The VLDB Journal
© Springer-Verlag 1997

Data placement in shared-nothing parallel database systems*

Manish Mehta, David J. DeWitt

12, S. First St., San Jose, CA 95113, USA; e-mail: mmehta@64k.com
1210, W. Dayton St., Madison, WI 53706, USA; e-mail: dewitt@cs.wisc.edu

Edited by M. Adiba. Received May 11, 1993 / Accepted April 24, 1996

Abstract. Data placement in shared-nothing database systems has been studied extensively in the past and various placement algorithms have been proposed. However, there is no consensus on the most efficient data placement algorithm and placement is still performed manually by a database administrator with periodic reorganization to correct mistakes. This paper presents the first comprehensive simulation study of data placement issues in a shared-nothing system. The results show that current hardware technology trends have significantly changed the performance tradeoffs considered in past studies. A simplistic data placement strategy based on the new results is developed and shown to perform well for a variety of workloads.

Key words: Declustering – Disk allocation – Resource allocation – Resource scheduling

1 Introduction

The last decade has seen a significant change in the characteristics of database applications. The demands of traditional applications, like transaction processing, have grown dramatically. In addition, emerging applications like geographical information systems, multimedia, and database mining, pose new performance challenges to existing database systems. Shared-nothing (SN) database systems (Bubba [Bora90], Gamma [DeWi90], Tandem [Tand88], Teradata [Tera85], Volcano [Grae89]), with their promise of scalability and availability, have evolved as an answer to these new challenges. Existing configurations contain hundreds of processors, each with multiple disks. Efficient resource management is essential for high performance in such large systems.

An important resource management issue in SN parallel database systems is the layout of the database in the system or data placement. Several earlier studies have shown that the performance and scalability of a SN parallel database systems is contingent on the physical layout of data across the nodes in the system. Moreover, data placement also serves as an important load-balancing mechanism. In the absence of remote data-access in a SN system, data placement determines not only the distribution of data but also the distribution of operators (like select) that directly access the data. Therefore, a poor data placement strategy can result in a non-uniform distribution of the load and the formation of bottlenecks. The relatively static nature of data placement decisions also increases the need for an efficient placement algorithm. Other resources, like processors and memory can be re-allocated at run-time, allowing for the design of dynamic policies that can adapt to workload transients [Brow93, Meht93, Rahm93, Brow94]. However, data placement can be changed only by an expensive reorganization of the relations in the database. All these factors make data placement an extremely important issue in high-performance SN systems.

In order to exploit I/O parallelism in a parallel SN database system, tuples belonging to a single relation are typically placed on multiple disks. The relation is said to be horizontally partitioned or *declustered* [Ries78, Livn87] in such cases[1]. Therefore, the first issue that needs to be addressed in selecting a data placement strategy for a SN database system is the number of nodes on which to partition or decluster tuples of each relation, called the *degree of declustering* of the relation. Declustering exploits I/O parallelism but it also leads to higher startup and termination costs because a process has to be started (and terminated) on *each* of the nodes where a relation is declustered in order to read all the tuples in the relation. Therefore, the degree of declustering of a relation should be chosen such that the increase in startup and termination costs is offset by the benefits of increased parallelism. After choosing the degree of declustering, a data placement algorithm must select the particular nodes on which to decluster each relation. Load balancing is an important objective in this phase of the placement algorithm. Finally, for each relation, the placement algorithm must also determine the mapping of individual data tuples to the nodes.

* This research was done while the first author was a student at the University of Wisconsin-Madison. The research was supported by the IBM Corporation through a Research Initiation Grant.

[1] The OS community generally uses the term striping instead of declustering for horizontal partitioning.

This paper addresses only the first two data placement issues: choosing the degree of declustering and selecting the set of nodes on which to place each relation. Though we do not explicitly address the third issue (mapping individual tuples to nodes) in this paper, we do explore the effect of processing overhead on system performance. For a more detailed discussion on the declustering policy, the interested reader should see [Ghan90, Hua90, Ghan92, Falo93]. While the data placement strategy presented in [Cope88] also decided the placement of relations in memory, we feel that caching of relations in memory is better managed by a dynamic memory management policy, like the fragment fencing algorithm presented in [Brow93]. Such policies can adapt to runtime changes while a static data placement strategy cannot. Therefore, data placement in this paper refers only to the placement of relations on disks.

Although data placement has been studied extensively in the past and various algorithms have been proposed, there is no consensus as to the "best" placement strategy. Commercial systems such as Tandem and Teradata simply use full declustering; i.e. relations are declustered across all nodes of the system. On the other hand, previous placement studies [Cope88, Padm92, Rahm93a, Rahm93b] have concluded that partial declustering is better. Unfortunately, the studies that have advocated partial declustering have either not presented data placement algorithms based on partial declustering or developed algorithms that depend on a static analysis of the workload. In this paper we assume that database workloads are complex and change dynamically so that static workload analysis cannot be used. Moreover, developments in processor and network technologies have significantly changed the performance tradeoffs considered in previous data placement studies. The goal of this paper therefore is to develop a data placement strategy for SN parallel database systems that does not use static workload analysis and takes recent hardware trends into consideration.

The remainder of this paper is organized as follows. Section 2 describes the architecture of a SN parallel database system and Sect. 3 presents a detailed discussion of the various factors that determine the performance of data placement algorithms. The simulation model and the workload used for the performance analysis are presented in Sect. 4. Section 5 contains the results of experiments on selecting the degree of declustering while algorithms to select nodes for placing relations are studied in Sect. 6. Section 7 explores the effect of workload changes on the performance of the declustering algorithms while multi-class workloads are discussed in Sect. 8. Section 9 presents related work in data placement and presents a comparison with previous declustering studies. Finally, Sect. 10 contains our conclusions and suggestions for future work.

2 System architecture

Figure 1 presents a schematic description of a typical SN parallel database system. The system consists of a set of external terminals from which transactions are submitted. The transactions are sent to a randomly selected scheduling node. The execution of each transaction on the processing nodes is coordinated by a specialized process called the scheduler.

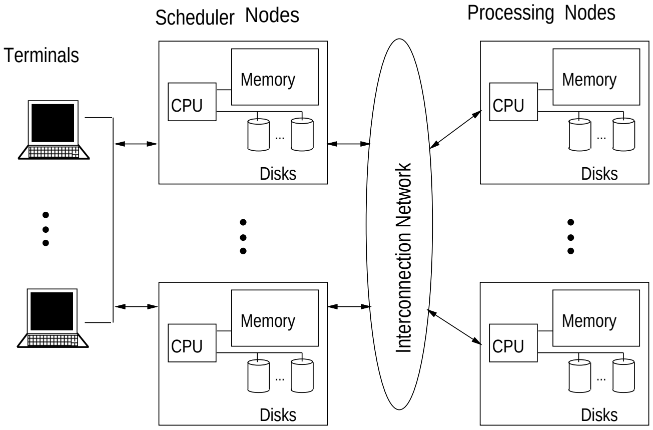

Fig. 1. Shared-nothing database system

The scheduler allocates resources (memory and processors) to the transaction and is responsible for starting and terminating all the operators in a transaction. The processing nodes are composed of a CPU, memory, and one or more disk drives[2]. The nodes use the interconnection network for all communication.

3 Data placement issues

This section discusses the factors that can affect the choice of the degree of declustering for relations and their placement on the system: startup and termination costs, communication costs, workload isolation, data skew, and result collection.

3.1 Startup and termination costs

As mentioned previously, the degree of declustering of a relation should be chosen such that the benefits of parallelism can offset the costs of operator startup and termination. The startup and termination of operators is handled by a specialized process called the query scheduler. The time consumed by startup and termination is dependent on the startup and termination protocols used by the query scheduler. We consider two alternative startup protocols in this study.

1. Parallel – In this protocol, the scheduler sends startup messages to all the nodes executing the operator and then waits for acknowledgments. Each node initiates an instance of the operator and then sends an acknowledgment back to the scheduler. The startup process is complete once acknowledgments have been received from all the nodes.
2. Sequential – In the sequential protocol, the scheduler sends a startup message to one node and then waits for an acknowledgment before sending the next startup message. Since the sequential protocol is obviously much slower than the parallel protocol, it is included only to simulate systems with very high startup costs. Only one termination protocol is modeled in this study. As each

[2] For the rest of this paper, the term node is used to collectively refer to a processor, its local memory and the attached set of disks.

instance of an operator completes, it sends a termination message to the query scheduler. The operator terminates once the scheduler receives a termination message from all operator instances.

3.2 Communication costs

[Cope88, Padm92, Rahm93a, Rahm93b] cite communication costs as an important factor in determining the degree of declustering. The degree of declustering determines the degree of parallelism of operators that access disk-resident data (e.g. select). A higher degree of declustering implies a higher degree of parallelism for these operators which, in turn, implies that data is read and possibly re-distributed on more nodes concurrently. If the bandwidth of the interconnection network is such that the network can become a bottleneck during re-distribution, a lower degree of parallelism and therefore a lower degree of declustering may be desirable. Although network bandwidth may have been a factor in determining the degree of declustering in the past, we assert that this is no longer the case. Several scalable interconnects have already been designed that provide extremely high bandwidths (up to 200 MB/s per node [Para93]). In comparison, data transfer rates from disks are only now reaching 10 MB/s [IBM93]. Therefore, data can be transmitted simultaneously from a large number of nodes without the network becoming a bottleneck.

3.3 Workload isolation

Typically, large database configurations are shared by many users performing a variety of tasks. For instance, a large system may execute workloads against multiple, independent databases. One problem in such an environment is workload interference. Users often want to prevent fluctuations in one workload from affecting other concurrent workloads. Data placement is often used as a mechanism to achieve workload isolation in these environments. By storing each database on a disjoint set of nodes, interference can essentially be eliminated[3]. Although disjoint placement provides complete workload isolation in an SN database system, it has a major drawback – a partitioned system is always less responsive to workload changes. Consider a workload consisting of two classes: the first class, called LowPriority, contains low-priority queries, and the other, HighPriority, consists of high-priority queries. If partitioning is used to isolate these two classes, one set of nodes will be dedicated to processing LowPriority queries while a disjoint set of nodes will be used to process HighPriority queries. If there is a sudden increase in the arrival rate of the HighPriority class, there will be increased contention in the corresponding part of the system and it will not be possible to distribute the additional load to the nodes processing LowPriority queries. In order to deal with such cases, a data placement strategy should allow for the use of flexible workload isolation mechanisms that can adapt to workload changes.

3.4 Skew

The variance in response times across the multiple instances of a parallel relational operator is called skew ([Wolf90, Hua91, Kits90, Omie91, Walt91, DeWi92b]). Since the degree of declustering determines the degree of parallelism of operators like select, skew is an important factor when selecting the degree of declustering for a relation. A taxonomy of data skew in parallel databases was presented in [Walt91]. Four types of skew were identified: skew in initial tuple placement (tuple placement skew), skew due to variations in predicate selectivity across nodes (selectivity skew), skew in redistribution of tuples in preparation for a join (redistribution skew), and finally, imbalance in the number of output tuples (join product skew). As in [Wolf90, Omie91, Walt91, DeWi92a], we assume that the relations are distributed such that there is no tuple placement or selectivity skew. Redistribution skew is also not considered, since techniques like the ones presented in [DeWi92a] can easily be used to eliminate it. On the other hand, remote-data access capabilities are required for reducing join product skew. For example, distributed virtual shared-memory was used in [Shat93] for handling join product skew. Since no remote data-access is assumed in this paper, join product skew may be present. Section 5.6 presents an experiment that explores the effect of join product skew on data placement.

