Hybrid-Range Partitioning Strategy: A New Declustering Strategy for Multiprocessor Database Machines

Shahram Ghandeharizadeh
David J. DeWitt

Computer Sciences Department
University of Wisconsin - Madison
Madison, WI 53706

ABSTRACT — In shared-nothing multiprocessor database machines, the relational operators that form a query are executed on the processors where the relations they reference are stored. In general, as the number of processors over which a relation is declustered is increased, the execution time for the query is decreased because more processors are used, each of which has to process fewer tuples. However, for some queries increasing the degree of declustering actually increases the query's response time as the result of increased overhead for query startup, communication, and termination. In general, the declustering strategy selected for a relation can have a significant impact on the overall performance of the system. This paper presents the **hybrid-range** partitioning strategy, a new declustering strategy for multiprocessor database machines. In addition to describing its characteristics and operation, its performance is compared to that of the current partitioning strategies provided by the Gamma database machine.

1. Introduction

In shared-nothing [STON86] multiprocessor database machines, the use of parallelism for all types of queries is not necessarily a good idea [COPE88]. In particular, when the overhead associated with using parallelism to execute a particular query begins to constitute a significant fraction of the execution time of the query, the overall throughput of the system will actually be higher than if fewer processors are used. On the other hand for those queries that consume significant CPU and I/O resources, the overhead associated with using parallelism is not significant. In this case, additional processors can be used to improve the response time without adversely affecting the throughput of the system. In addition, the sequential execution of such queries on a single processor may result in the formation of hot spots in a multiuser environment. Clearly, the strategy used to decluster each relation in such an environment can have a significant impact on the overall performance of the system.

Gamma [DEWI90] currently provides the database administrator with three alternative declustering strategies when creating a relation: round-robin, hashed, and range partitioned. With the first strategy, tuples are distributed in a round-robin fashion among the disk drives. If the hashed partitioning strategy is selected, a randomizing function is applied to the key attribute of each tuple to select a storage unit. In the third strategy, the user specifies a range of key values for each processor. The partitioning information for each relation is stored in the database catalog. For range and hash partitioned relations, the name of the partitioning attribute is kept and, in the case of range partitioned relations, the range of values of the partitioning attribute for each processor (termed a **range table**) is also stored.

With the round-robin declustering strategy, the degree of intra-query parallelism is equivalent to the number of disks that the relation is declustered across. The hash and range declustering strategies both provide enough information so that the execution of exact match queries on the partitioning attribute can be localized to a single node. However, in the case of a selection with a range predicate on the partitioning attribute (e.g., 10 < employee.age < 30), while the range declustering mechanism can localize the execution of the query to only those processors that contain relevant tuples, the hash declustering strategy must direct such queries to all the processors (like the round-robin declustering strategy). If the execution time of such range queries is small, using only 1-2 processors is optimal. However, if the query consumes significant CPU and I/O resources, its response time will be higher with the range declustering strategy than with the hash and round-robin strategies.

In this paper we describe a new declustering strategy termed the **hybrid-range partitioning strategy (HRPS)**. Unlike the other declustering strategies, the HRPS utilizes the characteristics of the queries that access a relation to obtain the appropriate degree of intra-query parallelism. In particular, the HRPS strikes a compromise between the sequential execution paradigm of the range declustering strategy and the load balancing/intra-query parallelism characteristics of the hash and round-robin declustering strategies.

With the hybrid-range declustering strategy a relation is declustered into many small logical fragments such that each fragment contains a distinct range of the partitioning attribute value (the number of fragments is independent of the number of

This research was partially supported by the Defense Advanced Research Projects Agency under contract N00039-86-C-0578, by the National Science Foundation under grant DCR-8512862, by a Digital Equipment Corporation External Research Grant, and by a research grant from the Tandem Computer Corporation.

Proceedings of the 16th VLDB Conference
Brisbane, Australia 1990

processors in the configuration). As we will demonstrate below, this allows the optimizer to execute range queries with minimal resource requirements on a single processor while directing CPU and/or I/O intensive queries to a large number of processors. A central issue to the successful application of the HRPS, is determining how many tuples each fragment of a relation should contain. In Section 3, we develop a methodology to answer this question by taking into account the following four factors:

(1) The resource requirements of the queries accessing the relation,

(2) The processing capability of the system,

(3) The overhead of using each additional processor to execute a query

(4) The cost of searching the range table constructed by the hybrid-range declustering strategy.

