# Dynamic Load Balancing in Parallel Database Systems

*Erhard Rahm*

University of Leipzig, Institute of Computer Science
E-mail: rahm@informatik.uni-leipzig.de

## Abstract

Dynamic load balancing is a prerequisite for effectively utilizing large parallel database systems. Load balancing at different levels is required in particular for assigning transactions and queries as well as subqueries to nodes. Special problems are posed by the need to support both inter-transaction/query as well as intra-transaction/query parallelism due to conflicting performance requirements. We compare the major architectures for parallel database systems, Shared Nothing and Shared Disk, with respect to their load balancing potential. For this purpose, we focus on parallel scan and join processing in multi-user mode. It turns out that both the degree of query parallelism as well as the processor allocation should be determined in a coordinated way and based on the current utilization of critical resource types, in particular CPU and memory.

## 1 Introduction

A significant trend in the commercial database field is the increasing support for parallel database processing [DG92, Va93]. This trend is both technology-driven and application-driven. Technology supports large amounts of inexpensive processing capacity by providing "super servers" [Gr95] consisting of tens to hundreds of fast standard microprocessors interconnected by a scalable high-speed interconnection network. The aggregate memory is in the order of tens to hundreds of gigabytes, while databases of multiple terabytes are kept online within a parallel disk subsystem. New application areas requiring parallel database systems for processing massive amounts of data and complex queries include data mining and warehousing, digital libraries, new multimedia services like video on demand, geographic information systems, etc.. Even traditional DBMS applications increasingly face the need of parallel query processing due to growing database sizes and query complexity [MPTW94]. In addition, high transaction rates must be supported for standard OLTP applications.

The effective use of super-servers for database processing poses many implementation challenges that are largely unsolved in current products [Se93, Gr95]. One key problem is the effective use of intra-query parallelism in multi-user mode, i.e., when complex queries are executed concurrently with other complex queries and OLTP transactions. Multi-user mode (inter-query/inter-transaction parallelism) is mandatory to achieve acceptable throughput and cost-effectiveness, in particular for super-servers where a high number of processors must effectively be utilized. While proposed algorithms for parallel query processing also work in multi-user mode, their performance may be substantially lower than in single-user mode. This is because multi-user mode inevitably leads

to data and resource contention that can significantly limit the attainable response time improvements due to intra-query parallelism.

Data contention problems may be solved by a multiversion concurrency control scheme which guarantees that read-only queries do not suffer from or cause any lock conflicts [CM86, BC92]. Increased resource contention, on the other hand, is unavoidable since complex queries pose high CPU, memory and disk bandwidth requirements which can result in significant delays for concurrently executing transactions. Furthermore, resource contention can be aggravated by the communication overhead associated with parallel query processing. In order to limit and control resource contention in multi-user mode, dynamic strategies for resource allocation and load balancing become necessary. In particular, the workload must be allocated among the processing nodes such that the capacity of different processing nodes be evenly utilized.

We first discuss the major forms of workload allocation and dynamic load balancing for database processing. Section 3 introduces the major architectures for parallel database processing, in particular Shared Nothing and Shared Disk systems. Their potential for dynamic load balancing is then evaluated for parallel relational database processing, in particular with respect to the two most important operators: scan (Section 4) and join (Section 5). In Section 6 we discuss additional considerations for supporting mixed OLTP/query workloads, in particular transaction routing.

## 2 Workload allocation

The general term "workload allocation" refers to the assignment of workload requests (processing steps) to physical or logical resources (processors, processes, memory, etc.). In this sense it corresponds to the term "resource allocation" which only expresses another perspective of the allocation problem. Depending on the workload or resource type special allocation problems can be considered, e.g., transaction and query allocation or processor and memory allocation. *Load balancing* refers to workload allocation in distributed systems where workload requests must be distributed among several processing nodes.

Heterogeneous database workloads consisting of OLTP transactions of different types as well as complex decision support queries pose special resource management problems even in the central case. One problem is to find a memory allocation that avoids that large queries monopolize the available buffer space thus causing unacceptable hit ratios for concurrent OLTP transactions. This problem can be addressed by giving higher priority to OLTP transactions and by using disjoint buffer areas for OLTP and large queries where the relative buffer sizes are dynamically controlled depending on the current workload. Such schemes have been proposed in [ZG90, PCL93, DG94] with respect to hash join queries. In [MD93, BMCL94], heuristics for dynamically controlling the number of concurrent queries are proposed in order to limit memory contention. Some commercial DBMS already support such dynamic memory allocation schemes, e.g., Tandem NonStop SQL and Informix.

For parallel database processing, load balancing is the major resource allocation problem in order to effectively utilize all available resources. Load balancing can be applied for different workload granularities depending on the level of parallelism. At the highest level, we have inter-transaction and *inter-query parallelism* with a concurrent execution of independent transactions and queries (multi-user mode). The corresponding load balancing is concerned with distributing transactions and queries among process-

ing nodes (transaction and query routing). *Intra-query parallelism* requires additional forms of load balancing for assigning subqueries to nodes. Several forms of intra-query parallelism can be distinguished in this context, namely inter-operator and intra-operator as well as pipeline and data parallelism [DG92]. Correspondingly, load balancing is necessary for operators (e.g., scan, join, sort) and sub-operators. In all cases, load balancing should be dynamic, that is the assignment decisions should be based on the current system utilization at runtime. Otherwise an even utilization of all nodes cannot be achieved due to typically high variations in the load composition (load surges, etc.) and system state.