3.5 Result collection

The output of a parallel operation sometimes needs to be merged into a single stream for transmission to an external processor. If the size of the output relation is large, the collection of result tuples can become a bottleneck [Padm92, Youn92]. However, since a single output stream may not always be required, and result sizes vary dynamically from query to query, we feel that the result size should not be a determining factor in selecting the degree of declustering of a relation. In this paper, we assume that join results are declustered on multiple disks; therefore, result collection is not a bottleneck in our experiments.

4 Simulator model

The performance studies presented in this paper are based on a detailed simulation model of a SN database system. The simulator is written in the CSIM/C++ process-oriented simulation language [Schw90] and models the database system as a closed queueing system. The following sections describe the configuration, database and workload models of the simulator in more detail.

4.1 Configuration model

The terminals model the external workload source for the system. Each terminal sequentially submits a stream of transactions. Each terminal has an exponentially distributed “thinktime” to create variations in arrival rates. All experiments in this paper use a configuration consisting of 128

[3] Assuming adequate network bandwidth to eliminate network contention.

nodes. The nodes are modeled as a CPU, a buffer pool of 16 MB[4] with 8 KB data pages, and one or more disk drives. The CPU uses a round-robin scheduling policy with a 5 ms timeslice. The buffer pool models a set of main memory page frames whose replacement is controlled via the LRU policy extended with "love/hate" hints [Haas90]. These hints are provided by the various relational operators when fixed pages are unpinned. For example, "love" hints are given by the index scan operator to keep index pages in memory; "hate" hints are used by the sequential scan operator to prevent buffer pool flooding. In addition, a memory reservation system under the control of the scheduler task allows buffer pool memory to be reserved for a particular operator. This memory reservation mechanism is used by hash join operators to ensure that enough memory is available to prevent their hash table frames from being stolen by other operators.

The simulated disks model a Fujitsu Model M2266 (1 GB, 5.25″) disk drive. This disk provides a cache that is divided into 32 KB cache contexts for use in prefetching pages for sequential scans. In the disk model, which slightly simplifies the actual operation of the disk, the cache is managed as follows: each I/O request, along with the required page number, specifies whether or not prefetching is desired. If prefetching is requested, four pages are read from the disk into a cache context as part of transferring the page originally requested from the disk into memory. Subsequent requests to one of the prefetched blocks can then be satisfied without incurring an I/O operation. A simple round-robin replacement policy is used to allocate cache contexts if the number of concurrent prefetch requests exceeds the number of available cache contexts. The disk queue is managed using an elevator algorithm.

The interconnection is modeled as an infinite bandwidth network so there is no network contention for messages. This is based on previous experience with the GAMMA prototype [DeWi90] which showed that network contention is minimal in typical SN PDBs. Messages do, however, incur an end-to-end transmission delay of 500 μs. All messages are "point-to-point" and no broadcast mechanism is used for communication.

Table 1 contains the configuration parameters while the CPU processing costs for various database operations are presented in Table 2.

Two kinds of configurations are considered in this paper:

- Disk-intensive: Each node in this configuration has a 40 MIPS CPU and one disk. This configuration represents systems whose workloads are typically I/O-bound.
- CPU-intensive: Nodes in the CPU-intensive configuration have a 10 MIPS CPU and four disks. This configuration models systems whose performance is typically CPU-bound.

Table 1. Simulator parameters and values

Configuration/Node parameter	Value
Number of nodes	128
Memory per node	16 MB
CPU speed	10/40 MIPS
Number of disks per node	1/4
Page size	8 KB
Disk seek factor [Bitt88]	0.617
Disk rotation time	16.667 ms
Disk settle time	2.0 ms
Disk transfer rate	3.09 MB/s
Disk cache context size	4 pages
Disk cache size	8 contexts
Disk cylinder size	83 pages
Wire delay for an 8K message	500 μs

Table 2. CPU cost parameters

Operation	Instructions
Initiate select operator	70000
Terminate select operator	5000
Initiate join operator	40000
Terminate join operator	10000
Apply a predicate	100
Read tuple from buffer	300
Probe hash table	200
Insert tuple in hash table	100
Start an I/O	10000
Copy a byte in memory	1
Send (receive) an 8K message	10000

4.2 Database model

The sizes of the input relations are chosen to explore a large range of query execution times. The database consists of sets of 100 relations for each of the following sizes (in number of tuples per relation): 5K, 10K, 15K, 20K, 25K, 50K, 75K, 100K, 200K, 300K, 400K, 500K, 1 Million. In addition, extra relations with 5 million tuples are used to achieve a disk occupancy of 60–70% for each configuration. The tuple size is 200 bytes for all relations, so there are 40 tuples per 8 KB page. We model both clustered and non-clustered B+ tree indices on the relations. The index key sizes are 12 bytes and the key/pointer pairs are 16 bytes long. The relations and indices are declustered across the nodes based on the particular declustering algorithm used in the performance study. Table 3 summarizes the database parameters used in this paper.

4.3 Workload model

The key workload characteristic for our study is the I/O access pattern. Although real workloads contain numerous transaction types with varied I/O behaviors, we use a simplified two-class workload that has been designed to capture

Table 3. Database parameters

Parameter	Value
Number of tuples	10 000–5 000 000
Tuple size	200 B
Tuples per page	40
Index key size	12 B
Index key/pointer pair size	16 B

[4] The simulated buffer pool size is smaller than buffer pools in typical configurations. Unfortunately, simulating a larger buffer pool size would require enormous amounts of resources. Some of our simulations took up to 90 and ran for 2.5 days on an IBM RS/6000 even with 16 MB of memory per node. On the other hand, our configuration is much more realistic than previous simulation studies ([Rahm93a] studied a configuration with 80 nodes and only 2 MB/node memory).

the essence of real workloads. The first class is designed to model short interactive transactions. Transactions of this class perform four single-tuple, non-clustered index selections on randomly selected relations. Each selection executes on one of the nodes on which the relation has been declustered. To process each selection, the transaction reads 3 pages (2 index pages and 1 data page). While each transaction reads a total of 12 pages, the total number of I/Os performed may differ due to variations in the buffer hit rate. Each of the four selections executes on a randomly chosen single node. Therefore the response time of the transactions is not affected by a relation's degree of declustering. This class is referred to as the Transaction class for the rest of the paper.

The second class, called the Join class, captures the I/O behavior of long-running batch queries and consists of binary hybrid hash-joins[5] [DeWi84, Gerb85] for most of the experiments. Binary join queries were chosen so that issues, like pipelining and intra-query parallelism, that arise when processing complex queries could be ignored. This is a reasonable simplification as most commercial database systems execute queries comprised of multiple joins as a series of binary joins and do not pipeline tuples between adjacent joins in the query tree. Moreover, such simplified query workloads have also been used previously in [Ng91,Yu93]. Consequently, complex multiple-join queries are included in only one of our experiments (Sect. 5.7). In order to maximize join processing costs, the declustering attribute is assumed to be different from the join attribute so complete redistribution of each relation is needed for join processing. Another simplification is that, unless stated otherwise, the selection predicates used in the joins have a 100% selectivity and no indices are used when processing the input relations. The effect of indices is examined separately in Sect. 5.5. Also except for Sect. 5.4, queries are always allocated their maximum memory requirement. Finally, in order to simplify processor allocation for the joins, the join is always executed on the same nodes on which the inner relation is declustered. Therefore, the degree of declustering of the inner relation also determines the degree of join parallelism and the response time is proportional to the degree of declustering of the inner (outer) relation. A higher degree of declustering decreases the amount of data to be read per node, improves the response time but also increases the overhead associated with parallel execution of the operator.

The number of transaction class terminals is fixed at 1000 in the simulation experiments and the number of query terminals varies from 1 to 40. The terminal think time are exponentially distributed and different mean values are used to change the load offered by each class.

5 Selecting the degree of declustering

The first series of experiments determines the degree of declustering for relations. Results are presented for the disk-intensive and CPU-intensive configurations in combination with both the sequential and parallel startup protocols. Except for Sect. 5.4, queries are always allocated their maximum memory requirement.

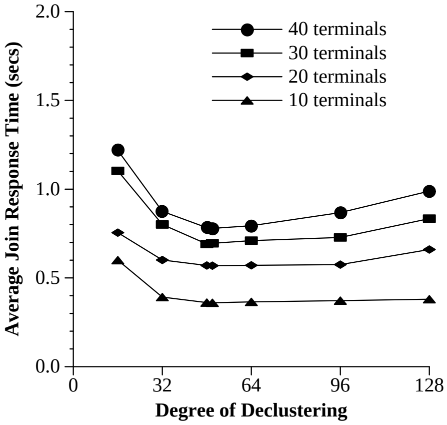

Fig. 2. Query response times for joins of 10K relations (disk-intensive configuration, parallel startup)

5.1 Disk-intensive configuration

5.1.1 Parallel Startup

The first experiment examines the effect of increasing the degree of declustering on queries of various sizes. The inner and outer relations have the same number of tuples, and the size of the input relations is used to label each query. For example, a 10K query refers to a join of two different randomly chosen 10 000 tuple relations. Figure 2 shows response times for 10K queries under various system loads as the degree of declustering is increased from 8 to 128. The declustering nodes for each relation are randomly chosen. Note that a degree of declustering of 128 represents full declustering.

As expected, the average query response time increases as the number of terminals (i.e. the system load) is increased from 10 to 40. With 10 terminals, response times initially decrease as the degree of declustering is increased from 8 to 50 and then start increasing slowly. The same behavior can also be observed for higher query loads with 20–40 terminals, but with sharper increases for higher degrees of declustering. Although startup and termination costs increase with a higher degree of declustering, the response time increases beyond a degree of declustering of 50 are not due to startup, termination, or communication costs. Beyond a degree of declustering of 50, there is less than one disk cache context worth of data per disk (recall that there is one disk per node in this configuration and each disk has a 256 KB cache divided into 32 KB cache-contexts). Therefore, the disk caches are not fully utilized beyond 50 nodes – one disk I/O fetches less than four 8 KB pages into the disk cache. This reduces the cache hit rate and increases disk arm contention. The resulting loss of I/O bandwidth causes the average query response time to increase. This increase in disk contention exacerbates the effect of lower I/O bandwidth, which is why the increase in response time is higher with more query terminals. A similar behavior can be ex-

[5] Since the primary factor of importance is the large number of I/Os performed by the queries, the use of other join methods (like nested-loop and sort-merge) will not change the results qualitatively.

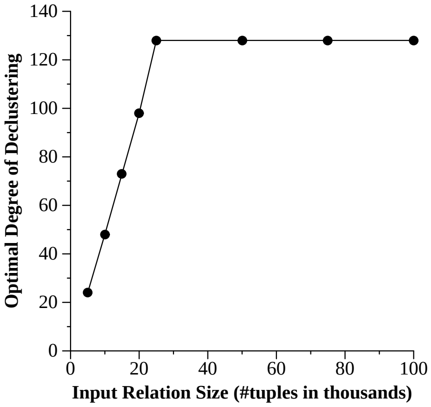

Fig. 3. Optimal degree of declustering (disk-intensive configuration, parallel startup)

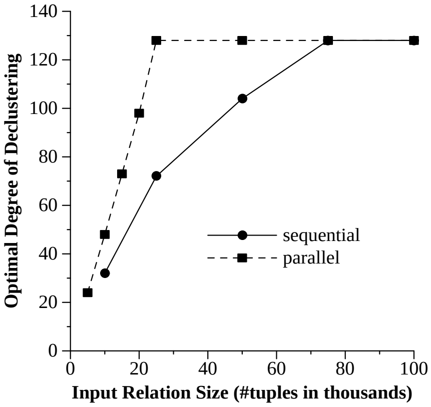

Fig. 4. Optimal degree of declustering (disk-intensive configuration)

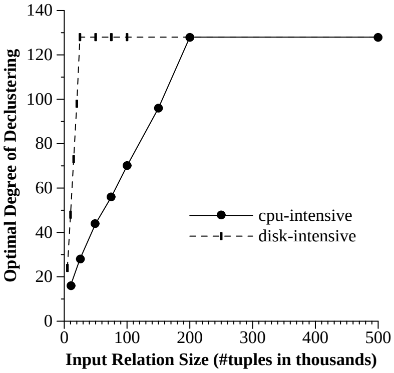

Fig. 5. Optimal degree of declustering parallel startup

pected if multiple-page I/Os are used instead of the cache to reduce disk-arm contention.