The last factor is very important because it allows the hybrid-range declustering strategy to degenerate into the range declustering strategy for those queries with such small execution times that the time to search the range table becomes significant (e.g., a Debit-Credit transaction).

The remainder of this paper is organized as follows. In Section 2 we describe related previous work . Section 3 describes the hybrid-range partitioning strategy. The description of the workload and the research vehicle used for evaluating the performance of this partitioning strategy are presented in Sections 4 and 5, respectively. The performance of the hybrid-range partitioning strategy is compared with the range and hash partitioning strategies in Section 6. Section 7 enumerates the other advantages of the hybrid-range partitioning strategy. Our conclusions appear in Section 8.

2. Related Work

There have been a number of earlier studies of the problem of file distribution in multiprocessor database machines. Some have proposed new declustering strategies [DU82, KIM88, COPE88, PRAM89], while others have analyzed the impact of alternative declustering strategies [LIVN87, GHAN90]. Studies that are concerned with the multi-disk architecture [DU82, LIVN87] are different and not directly relevant to this study. In order to make this distinction clear, we use the two dimensional file system of the XPRS shared-memory multiprocessor database management system [STON88] as an example. XPRS uses a disk array for mass storage (based on the RAID design [PATT88]). It supports intra-query parallelism by fragmenting a single relation into multiple files. Like range declustering in Gamma, each relation is fragmented using a distribution criteria [STON88], e.g.:

EMP where age < 20	to file 1
EMP where age >= 20 and age <= 40	to file 2
EMP where age > 40	to file 3

Furthermore, each file is organized as a number of extents. In traditional file systems, an extent corresponds to a sequential number of blocks on a disk drive. In XPRS, each extent is two dimensional: one dimension corresponds to the number of disks that contain an extent (W_i) while the second dimension corresponds to the number of tracks on each disk (S_i). The choice of W_i and S_i is suggested by the application software [STON88].

In an environment similar to that of XPRS, the hybrid-range declustering strategy would determine the appropriate number of files a relation must be fragmented into and not the values of S_i or W_i since the hybrid-range declustering strategy attempts to determine the appropriate degree of intra-query parallelism for the set of queries that access the relation. Determining the values of W_i and S_i is a separate problem with a different set of constraints that is beyond the scope of this paper.

[KIM88, PRAM89] presented an optimal file declustering strategy for partial match retrieval queries. The major contribution of this study was the development of a new approach termed FX distribution which maximizes data access concurrency in the presence of partial match queries. A limitation of FX distribution is that it does not provide support for range queries.

[COPE88] examined the problem of data placement in Bubba [ALEX88, BORA88]. This study introduced concepts such as Heat and Temperature which are crucial to understanding the issues relevant to the problem of data placement. However, the major thrust of this study was to maximize the throughput of the system. In order to achieve this objective, this study focused on two issues: 1) balancing the load across the nodes in the environment, and 2) whether to place the data on disk or cache it permanently in memory. Due to the complexity of the problem (NP-complete), a heuristic approach was used to solve the problem. This study did not consider the impact of alternative declustering strategies on the throughput of the system. In addition, the algorithms presented in [COPE88] do not attempt to minimize the response time of a transaction.

This paper differs from all of these earlier efforts in that it describes a new declustering strategy that utilizes the characteristics of the queries that access a relation to decluster it such that the appropriate degree of intra-query parallelism is obtained for the queries in the workload.

3. The Hybrid-Range Partitioning Strategy (HRPS)

The HRPS declusters a relation into fragments based on the following criteria: 1) each fragment contains approximately *FC* tuples and 2) each fragment contains a unique range of values of the partitioning attribute. The variable *FC* is determined based on the processing capability of the system and the resource requirements of the queries that access the relation (rather than the number of processors in the configuration). This variable is described further in Section 3.1.

By declustering a relation using the HRPS, the number of processors that participate in the execution of the queries in the workload is optimized, resulting in a lower average response time and a higher average throughput when compared with the previous declustering strategies. The HRPS localizes the execution of those queries that have minimal resource requirements to a few processors while directing queries with high resource requirements to a large number of processors. Furthermore, it provides effective support for small relations and relations with a non-uniform distribution of the partitioning attribute values. A major underlying assumption of this partitioning strategy is that the selection operators which access the database retrieve and process the selected tuples using either a range predicate or an equality predicate.