*Pipeline parallelism* is typically used for inter-operator parallelism in order to overlap the execution of several operators within a query. Data parallelism, on the other hand, is applicable for both inter- and intra-operator parallelism and requires a data partitioning so that different (sub) operators can concurrently process disjoint sets of data. While both data and pipeline parallelism are needed, pipeline parallelism is generally considered less effective for reducing query response times [DG92]. This is because typically only comparatively few (<= 10) operators can be used within a pipeline because the total number of operators is mostly small and because there are blocking operators like sort that require the total input data before they can produce their output.

Load balancing is difficult to achieve for pipeline parallelism due to precedence dependencies between operators and because individual operators can substantially vary in their resource requirements and execution times. Furthermore, the size of temporary results cannot be predicted very well [Gr93]. A careful flow control must be exercised at runtime in order to avoid that the input data generated by producer operators cannot be processed fast enough by consumer operators. Otherwise, the consumers' input data would have to be stored within temporary files introducing a potentially high amount of disk I/O.

For these reasons our further discussions on load balancing will concentrate on data parallelism allowing much higher degrees of intra-query parallelism than pipeline parallelism. In order to support both high throughput for OLTP and short response times for complex queries it is important to dynamically determine the degree of intra-query parallelism as well as which processing nodes should process the subqueries. As we will see, the implementation of such an approach largely depends on the respective architecture and query type.

# 3 Architecture of Parallel Database Systems

Parallel database systems are typically based on multiple standard microprocessors interconnected by a local high-speed network. Effective support for inter- and intra-transaction parallelism requires both adequate use of I/O parallelism and processing parallelism. I/O parallelism must be supported by an allocation of the database across multiple disks (declustering), either within conventional disk farms or disk arrays [PGK88, CLG94]. Declustering supports intra-query parallelism by reading and writing large amounts of data processed by a single query in parallel from or to multiple disks. Inter-transaction parallelism is supported because independent I/O requests on different disks can be served in parallel.

With respect to processing parallelism, there are three major architectures for parallel database systems [DG92, Va93]:

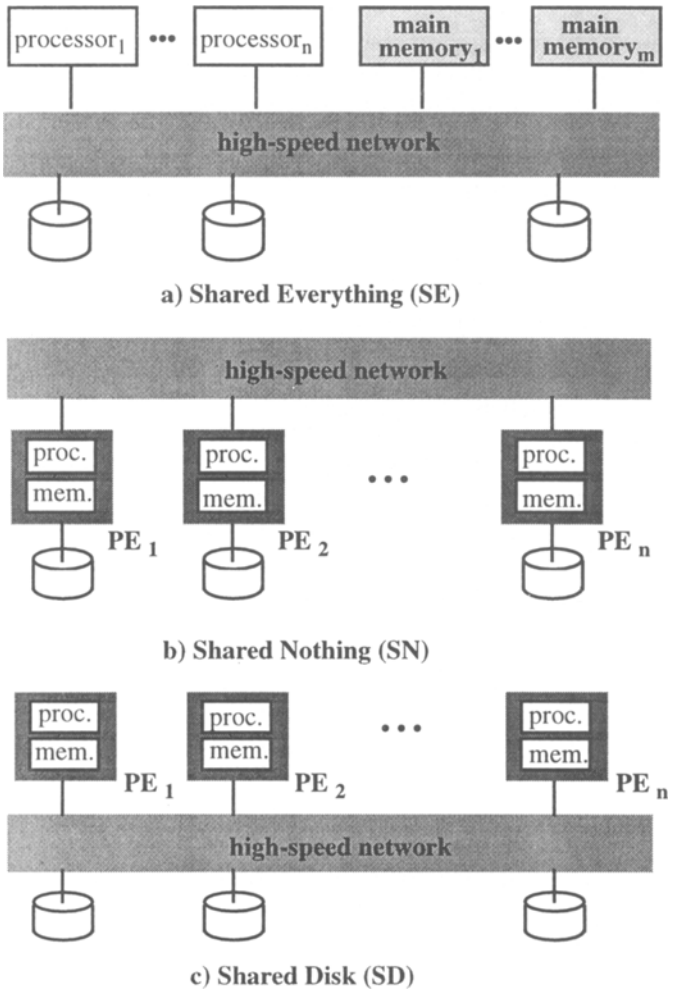


Figure 1: Architecture of parallel database systems

- *Shared Everything* (SE, Fig. 1a) refers to the use of multiprocessors for database processing. In this case, we have a tightly coupled system where all processors share a common main memory as well as peripheral devices (terminals, disks). There is only a single copy of the DBMS code that can be executed in multiple processes to utilize all processors. This approach is also referred to as Symmetric Multiprocessing (SMP).
- *Shared Nothing (SN,* Fig. 1b) systems consist of multiple autonomous processing elements (PE) each owning a private main memory and running separate copies of the operating system, DBMS and other software. Inter-processor communication takes place by means of message passing (loose coupling). A PE can consist of 1 or more processors, i.e., each node in a SN system may be a multiprocessor. The database is partitioned among the PEs so that each DBMS instance can di-

rectly access only data from the local partition. Access to non-local data requires a distributed query and transaction execution.

- Similar to SN, *Shared Disk (SD,* Fig. 1c) systems consist of multiple loosely coupled PE. However, the database is not partitioned but shared among the PE so that each DBMS instance has direct access to any data object. This assumes that each node can access any disk.