In order to verify the relationship between cache context size and the optimal degree of declustering, the same experiment was performed for relations of different sizes. Figure 3 shows the degree of declustering that achieves the lowest average response times, called the optimal degree of declustering, for relations with 5K to 100K tuples. The results show that the optimal degree of declustering increases linearly as the relation size is increased from 5K to 25K tuples. Moreover, for each relation size, the optimal degree of declustering occurs when there is just one disk cache context worth of data per node. The optimal degree of declustering cannot be greater than the system size (128 nodes) and therefore the optimal degree of declustering is constant beyond a relation size of 25K tuples. The results of these experiments demonstrate that a simple cache-context-per-disk rule can be used in this disk-intensive configuration with parallel operator startup to determine the maximum degree of declustering of relations.

5.1.2 Sequential startup

Figure 4 shows the optimal degree of declustering when operator instances are started sequentially by the scheduler (solid line). Compared to the results with parallel startup (dotted line), the optimal degree of declustering for the relations is lower and no longer has a linear relationship with relation size. For instance, if the relation size is increased by 250% from 10 000 tuples to 25 000 tuples, the optimal degree of declustering increases by 225% (from 32 to 72). Increasing the relation size further by 100% to 50 000 tuples increases the optimal degree of declustering by only 44% (to 104). The non-linear relationship of the optimal degree of declustering, d_{opt}, with the relation size can be explained as follows. If startup and termination costs are significant, the response time of a query can be modeled as [Wils92]

$$\text{ResponseTime} = a * d + \frac{b * S}{d} \qquad (1)$$

where a = startup and termination cost per node, d = degree of declustering of the input relation, b = processing cost per tuple, and S = number of tuples in the relation. The optimal degree of declustering for a relation with S tuples, $d_{opt}(S)$, can be found by differentiating the above equation with respect to d and equating it to 0. This leads to the following formula

$$d_{opt}(S) = \sqrt{\frac{b * S}{a}} \qquad (2)$$

This shows that d_{opt} is a non-linear function of S. However, even with sequential startup, d_{opt} rises rapidly as the size of the input relations increases. As a result, all relations with more than 75K tuples (15 MB) should still be fully declustered for this disk-intensive configuration.

5.2 CPU-intensive configuration

5.2.1 Parallel startup

Figure 5 shows the optimal degree of declustering for relations using the parallel startup algorithm in the CPU-intensive configuration (one 10 MIPS CPU and 4 disks per node). The optimal degree of declustering for the disk-intensive configuration is also shown for comparison. In the CPU-intensive configuration, only relations with more than 200K tuples should be fully declustered (solid line) compared to 25K tuples for the disk-intensive configuration (dashed line). Since startup and termination take longer with a slower CPU, they constitute a larger fraction of the total execution time of a query. As a result, the number of tuples processed per node must be increased in order to compensate for the increase in startup and termination costs.

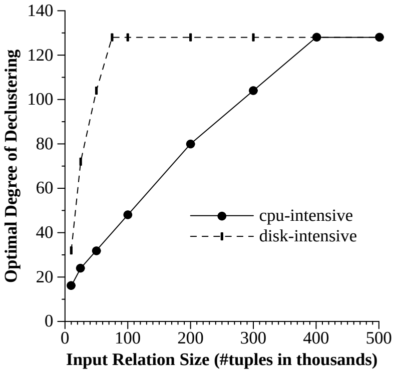

Fig. 6. Optimal degree of declustering sequential startup

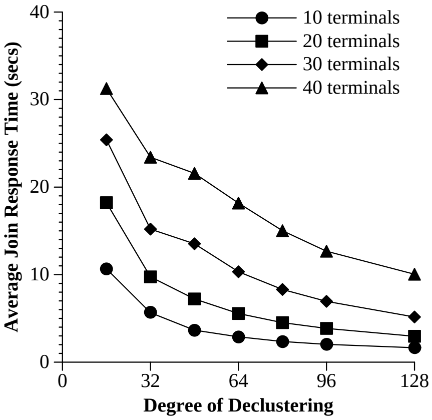

Fig. 7. Joins of 100K relations: minimum memory allocation

5.2.2 Sequential startup

The next experiment explores the effects of sequential startup in a CPU-intensive configuration. Figure 6 shows that only relations with more than 400K tuples should be fully declustered. The reason is that sequential startup increases the query startup time considerably and the slower CPU in this configuration exaggerates the effect. Therefore, the number of tuples at which full declustering becomes optimal increases from 200K to 400K tuples. Figure 6 also shows a non-linear relationship between the relation size and the optimal degree of declustering, as we saw before for sequential startup in the disk-intensive configuration.

5.3 Discussion

The previous experiments have explored the optimal degree of declustering for disk-intensive and CPU-intensive situations using both sequential and parallel startup algorithms. The optimal degree of declustering was highest for the disk-intensive configuration with parallel startup. It is lower for a CPU-intensive system or if a sequential startup algorithm is used. However, even for the slowest system (CPU-intensive, sequential startup), any relation with more than 400K tuples (80 MB), should be fully declustered. Therefore, only 625 KB of data is needed per node to overcome startup and termination costs for this configuration. These results indicate that most relations in a typical parallel database system should be fully declustered for the best performance.

The previous experiments also demonstrate that the optimal degree of declustering is a complex function of hardware and software parameters. The optimal degree of declustering had a linear relationship with relation size in the disk-intensive system with parallel startup, but a non-linear relationship for all other configurations. This means that if startup and termination costs are low and the system is disk-intensive, a simple rule like cache-context-per-disk (Sect. 5.1) can be used to determine the optimal degree of declustering. Fortunately, current hardware trends indicate that database systems are likely to be disk-intensive in the future. CPU speeds are increasing at an average of 50% each year [Sigm94]. In contrast, disk bandwidths are growing at only 20% a year [Sigm94]. Thus, future systems will most likely be disk-intensive, and fast CPUs and networks will further reduce startup and termination costs. Consequently, determining the optimal degree of declustering should become easier in future systems. For the remainder of this paper, a cache-context-per-disk rule is used to calculate the degree of declustering for each relation. Furthermore, results are presented only for the disk-intensive configuration with parallel startup.

5.4 Minimum memory allocation

Since startup and termination costs are the same when join queries are given their maximum or minimum memory allocation, the declustering results of the previous section (where maximum allocation was used) should be applicable even if minimum memory allocation is used. Figure 7 shows the effect of the degree of declustering on the response time for joins of two 100 000 tuple relations with minimum memory allocation for various system loads. The results confirm that full declustering (degree of declustering = 128) provides the best response time for all system loads. Experiments with larger relations produced the same result – full declustering had the best performance[6].

5.5 Effect of indices

The previous experiments used full relation scans of both input relations. The next experiment examines the effect of index scans on the degree of declustering. Figures 8 and 9 show the performance of joins of two one million tuple relations when a clustered index is used to read each relation. Index selectivities of 1% and 10% were considered. The results of this experiment are quite different from the previous ones. Figure 8 shows that, with a 1% selectivity factor, the optimal degree of declustering for a relation with 1 million tuples is 50 (compared to 128 in the previous section). This implies that the degree of declustering should be reduced if index scans are used to read relations. However, Figure 9 shows that if the selectivity is increased to 10%, the optimal degree of declustering increases to 128. The results of these two experiments can be explained as follows. The optimal

[6] Experiments were not done using smaller relation sizes because previous studies [Meht93, Meht94, Yu93] have shown that minimum memory allocation is justified only for large relations where it leads to substantial reduction in memory consumption.

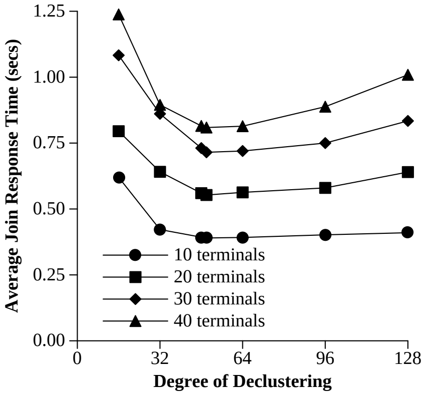

Fig. 8. Index selectivity = 0.01. Effect of clustered index scans on the degree of declustering

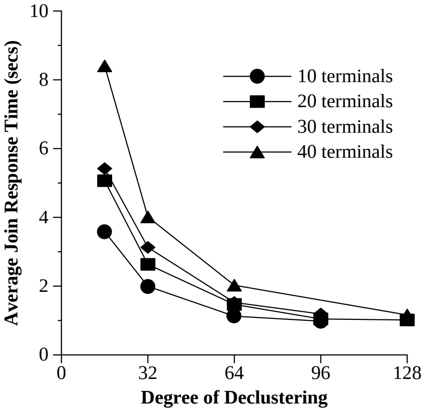

Fig. 9. Index selectivity = 0.1. Effect of clustered index scans on the degree of declustering

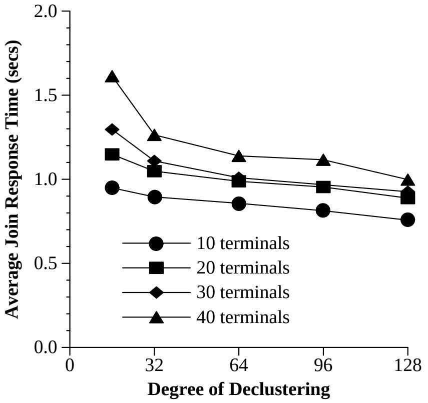

Fig. 10. Skew Factor – 400%. Effect of skew on the degree of declustering

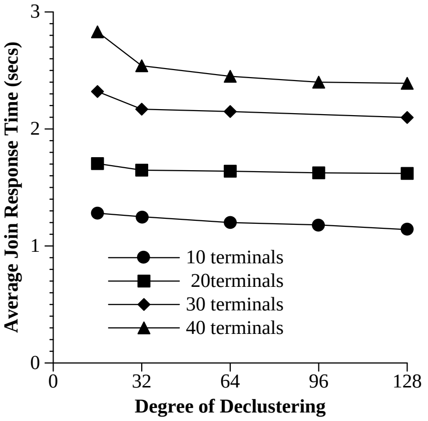

Fig. 11. Skew Factor – 800%. Effect of skew on the degree of declustering

degree of declustering depends on the number of tuples processed by each operator. Since a query using a clustered index scan with a 1% selectivity on a 1M relation processes the same number of tuples (10 000) as a file scan of a 10 000 tuple relation with 100% selectivity, the optimal degree of declustering in both cases is also the same. Similarly, the optimal degree of declustering of a query using a clustered index scan with a selectivity of 10% is similar to the optimal degree of declustering of a file scan of a 100 000 tuple relation. In general, if a clustered index scan with x% selectivity is used to read a relation of with R tuples, then the optimal degree of declustering is the same as the optimal degree of declustering for a file scan of a relation with $(R * x)$ tuples. Therefore, the optimal degree of declustering of a relation changes depending on the selectivity of the index scans used to read the relation. However, the optimal degree of declustering increases rapidly with an increase in relation size (Fig. 3), and therefore full declustering will provide the best performance unless the index selectivity is small. Experiments with unclustered index scans also showed that full declustering provides the best performance unless the selectivity is very small and have therefore been omitted.