Below, we will expand on this brief description and address the issues involved. We will describe the HRPS in the context of a mixed workload consisting of n selection queries[1]. For each query Q_i, the workload defines the CPU processing time (CPU_i), the Disk processing Time ($Disk_i$), and the Network Processing time (Net_i) of that query. Observe that these times are determined based on the resource requirements of each individual query and the processing capability of the system. Each query retrieves and processes ($TuplesPerQ_i$) tuples from the database. Furthermore, we assume that the workload defines the frequency of occurrence of each query ($FreqQ_i$).

Rather than describing the HRPS with respect to each query in the workload, we define an average query (Q_{Ave}) that is representative of all the queries in the workload. The CPU, disk and network processing quanta for this query are:

$$CPU_{Ave} = \sum_{i=0}^{n}(CPU_i * FreqQ_i)$$

$$Disk_{Ave} = \sum_{i=0}^{n}(Disk_i * FreqQ_i)$$

$$Net_{Ave} = \sum_{i=0}^{n}(Net_i * FreqQ_i)$$

$$TuplesPerQ_{Ave} = \sum_{i=0}^{n}(TuplesPerQ_i * FreqQ_i)$$

We will use this query throughout our discussion. In Section 3.3, we will expand the presentation of HRPS to include each individual query within the workload.

3.1. Computing the Number of Processors to Decluster a Relation Across

The three principal resources consumed during the execution of Q_{Ave} are: 1) CPU, 2) disk I/O , and 3) communications. The amount of each is dependent on the processing capability of the system and the resource requirements of the query. Assume that a single processor cannot overlap the use of two resources for an individual query[2]. Thus, the execution time of Q_{Ave} on a single processor in a single user environment is:

$$ExecutionTime = CPU_{Ave} + Disk_{Ave} + Net_{Ave} \quad (1)$$

As additional processors are used to execute the query, the response time of the query decreases. However, the overhead associated with using an additional processor must be incurred (we term this variable CP). This overhead is primarily in the form of additional messages to control the execution of the query on additional processors and is a function of the number of processors used.[3] For the purposes of this paper we have assumed this overhead to be a linear function of the number of processors since this is the case for Gamma.

The response time (RT) of a query in terms of the number of processors (termed M) used to execute it can be defined as follows:[4]

$$RT(M) = \frac{CPU_{Ave} + Disk_{Ave} + Net_{Ave}}{M} + M * CP \quad (2)$$

The first goal in declustering a relation using the HRPS is to decluster a relation across M processors in order to minimize the response time for the query. Setting the first derivative of the function $RT(M)$ to zero one can solve for desired value of M:

$$M = \sqrt{\frac{CPU_{Ave} + Disk_{Ave} + Net_{Ave}}{CP}} \quad (3)$$

If the relation is declustered across M processors (where M is rounded to the nearest integer), the response time for Q_{Ave} will be minimized.

While partitioning a relation across M processors minimizes the response time for Q_{Ave} in a single user environment, it does not necessarily maximize the throughput of the system in a multiuser environment. For example, assume that $M = 1$ (i.e., that the resource requirements of the queries in the workload are minimal) and that the selection predicate of each query is applied to the partitioning attribute. Consider also the performance of the system when the relation is declustered across N processors (say $N = 5$) using the range partitioning strategy. The range partitioning strategy, by its nature, will direct the queries in the workload to a single processor most of the time. Consequently, the average response time of query in a single user environment is almost the same for both partitioning strategies. However, in a multiuser environment, the range partitioning strategy will distribute the concurrently executing queries among all five processors, while

[1] In this paper we concentrate on how to obtain the appropriate degree of intra-query parallelism for a selection operator using an index. We do not consider selection operators that sequentially scan a file because [GHAN90] demonstrates that the range declustering strategy is the best partitioning strategy for this query type. The major reason for considering only the selection operator is that it is found in almost all query plans. Furthermore, it has a significant impact on the performance of a complex query as, if the declustering strategy cannot provide the appropriate degree of intra-query parallelism for a selection operator, the performance and degree of intra-query parallelism of the other operators may be severely limited. This is especially true for database machines that data flow techniques to execute complex queries. There is nothing restricting the model from being extended to correspond to a more complex query plan.

[2] The justification for this assumption is provided in Section 3.3.