All three architectures are supported by commercial DBMS for both inter- and intra-transaction parallelism. Virtually all commercial DBMS are able to utilize multiprocessors (SE systems) for inter-transaction parallelism; support for intra-query processing is being added to most DBMS. Well-known SN systems supporting intra-query parallelism include Tandem NonStop SQL and ATT/Teradata's database machine; newer implementations are Sybase MPP, DB2/6000 Parallel Edition and Informix XPS (eXtended Parallel Server). Parallel SD implementations include Oracle Parallel Server and IBM's database systems (IMS, DB2) for parallel sysplex configurations. Oracle Parallel Server is available on many platforms, in particular on parallel computers (e.g., nCUBE) as well as on most "cluster" architectures (VaxCluster, Sequent, Pyramid, Encore, Sun, etc.).

SE systems have the advantage that shared memory supports efficient cooperation and synchronization between DBMS processes. Furthermore, effective load balancing is supported by the operating system that automatically assigns the next ready process/subquery to the next free CPU. These advantages are especially valuable for parallel query processing leading to increased communication and load balancing requirements to start/terminate and coordinate multiple subqueries. Furthermore, large intermediate results can efficiently be exchanged between subqueries. Several studies addressed dynamic load balancing for parallel query processing in SE systems [HSIT91, Om91, Ho92, LT92].

On the other hand, there are significant availability problems since the shared memory reduces failure isolation between processors, and since there is only a single copy of system software like the operating system or the DBMS [Ki84]. Furthermore, scalability is limited because the shared memory can introduce performance bottlenecks. Consequently, the number of processors is quite low in current SE systems ($\leq 30$). Due to these problems, SN and SD are generally considered as more appropriate to meet high-performance and high-availability requirements [DG92, MPTW94].

From a hardware point of view, SN systems appear particularly attractive. They allocate each disk drive to one particular PE and interconnect all PE by a local network, which is feasible with standard hardware at little cost. Furthermore, a large number of PE can be interconnected in this way because there are no shared resources (other than the network). SD systems, on the other hand, require an interconnection between all PE and disk drives (Fig. 1c). Because all I/O requests (page transfers) have to go over this network, an extremely fast and scalable (multi-stage) interconnection network is needed that may be much more expensive than the network of SN systems. Furthermore, SD may face an increased potential of performance bottlenecks in the network and disk subsystem.

On the other hand, with current fiber-optic interconnection technology it appears unlikely that high bandwidth requirements pose a major problem. Furthermore, high-performance SN systems also need a very fast and scalable network in particular for a larger number of PE and for parallel processing of complex queries that often requires

a dynamic redistribution of large amounts of data. In addition, even in SN systems it is typically necessary to interconnect each disk drive to at least two PE for fault tolerance reasons thus increasing hardware cost.

Hence, we conclude that hardware-related aspects are less significant when comparing SN and SD than software-related aspects, in particular with respect to database processing[1]. This is also because providing powerful hardware, e.g. large numbers of PE, does by no means imply that this hardware can effectively be utilized for database processing. A key prerequisite for achieving this goal is dynamic load balancing. As we will see, the potential for such a load balancing differs significantly between SN and SD DBMS. Apart from the feasible approaches for parallel query processing and dynamic load balancing, SN and SD systems differ in additional areas like transaction management (global concurrency control and global logging for SD; distributed commit for SN) and the need for coherency control (for SD). Efficient solutions for these problems have been proposed [ÖV91, GR93, MN91, Ra93] but are beyond the scope of this paper.

# 4 Parallel scan processing

Scan is the simplest and most common relational operator. If predicate evaluation cannot be supported by an index, a complete *relation scan* is necessary where each tuple (record) of the relation must be read and processed. An *index scan* accesses tuples via an index (typically a B+ tree) and restricts processing to a subset of the tuples; in the extreme case, no tuple or only one tuple needs to be accessed (e.g., exact-match query on unique attribute). Parallel scan processing requires a declustering of the relation and index structures across several disks in order to allow for I/O parallelism.

We first analyze parallel scan processing for SN; SD scan processing is discussed afterwards.

## Shared Nothing

In SN systems, the database partitioning among PE implies a corresponding data allocation to disks because each disk is exclusively assigned to one PE[2]. Database partitioning is typically based on a horizontal (tuple-wise) declustering of relations defined by a hash or range function on a *partitioning attribute* (e.g., primary key) [DG92]. Indices are also partitioned with the relation so that each PE holds a (sub-)index for the local records. Parallelizing a scan operation is straight-forward and determined by the database allocation. For hash and range partitioning, exact-match queries on the partitioning attribute can be restricted to a single processor; range partitioning also allows restricting the number of nodes for range queries on the partitioning attribute. However, all other scan queries must be processed by all data processors, i.e. all PE holding a partition of the respective relation.

The performance of parallel scan processing is thus very dependent on the degree of declustering as it coincides with the degree of scan parallelism in many cases. To evaluate

1. This is also underlined by the fact that there are SN database systems (e.g., Teradata) running on SD hardware platforms and vice versa. The former case is easily feasible by not utilizing the accessibility of all disks during normal processing but restricting each DBMS/node to a subset of the disks. The latter case is used by Oracle on SN platforms like IBM SP2 and is made feasible by the operating system implementing a "virtual shared disk" environment where the distinction between local and remote I/O is transparent to the DBMS.

2. There may be multiple disks per PE.

the impact of different degrees of parallelism in both single-user and multi-user mode, we have performed several simulation experiments with a detailed simulator of a SN database system[3]. In Fig. 2, we show the average response times and speedup values of different scan queries on a relation of 1 million tuples in single-user mode. The degree of declustering and scan parallelism (P) is varied between 1 and 64.
Fig. 2 shows that parallel processing of a relation scan is very effective in single-user mode and that a linear speedup could be obtained (the sequential processing time of about 30 minutes is reduced by a factor 60 for 64 PE). Still, response time for 64 PE was higher than in the case of a selective index scan for which only 0.1% of the tuples qualify. This illustrates that the use of an index may be more effective than employing parallel processing in order to improve response time (of course, not all queries can be supported by an adequate index). Parallel processing of index scans also improves response time but to a much lower degree than for relation scans. This is because the number of records to be processed for selective index scans is much lower than for a relation scans. Furthermore, the actual work (number of records) per processor is reduced for growing degrees of parallelism while the overhead for starting and terminating the subqueries increases proportionally to P.