5.6 Effect of skew

This experiment explores the effect of data skew on the cache-context-per-disk rule. As mentioned in Sect. 3.4, only the effect of join product skew is considered. Join product skew is modeled in the following manner. It is assumed that the skew is present on only one of the processing nodes, which is chosen randomly; this is similar to the "scalar" skew model used in [Walt91], [Omie91] and [DeWi92]. The number of tuples produced at the selected node is calculated as the skew factor times the average number of tuples produced at other nodes. For example, if each node produces 10 000 result tuples on the average, a skew factor of 400% means that the node that experiences join product skew will produce 40 000 $(4 * 10\,000)$ tuples. In order to ensure that the same number of tuples are produced irrespective of the degree of declustering, the skew factor is increased linearly with the degree of declustering. For instance, a skew factor of 400% with a degree of declustering of 16 changes to 800% and 1600% when the degree of declustering increases to 32 and 64 respectively.[7] Linearly increasing the skew factor with the degree of declustering ensures that the same number of tuples are produced on the skew node for each degree of declustering. Figures 10 and 11 show the effect of two different degrees of skew on the average response time of 25K join queries (without indices).

In all the cases, the response time decreases with an increase in the degree of declustering. However, the reduction in response time is small for low loads (less than 20 ter-

[7] This represents an extremely high degree of skew. If a more realistic distribution was used instead of the scalar skew model, the increase in skew with the degree of declustering would be more gradual.

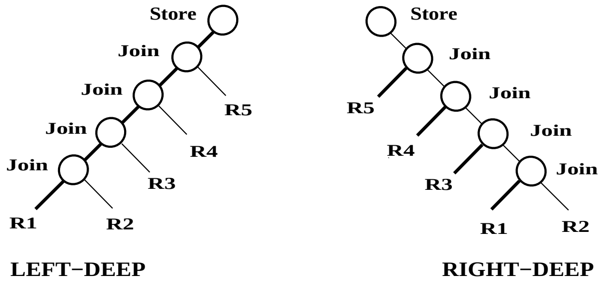

Fig. 12. Complex query schedules

minals). The reason is that the response time of the queries is dominated by the response time of the node that experiences the join product skew. Since the number of tuples produced at the skewed node does not change as the degree of declustering increases, the response time also does not change significantly. This is more evident in Fig. 11, where queries have a higher skew of 800% and hence the response time reduction is lower. At higher loads, the performance improves slightly with a higher degree of declustering since the query load gets spread over more nodes. This experiment shows that full declustering improves performance in the presence of low degrees of skew.

While there is less improvement with higher degrees of skew, full declustering still has the best performance. In the presence of skew, full declustering often helps in reducing the average response time and never leads to higher response times.

5.7 Complex queries

So far we have considered only binary join queries. This section examines the effect of declustering strategies on more complex queries. The workload used here consists of a single five-way join query where each input relation consists of one million tuples. The selection selectivity is 100%, join selectivity is 50% and the joins are allocated their maximum memory allocation. There are multiple ways of scheduling such a complex query, each differing in the amount of parallelism and pipelining exploited [Schn90, Chen92a, Chen92b]. In this experiment, both left-deep and right-deep scheduling [Schn90] are considered. These represent the two extremes in query scheduling strategies; left-deep schedules have the least parallelism and limited pipelining, while right-deep schedules have the highest parallelism and maximum pipelining. The two query schedules are shown in Fig. 12. The dark edges in the two schedules represent the build operation in the hash join, while the lighter edges represent the probe operation.

Figure 13 shows the response time for the complex queries as the degree of declustering of the input relations is increased from 16 to 128. The results show that increasing the degree of declustering improves performance irrespective of the query scheduling strategy; full declustering provides the best performance for both left-deep and right-deep trees. The reason is the same as before – the increase in parallelism with higher degrees of declustering more than compensates for the small increase in startup and termination costs. This experiment illustrates that full declustering not only improves the performance of simple binary join queries, but also complex join queries. Similar experiments with complex queries containing different numbers of multiple joins also produced the same results.

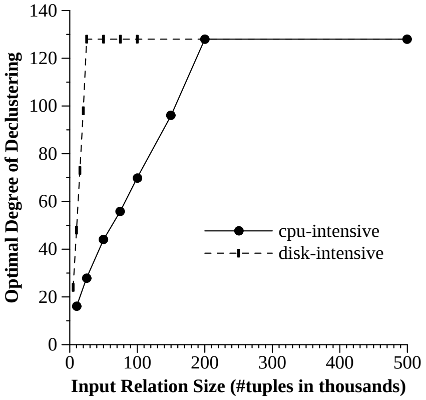

Fig. 13. Complex-query response times 5-way join queries (1M tuples per input relation)

6 Selecting nodes for placement of relations

This section presents and evaluates alternative strategies for deciding the placement of the fragments of a declustered relation. The previous sections showed that a simple cache-context-per-disk rule can be used to determine the degree of declustering of relations. Therefore, placement needs to be determined only for relations that should not be fully declustered under this rule (since if a relation can be fully declustered, it is simply placed on all nodes and no placement decision is needed).

The first part of this section (6.1) presents three alternative algorithms for placing relations that should not be fully declustered. This is followed by a detailed performance evaluation of the resulting (complete) data placement algorithm on a variety of workloads. The workloads have been divided into partitioned and non-partitioned workloads. A partitioned workload consists of disjoint sets of queries which access mutually exclusive sets of relations. A workload consisting of queries submitted from two separate organizations accessing their own private databases is an example of a partitioned workload. The queries from each organization can be isolated from one another by placing their relations on disjoint sets of nodes. However, disjoint placement of relations cannot be used to partition all workloads; such workloads are called non-partitioned. An example of a workload that cannot be partitioned is the TPC-C benchmark workload. This workload consists of five transaction types that access data in a database composed of ten relations. Each relation is shared across multiple transactions, making it impossible to partition the workload by placing the relations on disjoint nodes. Section 6.2 examines the effect of declustering on non-partitioned workloads, while partitioned workloads are considered in Sect. 6.3.

Table 4. Workload and database parameters

Workload			Database		
Class	Terminals	Think time	Relation	Size	Number
Join	10	0 s	Partially declustered	1000–25 000 tuples	10–100
			Fully declustered	100 000 tuples	100

6.1 Handling small relations

Small relations, which cannot be declustered on all nodes, are a potential source of load imbalance because their placement leads to increased load on only a subset of nodes. The experiments in this section compare three algorithms for placing small relations. For each algorithm, the degree of declustering is determined using the cache-context-per-disk rule. However, the algorithms differ in the order in which relations are chosen for placement and in the method used to select the nodes for placing each relation. The following placement algorithms are considered:

- Random: The first algorithm is the simplest and is completely workload independent. It randomly chooses both the next relation to be placed and the set of nodes on which to place it.
- Round-robin: In the round-robin algorithm, the order in which relations are placed is random, but the placement of relations on the nodes is performed in a round-robin fashion: if a relation is placed on processors 1 through 10, the placement of the next relation starts at processor 11, and so on. This algorithm is also workload independent, but tends to spread the data more uniformly than Random.
- Bubba: The final placement algorithm is the Bubba data placement heuristic [Cope88]. In this scheme, relations are placed in decreasing order of their access frequency (*heat*). For each node, the algorithm maintains the sum of the heats of all of the relations placed so far on the node. For each relation, the nodes with the least accumulated heat are chosen for declustering. This algorithm is the most efficient for load balancing but it requires detailed knowledge of the workload for an accurate prediction of each relation's heat. The placement algorithms are first compared for a non-partitioned workload, and the results are then used to determine data placement for partitioned workloads.

6.2 Non-partitioned workloads

The relations in a non-partitioned workload can be divided into two categories: fully declustered relations, which are partitioned across all nodes, and partially declustered relations, which are declustered across only a subset of nodes. The first experiment compares the performance of the algorithms for placing small relations on a disk-intensive configuration. The workload and the database used for this experiment are described in Table 4.

The database consists of one hundred 100 000 tuple relations that are fully declustered and a variable number of small relations that are partially declustered. The number of tuples in the small relations vary uniformly between 1000 and 25 000 tuples. The degree of declustering for partially declustered relations is determined by the cache-context-per-disk rule. For example, the 1K tuple relation (25 pages) is declustered on 5 disks[8] and the 10 000 tuple relation (250 pages) is declustered on 50 disks. The workload consists of a number of concurrently executing hash join queries; the number of queries was chosen such that the average disk utilization was 60%. The selection of the inner and outer relation for each query is performed as follows. The relation can be a partially or fully declustered relation with equal probability. If the relation is selected to be a partially declustered relation, a skewed Zipf distribution[9] with $z = 1.0$ is used to choose the particular partially declustered relation. This distribution was selected to vary the heat among the partially declustered relations. On the other hand, the placement of fully declustered relations is fixed and independent of the heat of the relation. Therefore, if an input relation is chosen to be a fully declustered one, it is chosen randomly with a uniform distribution from among the set of 100K tuple relations. The skewed probabilities (defined by the Zipfian distribution) are randomly assigned to the various partially declustered relations. Therefore, in order to remove the effect of this random assignment of access probabilities to relations, every result point represents the average of 30 simulation runs, each with a different assignment of access probabilities to the partially declustered relations.

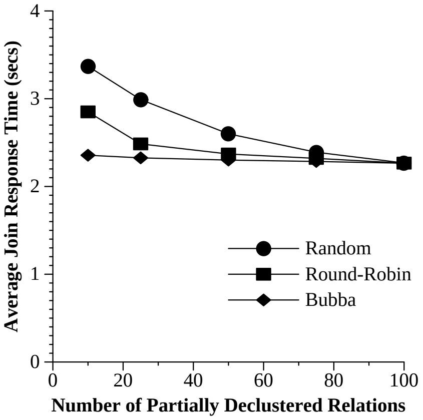

Fig. 14. Effect of number of partially declustered relations

Figure 14 shows the performance of the three algorithms as the number of partially declustered relations is varied from 10 to 100. In order to stress the performance of the algorithms, all of the data was read from the disk and memory buffering was switched off. The effect of memory buffering is examined separately in the next experiment.

[8] There are 40 tuples per page and one disk cache context (5 pages)per disk.

[9] $P(i) = \frac{1/i^z}{\sum_{i=1}^{N} 1/i^z}$ for all $1 \leq i \leq N$, where N is the number of relations [Zipf49].

Figure 14 shows that, with 10 partially declustered relations, the Random algorithm results in the highest average query response time. The Random algorithm places relations on a random set of nodes and the order of placement is also randomized. When the number of partially declustered relations is small, the Random algorithm can thus place fragments of multiple "high-heat" relations on the same node. This leads to a load imbalance, so the performance of the Random algorithm is the worst. The Round-Robin algorithm distributes the small relations more evenly across the nodes and thus has better performance. However, since the order in which relations are placed is random, fragments of two relations with high heat can still be placed on the same node. Therefore, the heat of the relations is still not spread uniformly. On the other hand, the Bubba algorithm places relations in decreasing order of heat and attempts to place fragments of "high-heat" relations on disjoint nodes. This technique achieves better load balancing and performs the best among the three algorithms.

As the number of partially declustered relations is increased in Figure 14, the performance of both the Random and Round-Robin algorithms improves because the effect of randomness decreases. With just 25 small relations, the performance of the Round-Robin algorithm is quite similar to the Bubba algorithm, and the performance of all three algorithms is nearly identical with more than 80 relations. The experiment therefore shows that Bubba is the best placement algorithm, but simpler algorithms like Random and Round-Robin also perform well as the number of small relations increases.

