 50 nodes resulting in sub-optimal performance for both query types (enormous communication overhead for the index scan relative to the actual work; sub-optimal degree of parallelism for the relation scan). SD, on the other hand, allows both query types to be processed by the optimal number of nodes (1 for the index scan, 100 for the relation scan), provided the relation is declustered across 100 disks.

The increased flexibility for parallel scan processing of SD is even more valuable in multi-user mode, in particular for mixed OLTP/query workloads [Ra93b]. So, OLTP transactions can always be processed sequentially on a single processing node to

* Operations on derived data, e.g. join, can be parallelized similarly in both architectures by dynamically redistributing the operations' input data among processors.

** Selections on the partitioning attribute, used to define the relation's partitioning, may be restricted to a subset of the data processors.

1 Introduction

Parallel database systems are the key to high performance transaction and database processing [DG92, Va93]. These systems utilize the capacity of multiple locally distributed (clustered) processing nodes interconnected by a high-speed network. Typically, fast and inexpensive microprocessors are used as processors to achieve high cost-effectiveness compared to mainframe-based configurations. Parallel database systems aim at providing both high throughput for on-line transaction processing (OLTP) as well as short response times for complex ad-hoc queries. This requires both inter- as well as intra-transaction parallelism. Inter-transaction parallelism (multi-user mode) is required to achieve high OLTP throughput and sufficient cost-effectiveness. Intra-transaction parallelism is a prerequisite for reducing the response time of complex and data-intensive transactions (queries).

Research on parallel database systems has so far focussed on "Shared Everything" (SE) or "Shared Nothing" (SN) architectures. By contrast, there is a growing number of commercially available DBMS supporting the "Shared Disk" (SD) alternative (Oracle, Rdb, IMS, DB2/MVS, Ingres, etc.), although most of them are currently restricted to inter-transaction parallelism. Presumably, Oracle’s "Parallel Server" represents the best-known SD implementation because it has achieved the highest transaction rates in TPC benchmarks. Furthermore, it is available for a variety of platforms including a growing number of “cluster” architectures (VaxCluster, Sequent, Pyramid, Encore, etc.) and massively parallel systems like nCube and KSR1. The new Oracle version 7.1 offers initial support for intra-query parallelism [Li93]. The forth-coming DB2-based S/390 Parallel Query Server of IBM will also provide intra-query parallelism.

Since SE is limited to relatively few processors, SN and SD are generally considered the most important approaches for parallel database systems [Pi90, DG92]. Both architectures consist of multiple loosely coupled processing nodes connected by a high-speed network. The software architecture is homogeneous in that each node runs an identical copy of the DBMS software. Through cooperation between these DBMS instances, complete distribution transparency (single system image) is achieved for database users and application programs. SN is based on a physical partitioning of the database among nodes, while SD allows each DBMS instance to access all disks and thus the entire database. The latter approach therefore requires a global concurrency control protocol (introducing communication overhead and delays) to achieve serializability. Furthermore, buffer coherency must be maintained since database pages may be replicated in multiple DBMS buffers [Ra86,

Abstract

Shared Disk database systems offer a high flexibility for parallel transaction and query processing. This is because each node can process any transaction, query or subquery because it has access to the entire database. Compared to Shared Nothing, this is particularly advantageous for scan queries for which the degree of intra-query parallelism as well as the scan processors themselves can dynamically be chosen. On the other hand, there is the danger of disk contention between subqueries, in particular for index scans. We present a detailed simulation study to analyze the effectiveness of parallel scan processing in Shared Disk database systems. In particular, we investigate the relationship between the degree of declustering and the degree of scan parallelism for relation scans, clustered index scans, and non-clustered index scans. Furthermore, we study the usefulness of disk caches and prefetching for limiting disk contention. Finally, we show the importance of dynamically choosing the degree of scan parallelism to control disk contention in multi-user mode.

Keywords: Parallel Database Systems; Shared Disk; Query Processing; Disk Contention; Multi-user Mode; Dynamic Load Balancing; Performance Analysis

Controlling Disk Contention for Parallel Query Processing in Shared Disk Database Systems

Erhard Rahm
Thomas Stöhr

Report No. 1

June 1994

University of Leipzig
Institute of Computer Science
Augustusplatz 10-11
04109 Leipzig (Germany)

E-mail: rahm@informatik.uni-leipzig.de

UNIVERSITÄT LEIPZIG

Institut für Informatik

Controlling Disk Contention for Parallel Query Processing in Shared Disk Database Systems

Erhard Rahm

Thomas Stöhr

Report Nr. 1 (1994)