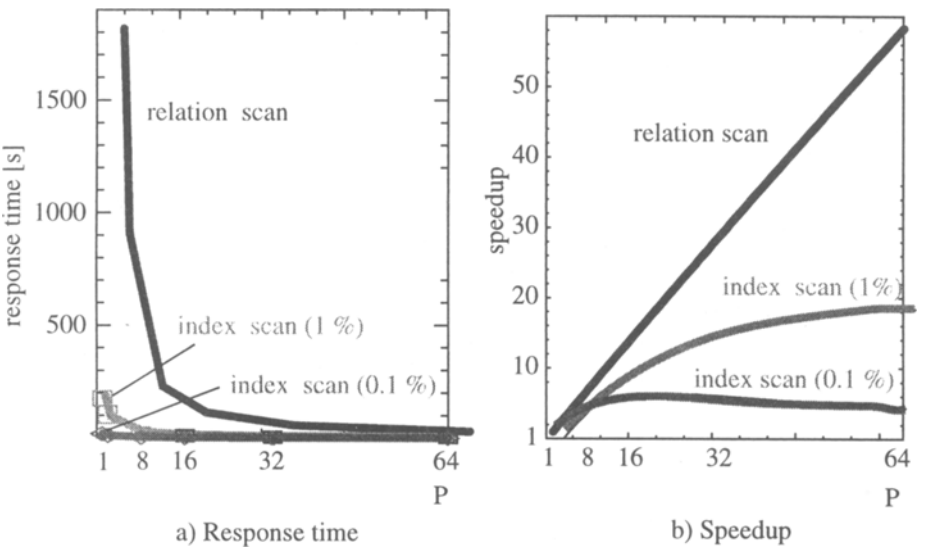


Figure 2: Parallel scan processing in SN systems (single-user mode)

A major implication of this observation is that different scan queries on the same relation have their response time minimum for different degrees of parallelism. As can be seen from Fig. 2b, a relation scan may be best processed by 64 PE while the index scans have their optimum for smaller degrees of parallelism. The optimal degree of scan parallelism may be computed by an analytical model that considers the tradeoffs between the actual amount of work (determined by factors like relation size, query selectivity, index usage etc.) and the overhead introduced by intra-query parallelism [Gh90, WFA92, Ma95]. Unfortunately, SN requires to statically choose the degree of declus-

3. Details on the simulation system can be found in [MR92, RM93].

tering so that there is no way to choose the degree of scan parallelism dependent on the scan type. As a result, the actual degree of declustering must be a compromise value for an average load profile, e.g., 30 for our example. However, this implies suboptimal performance for the different scan types, in particular a overly high communication overhead for index scans and an insufficient degree of parallelism for relation scans[4].

For scan processing in multi-user mode, we consider a homogeneous workload with multiple index scan queries. Because we want to linearly increase throughput with the number of PE we increase the arrival rate of our scan query proportionally with the system size. Fig. 3 shows the resulting response times for parallel scan processing and different arrival rates (in Queries Per Second per PE, QPS/PE). The main observation is that the optimal degree of scan parallelism depends on the arrival rate and thus on the current system (CPU) utilization; it becomes the lower the higher the system is utilized. This is because the communication overhead associated with a higher degree of parallelism is less affordable under high CPU utilization (high arrival rates).

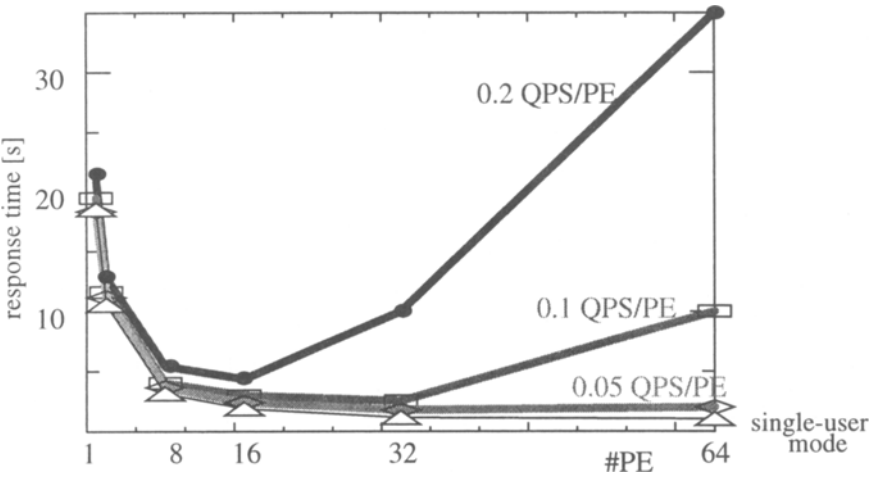


Figure 3: Parallel scan processing in SN systems (multi-user mode)

Hence, it would be desirable to choose the degree of scan parallelism according to the current system state. Unfortunately, this is not possible for SN as the data allocation statically determines the degree of scan parallelism. Furthermore, the scan processors themselves are also determined by the database allocation so that there is no support at all for dynamic load balancing.