The previous experiment assumed that all the data was disk-resident. In practice, data is often kept memory resident to improve performance (LRU, Gray's 5 min rule [Gray87], Bubba memory management [Cope88], Fragment Fencing [Brow93]). Figure 15 shows the performance of the algorithms as the percentage of partially declustered relations that are kept in memory is varied. The relations were kept memory resident by pre-reading and pinning the appropriate percentage of the data pages of the small relations in memory at system startup. The number of partially declustered relations was fixed at 10 since this is the value that showed the largest difference between the algorithms in the previous experiment. The first point in all the curves, with residency equal to 0, is the same as the value shown in the last experiment and shows the maximum difference between the three algorithms. However, as the residency increases the performance of the three algorithms becomes very similar. Beyond 50% residency, the algorithms have almost identical performance.

These two experiments have examined the performance of three different placement algorithms for handling relations that are too small to be fully declustered under the cache-context-per-disk rule. The results show that the Bubba algorithm has the best performance, followed by Round-Robin and then Random. However, the detailed workload knowledge required by the Bubba algorithm may not always be available, and thus the Bubba algorithm cannot be used in all cases. In such cases, the simplistic Round-Robin algorithm can be used. Even though the algorithm has no knowledge of the workload, it was seen to perform quite well. In addition, the last two experiments showed that if the small relations are partially memory resident (which will quite often be the case), or if there are a large number of small relations, the performance of the Round-Robin algorithm is nearly identical to the Bubba algorithm.

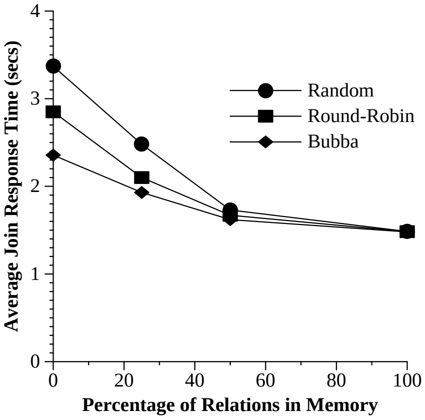

Fig. 15. Effect of memory residency

6.3 Partitioned workloads

All the experiments shown so far have included only non-partitioned workloads. This section examines partitioned workloads to evaluate the effect of declustering on workload isolation. Initially, a simple two-class workload is studied to give some insight into the problem. The results are then used in the following sections to study workloads with shift changes and a more general three-class workload.

Partitioned workloads consist of disjoint components which access mutually exclusive sets of relations. Therefore, the workload can be partitioned by placing the relations on disjoint sets of nodes. However, determining the size of the partitions to be created for each workload component is a difficult problem. The first experiment in this section studies different partitioning strategies for a two-class workload and compares their performance to a non-partitioned system with full declustering. The results are then used to study a workload with shift changes.

The workload consists of two classes, Transactions and Queries, which have been described in detail in Sect. 4. All the experiments are conducted on a 128 node system with 16 MB of memory and 1 disk per node. We study the performance as the number of nodes dedicated to transaction processing is increased from 16 to 96 nodes. and the number of nodes dedicated to query processing is decreased from 112 to 32. For each partitioning method, the performance is compared to a non-partitioned system where both queries and transactions execute on all the nodes. The experiment thus explores systems where the percentage of the system devoted to transactions (queries) varies from 12.5% (87.5%) to 75% (25%).

Table 5 describes the various parameters for each of the classes in the workload. The think times of the transaction terminals are set such that the disk utilization is around 60%. For example, when the number of nodes executing transactions is 16, the think time is 8.8 s. The think time is reduced to 0.95 s when 96 nodes are used to execute transactions and

Table 5. Partitioned workload

	Workload			Database	
Class	Terminals	Think time	Number of nodes	Relations	Size
Transaction	1000	0.95–8.8 s	16–96	Transaction	5 000 000 tuples (1 Gb)
Join	1	0 s	112–32	Join	5 000 000 tuples (1 Gb)

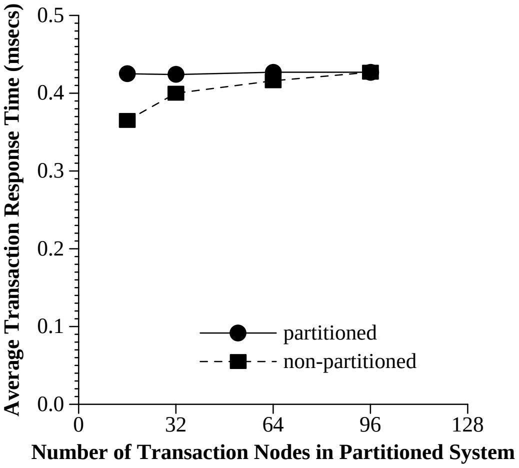

Fig. 16. Transactions

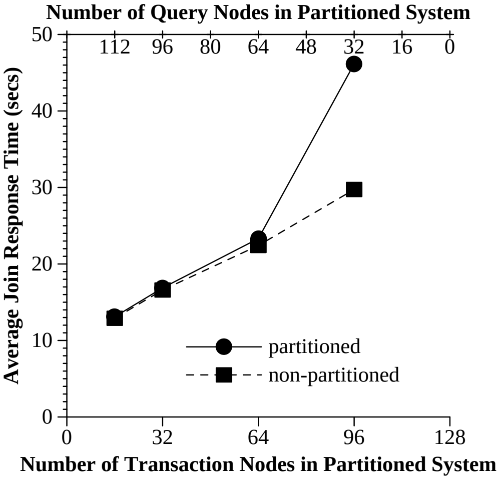

Fig. 17. Joins

a higher transaction load is needed to achieve 60% disk utilization. The MPL (Multi-Programming Level) for the query class has been fixed at 1 with a 0 think time (the effect of varying query MPL is explored in a later experiment). A simplified database is used to ease the exposition of the results; all relations are the same size and contain five million tuples. Each class accesses a mutually exclusive set of relations and the number of relations in each set is chosen to occupy about 60% of the disks. The rest of the experiments in the paper also use this simplified database. The selectivity of the join queries is also reduced to 10% to enable the inner relations of multiple join queries to fit in memory. The performance of the two classes for both partitioned and non-partitioned systems is shown in Figs. 16 and 17.

In the partitioned system, since the transaction load is configured such that the overall disk utilization is always around 60%, the response time of the transactions is relatively flat as the number of nodes executing transactions increases; however, transaction throughput increases significantly. The performance in the non-partitioned case is different. When the number of nodes processing transactions is small (e.g. 16), the average transaction response time is 17% lower in the non-partitioned system than in the partitioned system. As the number of transaction nodes increases, the transaction load also increases and the interference from the join query causes the average transaction response time to increase. The average transaction response time of the non-partitioned system is the same as the partitioned system when 96 nodes are dedicated to transaction processing in the partitioned system.

Now consider the performance of the join query (Fig. 17). The response time of the join increases significantly as the number of nodes processing transactions increases (and consequently, the number of nodes processing queries decreases). In the non-partitioned system, the join response time increases as the transaction load increases due to interference from the transactions. However, the increase in response time is much smaller than in the partitioned system since the join always executes on 128 nodes and there is no reduction in join parallelism. This experiment demonstrates that executing the workload of a partitioned system on a non-partitioned system can improve the performance of both transactions and queries. However, transaction response times may degrade in a non-partitioned system due to interference from concurrently executing queries. The next experiment explores whether this increase can cause a non-partitioned system to perform worse than a partitioned system. We fixed the number of dedicated transaction nodes at 96 since this configuration had the worst transaction performance for the non-partitioned case in the previous experiment. Figures 18 and 19 show the performance of the two declustering schemes as the number of queries is increased from 1 to 4.

In the partitioned case, the transactions execute on a separate set of nodes from the joins and therefore their performance remains unaffected as query MPL increases. The query response time, however, increases steeply as the MPL increases. The queries in the non-partitioned system perform much better than the partitioned case as a result of increased parallelism. However, interference from the queries causes the transactions to suffer and the average transaction response time increases with an increase in query MPL. The experiment shows that although query interference is not an issue at a low query MPL, some additional mechanism is needed to control the interference at higher loads. We evaluated three mechanisms that can be used to limit interference:

1. Disk priority scheduling: This mechanism gives higher priority to short interactive transactions at the disk while queries that perform long sequential scans have lower priority. High-priority requests are serviced first at the disk and low-priority requests are serviced only in the

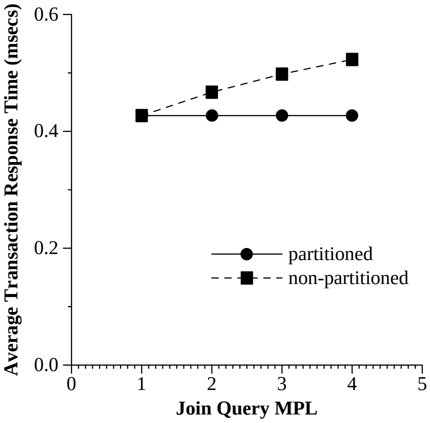

Fig. 18. Transactions

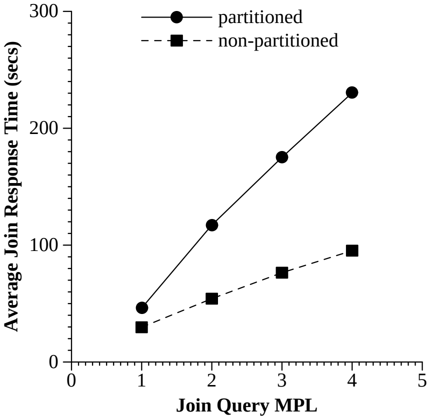

Fig. 19. Joins

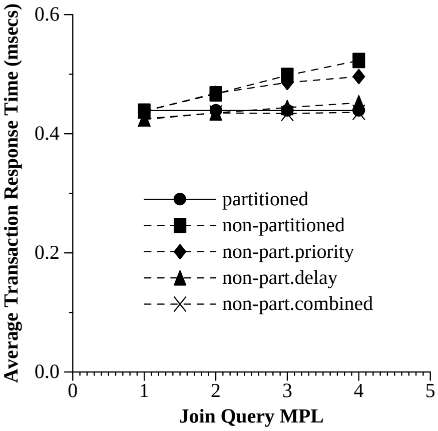

Fig. 20. Transactions

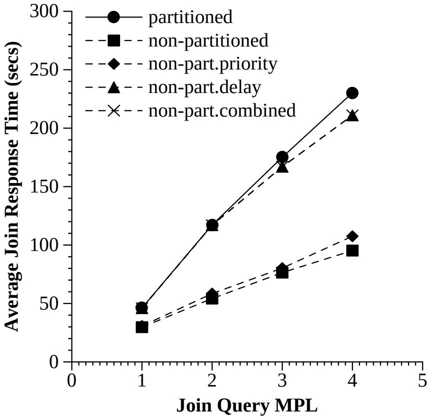

Fig. 21. Joins

absence of high-priority requests. This significantly reduces query interference perceived by transactions at the disk. This mechanism is used in the Tandem Non-Stop SQL product [Engl91].

2. Dynamic delay: The second mechanism limits interference by dynamically delaying queries before execution. This reduces the average execution MPL of queries in the system leaving more disk bandwidth for the transactions[10]. In a real system, the delay period would be dynamically set to match an expected response time (that could be user-specified or determined automatically by the system, e.g. optimizer estimates). For the purposes of this experiment, queries are delayed in the non-partitioned system until the average response time is equal to the corresponding query response time in the partitioned system. Therefore, a query is delayed by the amount of time by which previous query exceeded the expected average response time. A more detailed discussion about the use and setting of delay times can be found in [Brow94].
3. Combined: The last mechanism combines the two previous schemes. Priority scheduling is used at the disk and queries are also delayed dynamically to reduce interference.