[3] The algorithm used to schedule and commit a multi-processor query defines this function. For example, if a tree activation protocol is used, this function will be logarithmic in the number of processors participating in the execution of the query. (where the base of the log is the branching factor of the tree).

[4] The linear speedup results presented in [DEWI90] justify this assumption.

the HRPS will always direct them to a single processor. Thus, the throughput of the system will almost certainly be higher with the range partitioning strategy.

How can the throughput of the system be maximized? The solution is to change the interpretation of M. Instead of M representing the number of processors over which a relation should be declustered, M is used instead to represent the number of processors that should participate in the execution of Q_{Ave}. Since Q_{Ave} processes $TuplesPerQ_{Ave}$ tuples, each fragment of the relation should contain $FC = \frac{TuplesPerQ_{Ave}}{M}$ tuples. Furthermore, each fragment must contain a distinct, non-overlapping range of the partitioning attribute value.

3.2. Creation of the Fragments and Assignment of the Fragments to Processors

In order to guarantee that each fragment of a relation contains a distinct range of values of the partitioning attribute, the relation must first be sorted on the partitioning attribute. The relation can then be declustered such that each fragment contains approximately FC tuples. Finally, the fragments are distributed among the processors in a round-robin fashion, insuring that M adjacent fragments will be assigned to different processors (unless N, the number of processors, is less than M). At each processor all the fragments of a particular relation are stored in the same physical file. The assignment of fragments to processors is maintained in a one dimensional directory termed a range table. This simple heuristic results in the participation of at least M processors and at most $M + 1$ processors in the execution of the query

Since $\frac{Cardinality(Rel)}{FC}$ may be greater than N, each processor may contain more than one fragment of the relation. While this may appear both wasteful and/or unnecessary, it is actually a desirable property since it insures the participation of an optimal number of processors in the execution of Q_{Ave}.

In order to contrast the HRPS with the other partitioning strategies, consider the following example. Assume a 10,000 tuple relation with unique values for the partitioning attribute ranging from 0 to 9,999 and that the "average" query accessing this relation uses a range predicate on the partitioning attribute to retrieve and process 500 tuples ($TuplesPerQ = 500$). Also, assume that the optimal performance is achieved when 5 processors are used ($M = 5$). Thus, the cardinality of each fragment is 100 tuples ($FC = \frac{TuplesPerQ}{M} = 100$) and the relation will be partitioned into 100 fragments. Finally, assume that the alternative partitioning strategies decluster the relation across all the processors.

First, consider the case where N, the number of processors is equal to 5 (Figure 1.a). Since the HRPS assigns the fragments of the relation in a round-robin fashion among the processors, the query will overlap either 5 or 6 fragments. In either case, all the processors will be used to execute the query. The hash partitioning strategy will also use all N processors since it cannot localize the execution of a range query. The range partitioning strategy will also decluster the relation into 5 fragments. Since the range of a query falls within a range of a single fragment most of the time and overlaps the range of two fragments some of the time, the query will be directed to either 1 or 2 processors.

When N, the number of processors is less than M (see Figure 1.b), the HRPS will still use all N processors in the execution of the query because it enforces the constraint that the M adjacent fragments be assigned to different processors whenever possible. Conversely, the range partitioning strategy in this case declusters the relation into 2 (i.e., N) fragments, significantly increasing the probability of a query being directed to only one processor.

In the final case, $N > M$ (Figure 1.c), the HRPS will distribute the 100 fragments of the relation across all N processors in order to insure that all available resources are used in order to maximize the throughput of the system when executing multiple queries concurrently. However, since the range of a query will overlap only 5 or 6 fragments, each individual query is localized to almost the optimal number of processors. Conversely, the hash partitioning strategy will send the query to all N processors, incurring the startup, communication, and termination overheads associated with executing the query on more processors than absolutely necessary. The range partitioning strategy will again execute the query on only 1 or 2 processors, again using less than the optimal number of processors.

3.3. Discussion

One simplifying assumption that we have made is that a query does not use multiple resources on a single node simultaneously. This is not a realistic assumption. For example, Gamma uses a read-ahead process to overlap disk and CPU operations. Our justification for this simplification is that in a multiuser environment, the probability of overlap within a single query is fairly low since the total number of resources is relatively small compared to the multiprogramming level. Thus, while one query is consuming one of the resources (e.g. the CPU), it is highly likely that other queries will be consuming the other resources.