### Shared Disk

In SD systems each node has access to the entire database on disk. Hence, scan operations on a relation can be performed by any number of nodes. This gives the required flexibility to choose the degree of scan parallelism according to the query type and the

4. Some SN systems (e.g. Teradata) simply use a *full declustering* so that each relation is partitioned across all PE. Such an approach is clearly too simple and introduces a high communication overhead, in particular for small relations and index scans [CABK88]. The high overhead is especially harmful in multi-user mode [RM93].

current system utilization. For example, index scans on any attribute may be performed by a single processor thereby minimizing communication overhead. This would especially be appropriate for exact-match and selective range queries, and supports high throughput. For relation scans, on the other hand, a high degree of processing parallelism can be employed to utilize intra-query parallelism to reduce response time. Furthermore, under high system utilization a smaller degree of scan parallelism can be chosen to limit the communication overhead and support high throughput. A high potential for dynamic load balancing is also supported by the fact that all PE are eligible for scan processing allowing a dynamic decision about where a scan should be processed. For instance, a scan may be allocated to a set of processors with low CPU utilization in order to avoid interference with concurrent transactions on other nodes.

The load balancing potential of the SD architecture must be supported by an adequate data allocation that avoids disk contention between parallel subqueries of the same query. While in SN systems the data is partitioned among PE, SD (and SE) only requires a data declustering across multiple disks. Such a data allocation merely prescribes the maximal degree of I/O parallelism while the degree of scan parallelism can be chosen smaller. Parallel processing of relation scans is easily supported by choosing a degree of declustering D that is high enough for providing sufficiently short response times in single-user mode. Parallel relation scans are possible without disk contention 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.

Selective index scans returning only a few records are best processed sequentially which is feasible for SD with minimal communication overhead. Parallel index scans, on the other hand, may lead to disk contention between subqueries on both index and actual data, even when the shared index is declustered across several disks. The performance study [RS95] showed that this problem primarily exists for clustered index scans which should therefore be processed sequentially (unless the selectivity is so high that the data of multiple disks needs to be accessed). Parallel non-clustered index scans, on the other hand, did not suffer from a significant disk contention in the case of larger degrees of declustering.

Multi-user mode typically not only leads to CPU but also to disk contention for both SN and SD architectures. In [RS95] it was found that SD is able to reduce the level of disk contention by using smaller degrees of scan parallelism under high disk utilization. SN does not provide such a flexibility.

# 5 Parallel join processing

Parallel (equi-)join processing typically consists of a parallel scan phase and a parallel join phase. During the scan phase, the scan processors read the input relations from disk and perform selections on them. The scan output is then redistributed among multiple join processors performing the join phase using any sequential algorithm (e.g., hash join or sort-merge). Finally, the local join results are merged at a designated node. Data redistribution between scan and join processors is performed by applying a partitioning function (hash or range) on the join attribute. This ensures that matching tuples of both

input relations arrive at the same join processor. The advantage of such a scheme is that there is a high potential for dynamic load balancing even for SN. This is because the number of join processors as well as the choice of these processors can be based on the current load situation, similarly as for parallel scan processing in SD systems. On the other hand, the communication overhead for data redistribution can be substantial.

In the following, we outline some general tradeoffs to consider for dynamic load balancing and such a parallel join processing. Afterwards, we briefly discuss load balancing in the presence of data skew as well as some other join strategies with reduced redistribution overhead.

### General tradeoffs

Similar to parallel scan processing, it is possible to determine the optimal degree of intra-query processing (i.e. the optimal number of join processors) in single-user mode by means of an analytical model [Ma95]. In addition to the mentioned tradeoff between actual (CPU) work per subquery and communication overhead there are additional factors that need to be considered for determining the optimal number of join processors. In particular, the overhead for redistributing the data between scan and join processors increases with the number of join processors. Furthermore, the I/O overhead for join processing is very much dependent on the aggregate memory of the join processors that increases with the degree of join parallelism. For instance, if a hash join is used for local join processing an optimal I/O performance is achieved if the smaller join input can be completely kept in the join processors' memory [Gr93]. If less memory is available, additional I/O is necessary to keep the join input in temporary disk files at the join processors leading to a substantial response time degradation. Hence, the optimal degree of join parallelism in single-user mode is at least as high as required to avoid temporary file I/O (or, if this is unachievable, the total number of PE). Since all PE are lightly loaded in single-user mode, selection of the join processors is no problem (e.g., random selection is sufficient).

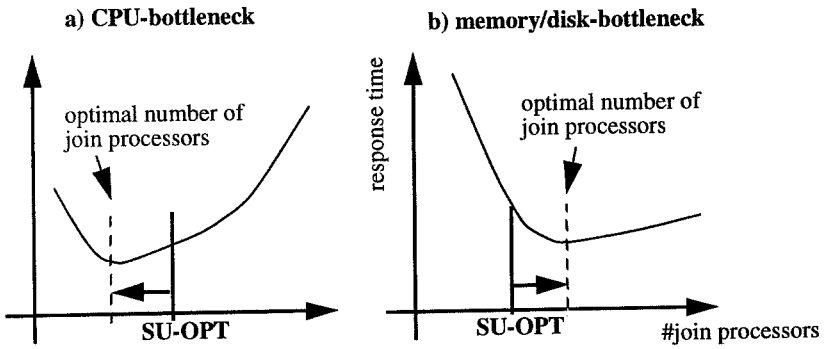


Figure 4: Optimal number of join processors in multi-user mode

The studies [RM93, RM95] showed however, that this changes significantly in multi-user mode. Similar to parallel scan processing, the optimal number of join processors is lower than in single-user mode under high CPU utilization (Fig. 4a). Moreover, the optimal degree of join parallelism is generally the lower the higher the system is utilized