The performance of these mechanisms is shown in Figs. 20 and 21, which show average transaction and join query response times as a function of the query MPL. The performance of the partitioned and non-partitioned schemes is also shown for reference.

The disk priority scheme (dashed line with diamonds) slows down the queries and reduces query interference at the disk by giving higher priority to the transactions. However, the reduction in interference is not enough and the transactions have a higher response time than in the partitioned system when the query MPL is greater than 1. The delay scheme (dashed line with triangles) delays the queries such that their response times are close to the query response time in the partitioned case. This reduces interference and lowers the average transaction response time. At low query MPLs (below 3), this scheme achieves lower transaction response times than the partitioned case. However, the reduction in interference is not enough and at higher query MPLs the transactions still have a higher response time. The combined scheme (dashed line with crosses), which delays queries and uses priority disk scheduling, performs the best and is able to limit interference even for high query MPLs[11].

The previous two experiments demonstrated that non-partitioned placement with full declustering out-performs

[10] Note that the reduction in average execution MPL is not just due to the use of a closed system. A delay mechanism along with a maximum MPL limit will result in a reduced execution MPL even in open systems.

[11] Although only average response time figures have been reported here, we also compared the 90th percentile values and the results were similar qualitatively.

Table 6. Partitioned workload with shift changes

Workload	Transaction terminals	Transaction think time	Query terminals	Query think time
Shift 1	1000	0.95 s	2	0 s
Shift 2	500	0.95 s	4	0 s

partitioned data placement except under high-load conditions when the transactions perform poorly due to query interference. In such cases, simple mechanisms like priority disk scheduling and dynamic delay of queries can be used to limit interference and outperform partitioned placement.

7 Handling workload shifts

As mentioned in Sect. 3, an important factor in the performance of a declustering scheme is its ability to provide good performance in the presence of workload shifts. The next experiment compares the performance of partitioned and non-partitioned placement for a two-class workload with shift changes. As in the previous partitioning experiment, the two classes in the workload are Transactions and Joins. Table 6 shows the structure of the classes for each workload shift. The workload models typical shifts in mixed workloads: higher transaction processing in the first shift and higher query processing in the second. In the first shift, the transaction workload is the same as the one used in the previous experiment (1000 terminals with 0.95 s think time) and there are two queries in the system. The transaction workload halves in the second shift while the query workload doubles.

This experiment compares the performance of a partitioning scheme (that allocates 96 nodes for transaction processing and 32 nodes for query processing) to two non-partitioning strategies. The first non-partitioning strategy, labeled Non-Partitioned Match, is similar to the method used in the previous experiment – the combined scheme (disk priority scheduling and query admission delay) is used to match query response times of the partitioned scheme. The second method, called Non-Partitioned Constant Trans, uses the combined mechanism to maintain the a constant transaction response time in both shifts. The performance of all the schemes is shown in Figs. 22 and 23.

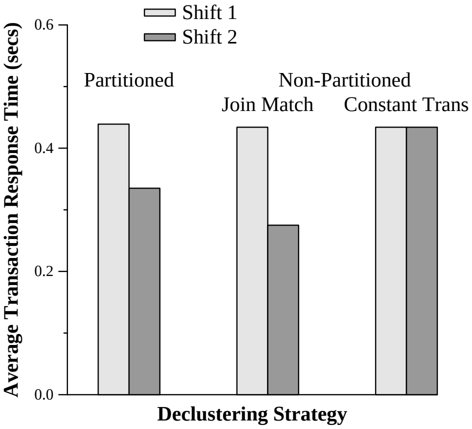

Fig. 22. Effect of shift changes – transactions

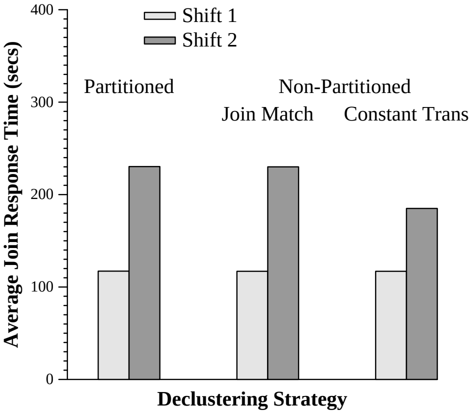

Fig. 23. Effect of shift changes – joins

For the partitioned system, transaction response times decrease in the second shift as the transaction load decreases while the query response times increase because the query workload increases. The partitioned system statically partitions the system between workloads and therefore prevents the system from adapting to dynamic workload changes. For example, the decrease in the transaction workload in the second shift cannot be used to improve the performance of the query class. On the other hand, the delay mechanism in the non-partitioned system can be used in multiple ways to modify the performance of the two classes in each shift. The Non-Partitioned Join Match scheme achieves query response times that are identical to query response times in the partitioned system and transaction response times that are better. The Non-Partitioned Constant Trans scheme, on the other hand, delays the queries such that the same transaction response time is maintained in both shifts. The resulting reduction in transaction workload in the second shift allows lower join query response times in the second shift (compared to the response times achieved in the second shift by the Partitioned and Non-Partitioned Join Match schemes).

This experiment has shown that not only can non-partitioned data placement with full declustering perform better than a partitioned system for static workloads, it is also a better strategy for dynamically varying workloads since schemes like query delaying and priority scheduling can be used to adapt the performance of the system as the workload changes. Partitioned placement obtains workload isolation by placing data on disjoint nodes but this prevents the use of dynamic mechanisms that can adapt to changes in the workload.

8 Handling multiple classes

Our final experiment compares the performance of the partitioned and non-partitioned systems for a more general three-class workload. The database used for this experiment is the same as that used in the previous experiment – all relations

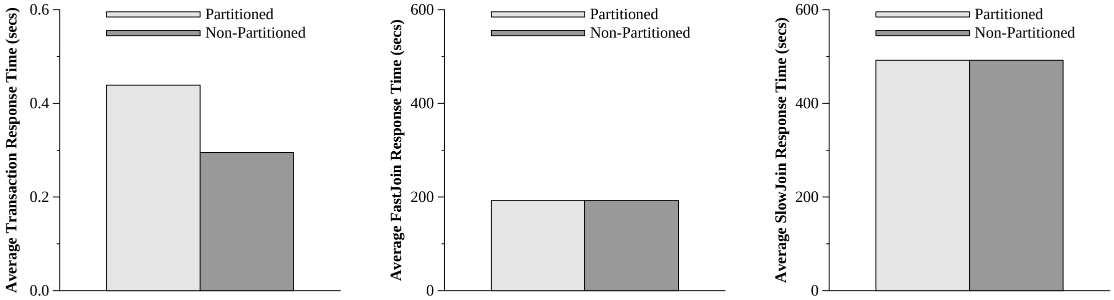

Fig. 24. Multiclass workload

have five million tuples. However, in addition to transactions, the workload has two query classes with very different query response times and resource requirements. The first query class, SlowJoins, represents a class with low memory allocation and high response times. Queries from this class are allocated only their minimum memory requirement [$\sqrt{sizeofinner}$ (34 MB)] and need two passes of the hybrid-hash join algorithm to execute. Queries from the second class, FastJoins, have higher memory allocations and lower response times. FastJoin queries use selection predicates with a selectivity of 25%; i.e. only a quarter of the input tuples are selected for the join[12]. Queries from the FastJoin class are allocated their maximum memory requirement [size of inner relation (286 MB)] and require only a single pass of the join algorithm to execute. Similar to the previous experiment, we compare the performance of a partitioned system, with 96 nodes dedicated to transaction processing and 32 nodes dedicated to query processing, with that of a non-partitioned system. The non-partitioned system uses the combined delay mechanism to control query interference and delays the queries to match the response times of each query class in the partitioned system. Figure 24 shows the performance of the two schemes.

As can be seen from the graphs, non-partitioning achieves the same query response time as the partitioning scheme and a significantly better transaction response time. The results also demonstrate the flexibility of the query delay mechanism. In the non-partitioned system, the average execution time of the joins in the FastJoin class is 60 s while the execution time of the joins from the SlowJoin class is 147 s. In order to match the corresponding response times in the partitioning system, the delay periods of the two classes are also changed accordingly: the FastJoin queries are delayed by 133 seconds while SlowJoin queries are delayed by 345 s. As a result of slowing down the queries, the transaction response times improve significantly (24%) compared to the partitioned system.

This experiment demonstrates that the delay mechanism can be used with even greater flexibility in a multi-class workload; a different delay period can be used to tune the performance of multiple classes in the workload. Therefore, for a multi-class workload, a system in which processors are not partitioned between different workload classes will perform significantly better than a partitioned system.

9 Related work

There has been a wealth of research in the area of data placement. Several SN database systems (Teradata DBC/1012 [Tera85], Tandem Non-Stop SQL [Tand88] and Gamma [DeWi90]) use full declustering as their data placement strategy. However, there have been several studies that have argued for partial declustering in data placement ([Cope88], [Padm92], [Rahm93a], [Rahm94]). As a result of these studies, partial declustering has also been considered in several other performance studies ([Ohmo90], [Ohmo91], [Pena92], [Baru93], [Frie94]). In this section, we examine each of the earlier declustering studies and compare their conclusions to our own. This is followed by a discussion of related work on data placement in the context of system architectures other than SN parallel database systems.

9.1 The case for partial declustering

9.1.1 Copeland et al.

Copeland et al. [Cope88] presented the data placement algorithm used in the Bubba parallel database system. In their scheme, two copies were maintained for each relation in the database. The base relation is called the direct copy. In addition, the system also maintained a replica, called the IF copy, for recovery purposes. Data placement in Bubba was simplified by choosing the same degree of declustering, called CDegDecl, for every relation. The CDegDecl nodes were split between the Direct and IF copy of the relation using the relative "heat", or access frequency, of the two copies. Note that splitting the nodes between the direct and IF copies implies partial declustering for each relation (since no relation can be declustered on all the nodes).The actual placement of relations was performed in two steps. In the first step, relations with the highest temperatures (heat per unit size) were cached in memory. In the second step, the remaining relations were placed in decreasing order of heat on the nodes with the least accumulated heat. The performance of the Bubba declustering algorithm was investigated using an analytical queueing model. It was shown that increasing the degree of declustering improves performance and load

[12] Note that, as mentioned in Sect. 4, a 100% selectivity was used for all other joins in the paper. The 25% selectivity was chosen for joins in the FastJoins class to ensure that their hash table can fit in memory.

balancing until the system is CPU-bottlenecked. In that case, increase in the degree of declustering degrades performance due to high startup, termination, and communication costs.

The results presented in [Cope88] have long been interpreted as showing that partial declustering is the preferred declustering technique. However, a closer look at the results can lead to very different conclusions. For instance, even though the analytical study in [Cope88] used only 5 MIPS processors, the processors become the bottleneck and performance degrades only when the degree of declustering is increased beyond 736. With faster processors, the degree of declustering could be increased further before performance degrades. In fact, the authors note in the paper that "as processor speeds continue to increase with respect to disk speeds in the coming decade. . . , the overheads associated with declustering will diminish." Since most parallel database machines are installed with fewer than 736 nodes and much faster processors, the results presented in [Cope88] can be interpreted as advocating full declustering for most database relations.

However, there are several drawbacks in the data placement algorithm presented in [Cope88]. The Bubba algorithm requires the presence of two copies (Direct and IF) for each relation. Also, it does not provide a mechanism for selecting the degree of declustering of relations – the degree of declustering is an input parameter for the placement algorithm. Moreover, the degree of declustering is identical for each relation (while the results presented in this paper show that the optimal degree of declustering varies with relation size). Also, the performance of the Bubba placement algorithm depends on the accuracy with which the access frequencies of relations can be predicted. Since database workloads are often so complex and dynamic that static workload analysis is not and the Bubba data placement algorithm cannot be used for such cases.