For each relation, the HRPS creates a one dimensional directory that specifies the range of each fragment and the mapping of fragments to processors. In the worst case, the FC of each fragment will equal one (i.e., the number of entries in the directory will equal the cardinality of the relation). This might be reasonable if the resource requirements of the query are extremely high (e.g., image processing). However, if they are very low, the overhead of searching the one dimensional directory might become a significant fraction of the query's execution time. Thus, this overhead must be considered when determining the size of each fragment. The number of entries in the one dimensional HRPS directory is $\frac{M * Cardinality(Rel)}{TuplesPerQ}$. If a lookup in this directory is performed using a linear search, the response time of the query is defined by the following equation:

$$RT(M) = \frac{CPU + Disk + Net}{M} + M * CP + \frac{M * Cardinality(Rel) * CS}{TuplesPerQ}$$

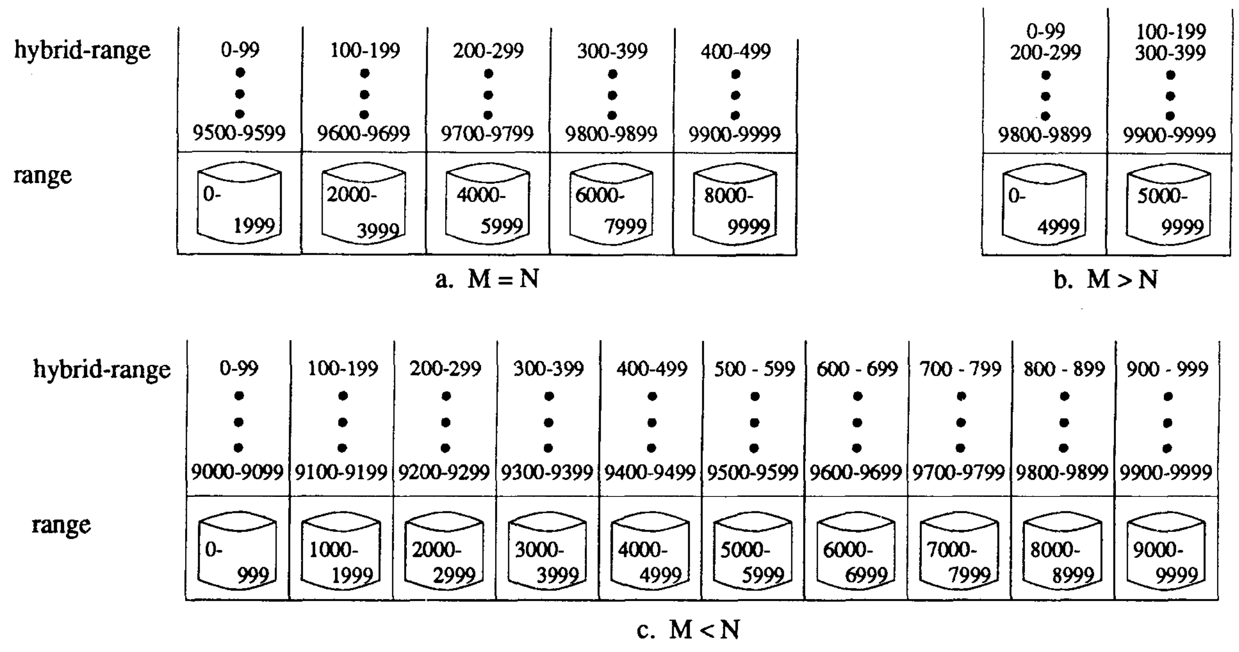

Figure 1

where CS represents the overhead associated with searching a single entry in the directory. Hence, the number of processors used to execute the query is specified by the following formula:

$$M = \sqrt{\frac{CPU + Disk + Net}{CP + \frac{Cardinality\,(Rel) * CS}{TuplesPerQ}}} \qquad (4)$$

If a binary search algorithm is used instead to process accesses to the HRPS directory, M becomes:

$$M = \frac{-\frac{CS}{\ln 2} + \sqrt{(\frac{CS}{\ln 2})^2 + (4 * CP * (CPU + Disk + Net))}}{2 * CP}$$