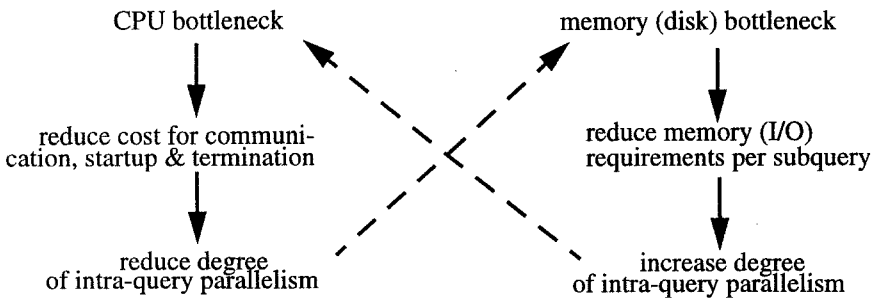


Figure 5: Tradeoffs in dynamic load balancing with multiple bottlenecks

due to the high communication overhead associated with many join processors. Furthermore, the least utilized CPUs should be selected for join processing [RM93]. However, if response times are largely dominated by temporary file I/O, i.e., if we have memory or disk bottlenecks, it is generally advisable to choose more join processors to obtain more memory and thus to reduce the amount of temporary I/O [RM95]. As a result, under high memory (disk) utilization the optimal degree of join parallelism is typically higher than in single-user mode (Fig. 4b). These tradeoffs are summarized in Fig. 5 which shows that the degree of join parallelism in multi-user mode must be chosen dynamically based on the current memory, disk and CPU utilization. Similarly, selection of the join processors must consider the current CPU and memory utilization of the PE.

## Policies

Specific approaches for such a dynamic load balancing considering multiple bottleneck types have been proposed and evaluated in [RM95]. A major finding is that the dynamic decisions should be based on the actual resource utilization at the individual nodes rather than on average utilization levels (since there may be large differences in resource utilization). Furthermore, it is important to use integrated policies for drawing the two control decisions (degree of parallelism, processor allocation) in a coordinated way. For example, a policy called MIN-IO-SUOPT proved to be effective in the presence of memory bottlenecks. Based on the current memory availability it determines the degree of join parallelism p so that the amount of temporary file I/O is minimized. In the case of multiple configurations avoiding temporary file I/O the number of processors closest to the single-user optimum is selected (in order to ensure a sufficiently high degree of parallelism). The subjoins are assigned to the p nodes with the most available memory.

In [MD95] an algorithm called RateMatch for dynamically determining the number of join processors is presented. This scheme is based on the observation that the size of the join input is less significant for finding the optimal number of join processors than the rate at which the scan processors generate the join input. Thus the scheme tries to determine the number of join processors such that their aggregate join processing rate matches the rate at which the join input is provided by the scan processors. However, RateMatch is an isolated scheme that uses an independent algorithm for selecting the join processors. Furthermore, it only considers average values for the current resource utilization rather than node-specific utilization information.

### Treatment of Data Skew

For data parallelism to be effective the underlying data partitioning must ensure about equally sized input partitions for subqueries. In general, this is difficult to achieve due to non-uniform value distributions for the partitioning attributes or because of nonuniform query selectivities. The resulting data skew can result in large differences in the execution times of subqueries thereby reducing the effectiveness of intra-query parallelism (since response times are determined by the slowest subqueries).

For parallel join processing based on a dynamic redistribution of the join input it is feasible to dynamically check the size of the scan output and to extend the redistribution scheme in order to generate about equally sized join inputs in the presence of data skew. Approaches for such a dynamic load balancing have been proposed in [WDJ91, DNSS92, HLY93, WDYT94, HLH95]. However, these studies assumed single-user mode corresponding to a best-case situation with little or no resource contention. Hence, only intra-query load balancing is supported and the effectiveness of the proposals in multi-user mode must be questioned. In fact, the overhead associated with the proposed load balancing schemes is likely to be a major problem in multi-user mode. A more appropriate approach to reduce the skew problem in multi-user mode may be to avoid the expensive generation of equally-sized subjoins but to use information on the current system utilization to select the join processors dependent on the size of the subjoins (by assigning larger subjoins to less loaded nodes, etc.).

### Limiting the redistribution overhead

Dynamically redistributing both input relations supports dynamic load balancing, but incurs a high communication overhead for large relations. In SN systems, this overhead is reduced if at least one of the join inputs is already declustered on the join attribute. In this case, the data processors of this relation act as join processors and only the second relation needs to be redistributed. Data redistribution is completely avoided if both relations are equally partitioned on the join attributes and allocated to the same set of PE. Unfortunately, these special cases leave no room for dynamic load balancing because the degree of join parallelism and the join processors are statically determined by the database allocation, similar as for scan processing. Still, in many cases the communication savings are more significant than the lost load balancing potential [RM93].

SD systems can also avoid the redistribution overhead if one or both join inputs are physically declustered across several disks by a logical partitioning on the join attribute. In this case, however, a large potential for dynamic load balancing remains because each PE may be selected as join processor. Furthermore, the declustering still leaves several choices for the degree of join parallelism without causing disk contention between subqueries (similar to parallel scan processing). For instance assume two relations declustered across 50 disks by using the same value partitioning (range or hash) on the join attributes. Then there may be 1, 2, 5, 10, 25 or 50 scan and join processors working in parallel and accessing disjoint sets of disks. In this case, the redistribution overhead is completely avoided while a high potential for dynamic load balancing is preserved.