9.1.2 Padmanabhan

Padmanabhan [Padm92] studied data placement in a SN database system. He showed that the optimal degree of declustering should be different for each relation and that the degree of declustering and the placement of each relation should be performed by a single integrated algorithm in order to obtain the best possible performance. Data placement was shown to be NP complete and randomization techniques were used to develop data placement heuristics. An analytical model was used to show that partial declustering is superior to full declustering and that the randomized algorithms perform well for a variety of workloads. However, the conclusions of this study are based on examining the performance of very small database relations. In order to show that the results do not hold for larger relation sizes, we duplicated some of the results presented in [Padm92] for small relation sizes and then repeated the experiments with larger relation sizes.[13] Figure 25 shows the response time of an indexed scan of a 10 million tuple relation with a selectivity of 0.001% (only 100 tuples are selected).

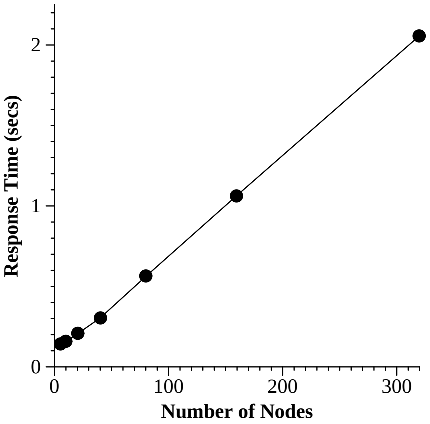

Fig. 25. Index selectivity = 0.001%. Comparison with [Padm92]: clustered index scan on a 10 million tuple relation

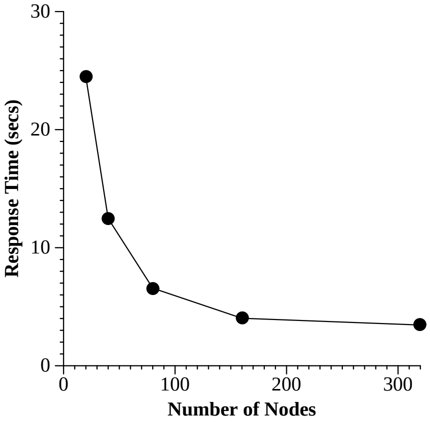

Fig. 26. Index Selectivity = 10%. Comparison with [Padm92]: clustered index scan on a 10 million tuple relation

The results, similar to the results in [Fig. 3.1, Padm92], show that the response time increases for all degrees of declustering. Note that these results agree with the results presented in Sect. 5.5 which show that the optimal degree of declustering when using a clustered index scan with x% selectivity is the same as the optimal degree of declustering for a file scan of a relation with $(R * x)$ tuples. Therefore, for a clustered index scan of a 10 million tuple relation with a selectivity of 0.001 %, the optimal degree of declustering is 1. These results were used in [Padm92] to show that partial declustering is desirable for declustering. However, as shown in Fig. 26, if the scan selectivity increases to 10%, full declustering is the best option for all system sizes. These experiments show that the results of [Padm92] hold only for very small query sizes and that full declustering is the best option for most non-trivial relation sizes.

9.1.3 Rahm and Marek

Rahm and Marek studied processor allocation in a SN database system [Rahm93a, Rahm94]. However, some of their experiments compared full and partial declustering. These results showed that partial declustering outperforms full declustering for both query-only workloads and mixed workloads containing update transactions and read-only que-

[13] The experimental parameters for this experiment were established in consultation with the author of [Padm92], Sriram Padmanabhan.

Table 7. Simulation parameter

Configuration		Database/workload	
Parameter	Value	Parameter	Value
Number of PEs	10–80	Relation A size	1M tuples
CPU speed	20 MIPS	Relation B size	250 000 tuples
Page size	8 KB	Tuple size	200 bytes
Avg. disk access time	15 ms	Selectivity	0.25/50%
Send message instructions	5000	Result size	625 tuples
Receive message instructions	10 000	Size of result tuples	400 bytes
Copy 8 KB message	5000	Access method	Clustered B + tree

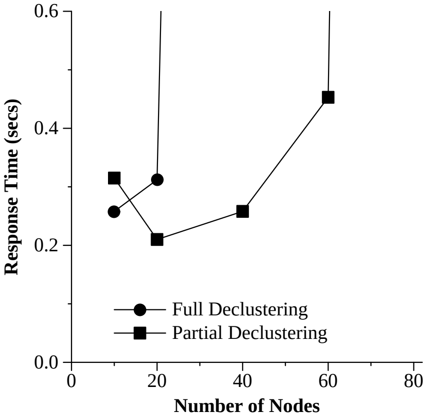

Fig. 27. Index selectivity = 0.25%. Comparison with [Rahm93a]

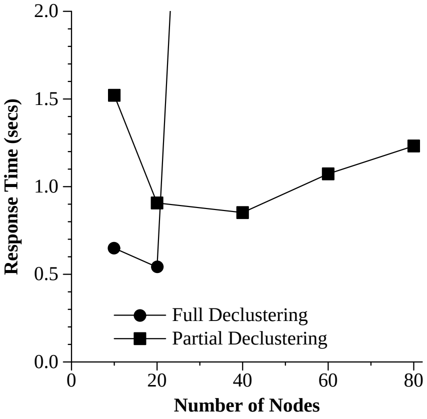

Fig. 28. Index selectivity = 1%. Comparison with [Rahm93a]

ries. However, these results (like those of [Padm92]) are based on workloads consisting of very small queries: binary joins where the input relations had 1 million and 250 000 tuples and the selectivity was only 0.25% – less than 2500 tuples per relation are selected for the join. In order to show that their results are not valid for larger relation sizes, we used our simulation environment to first duplicate the results of [Rahm93a] and then re-ran similar experiments with larger query sizes. The experimental parameters used for these experiments are taken from [Rahm93a] and are shown in Table 7.

The workload consists of multiple binary join queries which join relations A and B. The performance of full declustering is compared to a partial declustering strategy in which the relations are declustered across a disjoint set of disks in proportion to their size (i.e. A is declustered on 80% of the nodes and B is declustered on the other 20%).

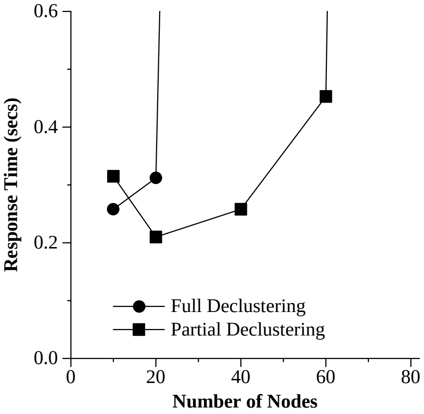

Fig. 29. Index selectivity = 10%. Comparison with [Rahm93a]

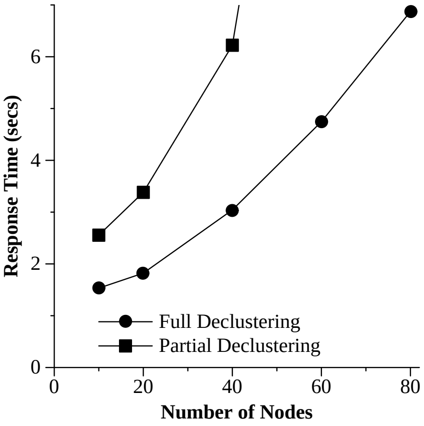

Fig. 30. Index selectivity = 50%. Comparison with [Rahm93a]

We examine a multi-user workload where the selectivity is varied from 0.025% to 50%. Figures 27–30 show the performance of the average query response time obtained with the full and partial declustering schemes for the various selectivities as the number of nodes in the system is increased from 10 to 80. In order to simulate a constant throughput per system node for a given selectivity (as in [Rahm93a]), the query arrival rates are adjusted for each system size. Figure 27 shows that at a selectivity of 0.025% (the corresponding figure in [Rahm93a] is Fig. 7), full declustering performs much worse than partial and the system saturates with full declustering once the number of nodes is increased beyond 20. A similar phenomenon is observed even for a selectivity of 1% (Fig. 28).

Rahm and Marek [Rahm93a] conclude from results similar to the ones shown in Figs. 27 and 28 that partial declustering is better than full declustering. However, the relative performance of partial and full declustering is very different if the number of tuples processed by the queries is increased. Figure 29 compares the performance of the two declustering schemes when the selectivity has been increased to 10% (25 000 and 100 000 tuples are selected from relation A and B, respectively). Now, partial declustering is better than full declustering only when the degree of declustering increases to 80, and there is less than 64 KB of data per node. If the selectivity is increased to 50%, Fig. 30 shows that full declustering out-performs partial declustering for all configurations. These experiments have shown that partial declustering is a good option only if the queries are extremely small (very low selectivities), and that the conclusions of [Rahm93a, Rahm93b] are not valid for larger-sized queries.

9.2 Other related work

In addition to SN systems, data placement has also been studied for other computing environments. The performance of full declustering was compared to a scheme that places entire relations on a single disk in [Livn87] for a centralized multi-disk file system. The results showed that full declustering improves performance except under high utilization. Data placement for file systems has also been studied in ([Wolf89],[Dowd92]). Their results are, however, not applicable for SN parallel database systems as they place whole copies of the files on single disks and do not use declustering.

Data placement in disk arrays was studied in [Weik91, Weik92]. The authors developed analytical formulas to decide the striping unit of files. Although the striping unit of a file is analogous to the degree of declustering of a relation in SN systems, the formulas are not directly applicable. While the degree of declustering in SN systems also determines the degree of intra-operator parallelism, such a correlation is absent in disk arrays. Consequently, only the average size of I/O requests to a file can be used determine the striping unit [Weik92] and intra-operator parallelism can be ignored.

Data placement in distributed database systems (DDBs) also deals with issues of partitioning and assignment ([Ceri84], [Oszu90]). However, due to the high cost of accessing remote data, the main emphasis of placement in DDBs is to enhance locality and not to increase parallelism. Finally, our study considered horizontal partitioning of database relations, which divides the tuples in a relation into disjoint sets. Several authors have also proposed vertical partitioning, which divides the attributes of a relation, for parallel database systems ([Ceri84],[Cope85],[Nava89]). However, the performance impact of vertical partitioning has not been carefully evaluated and no current parallel database system uses it. Therefore, our study also ignored vertical partitioning issues.

10 Conclusions

Data placement is an important issue in achieving high performance with SN parallel database systems. Intelligent data placement not only enhances performance by exploiting parallelism but also serves as a powerful tool for load balancing. This paper has explored two data placement issues in detail: determining the degree of declustering and the placement of declustered data on system nodes. The results demonstrate that, given the current state and future trends of hardware technology, the degree of declustering for relations can be increased without penalizing performance until there is only one disk cache-context worth of data per disk. This rule implies that except for very small relations, most relations in a database should be fully declustered. A performance analysis of three algorithms for placing relations that are too small to be fully declustered was also presented. A simple, workload-independent, round-robin algorithm was shown to perform well under a variety of conditions. These results, albeit simple, are contrary to the prevailing view of the research community. Numerous studies have shown that full declustering leads to reduced performance and this has lead many to believe that partial declustering is the correct solution. However, as shown in this study, recent changes in technology have significantly changed the performance tradeoffs. As a result, full declustering should now be used as the data placement strategy for parallel SN database systems. In addition to non-partitioned workloads, we also examined the effect of data placement schemes on partitioned configurations. The results indicate that full declustering can be used in conjunction with a dynamic delay scheme to achieve a variety of response times for workload classes. Further experiments showed that using such schemes leads to a very flexible system that can adapt successfully to workload changes.