## 6 Mixed workloads

In the previous sections we have already shown the need for dynamic load balancing for parallel query processing in multi-user mode, i.e. with both intra-query and inter-query parallelism. Similar requirements are posed for mixed workloads consisting of

simple OLTP transactions and complex queries. Since efficient processing of OLTP transactions has typically highest priority, the need for limiting resource contention due to parallel query processing is aggravated. As we have seen SD offers significant advantages over SN to achieve this goal:

- SN requires definition of a (static) database allocation for an "average" load profile that must be a compromise between the different requirements for OLTP and complex queries. This inevitably leads to sub-optimal performance for both workload types and does not support dynamic load balancing. In particular, complex queries have to be restricted to fewer nodes than desirable to limit the communication overhead so that response times may not sufficiently be reduced. On the other hand, OLTP transactions cannot be confined to a single node in many cases thereby causing extra communication overhead and lowering throughput. In both cases, the sub-optimal performance must be accepted even if only one of the two workload types is temporarily active.
- In SD systems, declustering of data across multiple disks does not increase the communication overhead for OLTP. In general, OLTP transactions are completely executed on one node to avoid the communication overhead for intra-transaction parallelism and distributed commit. On the other hand, the degree of processing parallelism and thus the communication overhead for complex queries can be adapted to the current load situation. Furthermore, resource contention for CPU and memory between OLTP transactions and complex queries may largely be avoided by assigning these workload types to disjoint sets of processors which is not possible for SN, in general.

SD also offers an increased flexibility for ***transaction routing***, i.e. for determining the PE where a transaction request should be routed to. This decision is particularly important for OLTP transactions where it can largely influence the number of remote requests. In SN systems, the best load assignment is primarily determined by the static database allocation. This is because a transaction should be assigned to the PE where most of the needed data is locally available. Assigning the transaction to another PE would still require that the data-owning PE has to process the operations on the required data. Hence, only little work would be saved for this node but additional communication overhead would be introduced for starting the subqueries, returning the results and for commit processing.

In SD systems, on the other hand, each PE can process any database operation and thus any transaction leaving a high potential for dynamic transaction routing and load balancing. However, sequentially executing a transaction on one PE can also require inter-PE communication for SD, in particular for global concurrency and coherency control [Ra93]. To limit the communication overhead for these functions it is generally advisable to support locality of reference by means of a so-called affinity-based transaction routing [YCDI87, Ra92, Ra93]. It assigns transactions with an affinity to the same database portions to the same processing nodes which supports a local concurrency and coherency control (depending on the chosen protocol) and good I/O performance. A survey of transaction routing schemes can be found in [Ra92].

# 7 Summary and Outlook

Dynamic load balancing is a prerequisite for effective utilization of parallel database systems consisting of many processing elements. Intra-query parallelism must effectively be supported in multi-user mode, i.e., in combination with inter-query and inter-transaction parallelism. The major control decisions to draw dynamically include determining the degree of intra-query parallelism and selecting the processors for executing subqueries. Load balancing must also be supported by a dynamic transaction routing for the initial assignment of queries and transactions.

Dynamic load balancing is most easily achieved for Shared Everything, but these systems suffer from availability and scalability limitations introduced by the shared memory. Shared Nothing and Shared Disk DBMS, on the other hand, require a comparatively large communication overhead for parallel query processing, in particular if intermediate query results are dynamically distributed in the system. We found out that Shared Nothing only offers a limited flexibility for dynamic load balancing because the database allocation determines in many cases where operations have to be processed, in particular for scan operations. Shared Disk systems, on the other hand, provide a higher load balancing potential since each PE can access any data and thus process any transaction, query or subquery which is especially valuable for query processing in multi-user mode and for mixed workloads. SD also requires an appropriate declustering of the data across multiple disks. But this data allocation only determines the maximal degree of intra-query parallelism, while the actual degree of parallelism can be chosen smaller (depending on the query type and current system state) without causing disk contention between subqueries.

Both SD and SN can employ dynamic load balancing for parallel join processing if the join inputs are dynamically redistributed among several join processors. We have discussed basic performance tradeoffs to consider for determining the optimal degree of join parallelism and selecting join processors in multi-user mode. Under high CPU utilization we found it necessary to reduce the degree of join parallelism in order to limit CPU contention (communication overhead for startup/termination and data redistribution). Under disk and memory bottlenecks, on the other hand, the degree of join parallelism should be increased in order to reduce the memory and I/O requirements per subquery. Furthermore, both control decisions should be drawn in a coordinated way and based on node-specific utilization information rather than system-wide averages. The redistribution overhead is reduced or avoided if the join input is already physically declustered on the join attribute. In this case, only SD preserves a potential for dynamic load balancing while for SN the data processors have to perform the join.

More work is needed on dynamic load balancing in several areas, in particular for inter-operator parallelism, to deal with data skew in multi-user mode, and to integrate local resource allocation policies for mixed workloads. Moreover, parallel database processing for object-oriented databases needs further investigation.

# 8 References


BC92 Bober, P.M., Carey, M.J.: On Mixing Queries and Transactions via Multiversion Locking. *Proc. 8th IEEE Data Engineering Conf.*, 535-545, 1992

BMCL94 Brown, K.P.; Mehta, M.; Carey, M.J.; Livny, M.: Towards Automated Performance Tuning for Complex Workloads. *Proc. 20th VLDB Conf.*, 72-84, 1994

CABK88 Copeland, G., Alexander, W., Boughter, E., Keller, T.: Data Placement in Bubba. *Proc. ACM SIGMOD Conf.*, 99-108, 1988

CLG94 Chen, P.M., Lee, E.K., Gibson, G.: RAID: High-Performance, Reliable Secondary Storage. *ACM Computing Surveys* 26 (2), 145-185, 1994

CM86 Carey, M.J., Muhanna, W.A.: The Performance of Multiversion Concurrency Control Algorithms. *ACM Trans. on Computer Systems* 4 (4), 338-378, 1986

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

DG94 Davison, D.L.; Graefe, G.: Memory-Contention Responsive Hash Joins. *Proc. 20th VLDB Conf.*, 379-390, 1994.

DG95 Davison, D.L.; Graefe, G.: Dynamic Resource Brokering for Multi-User Query Execution. *Proc. ACM SIGMOD Conf.*, 281-292, 1995

DNSS92 DeWitt, D.J., Naughton, J.F., Schneider, D.A., Seshadri, S.: Practical Skew Handling in Parallel Joins. *Proc. 18th VLDB Conf.*, 27-40, 1992

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

Gr93 Graefe, G.: Query Evaluation Techniques for Large Databases. *ACM Comput. Surveys* 25 (2), 73-170, 1993

Gr95 Gray, J.: Super-Servers: Commodity Computer Clusters Pose a Software Challenge. *Proc. German Database Conf. BTW*, March 1995

GR93 Gray, J., Reuter, A.: *Transaction Processing.* Morgan Kaufmann, 1993

Ho92 Hong, W.: Exploiting Inter-Operation Parallelism in XPRS. *Proc. ACM SIGMOD Conf.*, 19-28, 1992

HLH95 Hua, K.A., Lee, C.; Hua, C.M.: Dynamic Load Balancing in Multicomputer Database Systems Using Partition Tuning. *IEEE Trans. on Knowledge and Data Engineering* 7(6), 968-983, 1995

HLY93 Hua, K.A., Lo, Y., Young, H.C.: Considering Data Skew Factor in Multi-Way Join Query Optimization for Parallel Execution. *VLDB Journal* 2(3), 303-330, 1993

HSIT91 Hirano, Y., Satoh, T., Inoue, U., Teranaka, K.: Load Balancing Algorithms for Parallel Database Processing on Shared Memory Multiprocessors. *Proc. 1st Int. Conf. on Parallel and Distributed Information Systems*, 210-217, 1991

Ki84 Kim, W.: Highly Available Systems for Database Applications. *ACM Computing Surveys* 16 (1), 71-98, 1984

LT92 Lu, H., Tan, K.: Dynamic and Load-Balanced Task-Oriented Database Query Processing in Parallel Systems. *Proc. EDBT*, LNCS 580, 357-372, 1992

Ma95 Marek, R.: A Cost Model for Parallel Query Processing in Shared Nothing DBS (in German). *Proc. German Database Conf. BTW*, March 1995

MD93 Mehta, M., DeWitt, D.J.: Dynamic Memory Allocation for Multiple-Query Workloads. *Proc 19th VLDB Conf.*, 354-367, 1993

MD95 Mehta, M., DeWitt, D.J.: Managing Intra-Operator Parallelism in Parallel Database Systems. *Proc 21th VLDB Conf.*, 382-394, 1995

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

MPTW94 Mohan, C., Pirahesh, H., Tang, W.G., Wang, Y.: Parallelism in Relational Database Management Systems. *IBM Systems Journal 33 (2), 1994*

MR92 Marek, R., Rahm, E.: Performance Evaluation of Parallel Transaction Processing in Shared Nothing Database Systems, *Proc. 4th Int. PARLE Conf.*, LNCS 605, 295-310, 1992

Om91 Omiecinski, E.: Performance Analysis of a Load-Balancing Hash-Join Algorithm for a Shared-Memory Multiprocessor. *Proc 17th VLDB Conf.*, 375-385, 1991

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

PCL93 Pang, H., Carey, M.J., Livny, M.: Partially Preemptible Hash Joins. *Proc. ACM SIGMOD Conf.*, 59-68, 1993

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

Ra92 Rahm, E.: A Framework for Workload Allocation in Distributed Transaction Processing Systems. *Journal of Systems and Software* 18, 171-190, 1992

Ra93 Rahm, E.: Empirical Performance Evaluation of Concurrency and Coherency Control for Database Sharing Systems. *ACM Trans. on Database Systems* 18 (2), 333-377, 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

RM95 Rahm, E., Marek, R.: Dynamic Multi-Resource Load Balancing in Parallel Database Systems. *Proc 21th VLDB Conf.*, 395-406, 1995

RS95 Rahm, E., Stöhr, T.: Analysis of Parallel Scan Processing in Shared Disk Database Systems. *Proc. Euro-PAR95,* LNCS 966, 485-500, 1995

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

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

WDJ91 Walton, C.B; Dale A.G.; Jenevein, R.M.: A Taxonomy and Performance Model of Data Skew Effects in Parallel Joins. *Proc. 17th VLDB Conf.*, 537-548, 1991

WDYT94 Wolf, J.L., Dias, D.M., Yu, P.S., Turek, J.: New Algorithms for Parallelizing Relational Database Joins in the Presence of Data Skew. *IEEE Trans. on Knowledge and Data Engineering* 6(6), 990-997, 1994

WFA92 Wilschut, A.; Flokstra, J.; Apers, P.: Parallelism in a Main-Memory DBMS: The performance of PRISMA/DB. *Proc. 18th VLDB Conf.*, 521-532, 1992

YCDI87 Yu, P.S., Cornell, D.W., Dias, D.M., Iyer, B.R.: Analysis of Affinity-based Routing in Multi-system Data Sharing. *Performance Evaluation* 7 (2), 87-109, 1987

ZG90 Zeller, H., Gray, J.: An Adaptive Hash Join Algorithm for Multiuser Environments. *Proc. 16th VLDB Conf.*, 186-197, 1990