In summary, the results in this paper conclusively demonstrate that full declustering is a viable strategy for placing relations in a SN parallel database system. Full declustering provides high parallelism and efficient load balancing. Also, mechanisms like query delay can be used in conjunction with full declustering to adapt the performance of the system to workload changes instead of resorting to the very expensive strategy of data reorganization.

The results of this paper have a significant impact on other query processing issues as well. Since startup, termination and communication costs are not a dominant factor in query processing, processor allocation issues can be simplified with most queries executing on all the nodes in the system. Similarly, low processing costs will affect the selection of the declustering strategy to map tuples to relation partitions. The various declustering strategies differ in the degree of parallelism used for each database operation. If processing costs are low, the effect of using different degrees of parallelism will be small. Therefore, the relative performance of different declustering algorithms may also be very similar. We also want to explore the effect of data placement on complex query processing. Complex-query scheduling strategies, like right-deep joins, which read multiple base relations simultaneously, may cause too much disk interference with full declustering. In such cases, left-deep scheduling, which reads only one relation at a time, will lead to less interference and may be a better strategy for scheduling complex queries. And finally, we want to validate the results of this simulation study on real parallel database systems.

Acknowledgements. The authors would like to thank Kurt Brown for helpful comments on an earlier draft of this paper.

References

[Bitt88] Bitton, D. and Gray, J., "Disk Shadowing", *Proc. VLDB Conf.*, Los Angeles, Calif, 1988.

[Bora90] Boral, H. et al., "Prototyping Bubba, A Highly Parallel Database System", *IEEE Transactions on Knowledge and Data Engineering*, 2(1), March 1990.

[Brow92] Brown, K., Carey, M., Dewitt, D., Mehta, M. and Naughton, J., "Resource Allocation and Scheduling for Mixed Database Workloads," *Computer Sciences Technical Report #1095*, Department of Computer Sciences, University of Wisconsin, Madison, July 1992.

[Brow93] Brown, K., Carey, M., and Livny, M., "Managing Memory to Meet Multiclass Workload Response Time Goals", *Proc. VLDB Conf*, Dublin, Ireland, August 1993.

[Brow94] Brown, K., Mehta, M., Carey, M. and Livny, M., "Towards Automated Performance Tuning for Complex Workloads", *Proc. VLDB Conf.*, Santiago, Chile, September 1994.

[Ceri84] Ceri, S. and Pelagatti, G., *Distributed Databases: Principles and Systems*, McGraw-Hill, New York, NY, 1984.

[Chen92a] Chen, Ming-Syan et al., "Using Segmented Right-Deep Trees for the Execution of Pipelined Hash Joins", *Proc. VLDB Conf.*, Vancouver, Canada, August 1992.

[Chen92b] Chen, Ming-Syan et al., "Scheduling and Processor Allocation for Parallel Execution of multi-join Queries", *Proc. 8th IEEE Data Engineering Conf.*, Phoenix, Ariz, Feb. 1992.

[Cope85] Copeland, G. and Khoshafian, S., "A Decomposition Storage Model", *Proc. ACM SIGMOD Conf.*, 1985.

[Cope88] Copeland, G. et al., "Data Placement in Bubba", *Proc. ACM SIGMOD Conf.*, Chicago, Ill, June 1988.

[DeWi84] DeWitt, D. et al., "Implementation Techniques for Main Memory Database Systems", *Proc. ACM SIGMOD Conf.*, Boston, Mass, June 1984.

[DeWi90] DeWitt, D. et al., "The Gamma Database Machine Project", *IEEE Transactions on Knowledge and Data Engineering*, 2(1), March 1990.

[DeWi92a] DeWitt, D. and Gray, J., "Parallel Database Systems: The Future of High Performance Database Systems", *CACM*, 35(6), June 1992.

[DeWi92b] DeWitt, D. et al., "Practical Skew Handling in Parallel Joins", *Proc. PDIS Conf.*, San Diego, Calif, January 1992.

[Dowd92] Dowdy, L.W. and Foster, D.V., "File Assignment in a Computer Network", *ACM Computer Surveys*, 14(2), 1982.

[Engl91] Englert, S., "Load Balancing Batch and Interactive Queries in a Highly Parallel Environment", *Proc. IEEE COMPCON Conf.*, San Francisco, Calif, February 1991.

[Falo93] Faloutsos, C. and Bhagwat, P., "Declustering Using Fractals", *Proc. PDIS Conf.*, San Diego, Calif, January 1993.

[Gerb85] Gerber, R. and DeWitt, D., "Multiprocessor Hash-Based Join Algorithms", *Proc. VLDB Conf.*, Stockholm, Sweden, August 1985.

[Ghan90] Ghandeharizadeh, S., "Physical Database Design in Multiprocessor Systems", *PhD Thesis*, University of Wisconsin-Madison, 1990.

[Ghan92] Ghandeharizadeh, S., DeWitt D. and Qureshi, W., "A Performance Analysis of Alternative Multi-Attribute Declustering Strategies", *Proc. ACM SIGMOD Conf.*, San Diego, Calif, June 1992.

[Grae89] Graefe, G., "Volcano: An extensible and parallel dataflow query processing system.", *Computer Science Technical Report*, Oregon Graduate Center, Beaverton, Ore, June 1989.

[Gray87] Gray, J. and Putzolu, F., "The 5 Minute Rule for Trading Memory for Disk Accesses and 10 Byte Rule for Trading Memory for CPU Time", *Proc. ACM SIGMOD Conf.*, San Francisco, Calif, May 1987.

[Haas90] Haas, L. et al., "Starburst Mid-Flight: As the Dust Clears", *IEEE Trans. on Knowledge and Data Eng.*, 2(1), March 1990.

[Hua90] Hua, K. A. and Lee, C., "An Adaptive Data Placement Scheme for Parallel Database Computer Systems", *Proc. VLDB Conf.*, Brisbane, Australia, 1990.

[Hua91] Hua, K. A. and Lee, C., "Handling Data Skew in Multiprocessor Database Computers using Partition Tuning", *Proc. VLDB Conf.*, Barcelona, Spain, September 1991.

[IBM93] IBM Corporation, Product Announcement for Disk Drives DFHS-31080, -32160, -34320, IBM Corporation, November 1993.

[Kits91] Kitsuregawa, M. and Ogawa, , "Bucket Spreading Parallel Hash: A new, robust, parallel hash-join method for data skew in the Super Database Computer (SCD)", *Proc. VLDB Conf.*, Brisbane, Australia, August 1991.

[Livn87] Livny, M., Khoshafian, S. and Boral, H., "Multi-Disk Management Algorithms", *Proc. ACM SIGMETRICS Conf.*, Alberta, Canada, May 1987.

[Meht93] Mehta, M. and DeWitt D., "Dynamic Memory Allocation for Multiple-Query Workloads", *Proc. VLDB Conf.*, Dublin, Ireland, August 1993.

[Meht94] Mehta, M., "Resource Allocation in Parallel Shared-Nothing Database Systems", *PhD. Thesis*, University of Wisconsin, 1994.

[Nava89] Navathe, S. and Ra, M., "Vertical Partitioning in Database Design: A Graphical Algorithm", *Proc. ACM SIGMOD Conf.*, Portland, Ore, June 1989.

[Ng91] Ng, R., Faloutsos, C. and Sellis, T., "Flexible Buffer Allocation based On Marginal Gains", *Proc. ACM SIGMOD Conf.*, Denver, Colo, May 1991.

[Omie91] Omiecinski, E., "Performance Analysis of a Load Balancing Hash-Join Algorithm for a Shared Memory Multiprocessor" *Proc. VLDB Conf.*, Barcelona, Spain, September 1991.

[Oszu90] Oszu, M. T. and Valduriez, P., *Principles of Distributed Database Management Systems*, Prentice Hall, 1990.

[Padm92] Padmanabhan, S., "Data Placement in Shared-Nothing Parallel Database Systems", *PhD. Thesis*, CSE-TR-135-92, University of Michigan.

[Para93] Release 1.1 Release Notes for the Paragon XP/S System, Intel Corporation, Beaverton, Ore, October 1993.

[Rahm93a] Rahm, E. and Marek, R., "Analysis of Dynamic Load Balancing Strategies for Parallel Shared Nothing Database Systems", *Proc. VLDB Conf.*, Dublin, Ireland, August 1993.

[Rahm93b] Rahm, E., "Parallel Query Processing in Shared-Disk Database Systems", *Proc. 5th International HPTS Workshop*, Asilomar, Calif, September 1993.

[Ries78] Ries, D. and Epstein, R., "Evaluation of Distribution Criteria for Distributed Database Systems", *UCB/ERL Technical Report M78/22*, UC Berkeley, May 1978.

[Schn90] Schneider, D. and DeWitt, D., "Tradeoffs in Processing Complex Join Queries via Hashing in Multiprocessor Database Machines", *Proc. VLDB Conf.*, Melbourne, Australia, August 1990.

[Schw90] Schwetman, H., CSIM Users' Guide, *MCC Technical Report No. ACT-126-90*, Microelectronics and Computer Technology Corp., Austin, Tex, March 1990.

[Seli93] Selinger, P., "Predictions and Challenges for Database Systems in the Year 2000", *Proc. VLDB Conf*, Dublin, Ireland, August 1993.

[Sell88] Sellis, T., "Multiple Query Optimization", *ACM TODS 13(1)*, March 1988.

[Shat93] Shatdal, A. and Naughton, J., "Using Shared Virtual Memory for Parallel Join Processing,", *Proc. ACM SIGMOD Conf.*, Washington, DC, May 1993.

[Stel93] Stellwagen, R. (NCR Corporation), Personal Communication, 1993.

[Tand88] Tandem Performance Group, "A benchmark of non-stop SQL on the debit credit transaction", *Proc. ACM SIGMOD Conf.*, Chicago, Ill, June 1988.

[Tera85] Teradata Corp., "DBC/1012 Data Base Computer System Manual", *Teradata Corp. Document No. C10-0001-02*, Release 2.0, November 1985.

[Walt91] Walton, C., et al., "A Taxonomy and Performance Model of Data Skew in Parallel Joins", *Proc. VLDB Conf.*, Barcelona, Spain, September 1991.

[Weik91] Weikum, G., Zabback, P., Scheurmann, P., "Dynamic File Allocation in disk Arrays", *Proc. ACM SIGMOD Conf.*, Denver, CO, May 1991.

[Weik92] Weikum, G. and Zabback,, "Tuning of Striping Units in Disk-Array-Based File Systems", *Proc. RIDE-TQP Workshop*, Phoenix, Ariz, February 1992.

[Wils92] Wilschut, A., Flokstra, J., Apers, P., "Parallelism in a main-memory DBMS: The performance of PRISMA/DB" *Proc. VLDB Conf.*, Vancouver, Canada, August 1992.

[Wolf89] Wolf, J., "The Placement Allocation Program: A Practical Solution to the Disk File Assignment Problem", *Proc. ACM SIGMETRICS Conf.*, 1989.

[Wolf90] Wolf, J. et al., "An effective algorithm for parallelizing hash joins in the presence of data skew", *Proc. 7th IEEE Data Engineering Conf.*, Kobe, Japan, April 1991.

[Youn92] Young, H. and Swami, A., "A Family of Round-Robin Partitioned Parallel External Sort Algorithms", *Research Report RJ 9104*, IBM Research Division, November 1992.

[Yu93] Yu, P. and Cornell., D., "Buffer Management Based on Return on Consumption in a Multi-Query Environment", *VLDB Journal*, 2(1), January 1993.

[Zipf49] Zipf., G. K., *Human Behavior and the Principle of Least Effort*, Addison Wesley, Reading, Mass, 1949.