Thus far, we have described how the HRPS declusters a relation with respect to the resource requirements of Q_{Avg}. We now expand our discussion and consider the n individual queries that constitute the full workload for a relation. In this case, the response time $RT(M)$ becomes:

$$RT(M) = \sum_{i=0}^{n} (\frac{CPU_i + Disk_i + Net_i}{M} + M * CP) * FreqQ_i \qquad (5)$$

$$\text{Thus, } M = \sqrt{\frac{\sum_{i=0}^{n} (CPU_i + Disk_i + Net_i) * FreqQ_i}{CP}}$$

and the FC for each fragment is:

$$\frac{\sum_{i=0}^{n} (TuplesPerQ_i * FreqQ_i)}{M} \qquad (6)$$

In Section 6, we will compare the performance of the HRPS with that of the range and hash partitioning strategies. First, we will describe the workload and the research vehicle used for this performance analysis.

4. Workload Definition

We utilized a multiuser workload to evaluate the performance of the hybrid-range partitioning strategy since the response time of a query and the processing effort of the system required to execute the query with each of the alternative partitioning strategies are not necessarily correlated. For example, consider the execution of a 1% range selection query that is executed using a clustered index and whose predicate is applied to the partitioning attribute. While the hybrid-range partitioning strategy directs the query to a subset of the processors, the hash partitioning strategy directs the query to all the processors. Consequently, more resources will be consumed when the hash partitioning strategy is used than when the hybrid-range partitioning strategy is employed. However, the response time of the query will almost certainly be identical for both declustering strategies since the overhead of the extra control messages is small compared to the overall execution time of the query. In such a situation, the multiuser throughput of the system will reveal the tradeoffs associated with the different execution paradigms.

For our performance evaluation, we used a database consisting of 10 relations. Each relation was based on the Wisconsin Benchmark relations [BITT83] but contained one million 208 byte tuples. A uniform distribution of access to the different relations was used.

We choose three query types for this performance evaluation. In order to justify their selection consider Figure 2. In this figure, space A represents the domain of all possible queries. The region marked "Range" represents the region in space A for which the range partitioning strategy provides the best response time. These queries primarily consist of queries whose execution requires minimal CPU and I/O resources since the range partitioning strategy localizes their execution to a single processor and avoids the overhead associated with a multi-processor query.

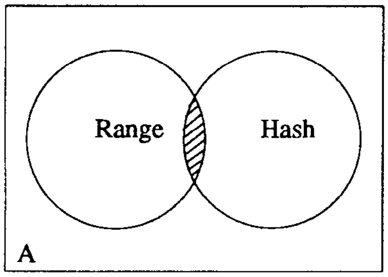

Figure 2

The area marked "Hash" represents the query types for which the hash partitioning strategy provides the best response time possible. Ignoring the shaded area for now, this region consists of queries with high resource requirements since the hash partitioning strategy directs such queries to all the processors, thus utilizing intra-query parallelism effectively to obtain the best possible response time.

These two regions (again ignoring the shaded area) are disjoint since the range partitioning strategy does not perform as well as the hash partitioning strategy for queries with high resource requirements and, similarly, the hash partitioning strategy does not perform as well as the range partitioning strategy for queries with minimal resource requirements.

The shaded region represents those selection queries with an equality predicate on the partitioning attribute that have minimal resource requirements. Both the range and hash partitioning strategies can localize the execution of such queries to a single processor.

In order to demonstrate that the HRPS is generally a better declustering strategy, we must show that it covers a region in space A that largely subsumes both regions marked "Range" and "Hash". Note that we do not have to consider all the different queries that might occur in either the "Range" or the "Hash" region in order to achieve this objective. Rather, we must demonstrate that the HRPS provides approximately the same level of performance as:

- the range declustering strategy for queries with low resource requirements
- the hash partitioning strategy for queries with high resource requirements
- both partitioning strategies for queries with an equality predicate and minimal resource requirements.

Finally, in those cases where the mix of queries have conflicting partitioning requirements, we must demonstrate that the HRPS provides better performance than both the hash and range partitioning strategies.

For a query with minimal resource requirements, we chose a 0.001% range selection on a 1 million tuple relation using a clustered index. This query retrieves and processes 10 tuples. Its single processor, single user execution time is 0.08 seconds on Gamma. For a query with high CPU and I/O requirements, we selected a 10% range selection on a 1 million tuple relation using a clustered index. This query retrieves and processes 100,000 tuples and has an execution time of 54.33 seconds. The third query type is essentially redundant since it is a special case of the first class of queries (0.001% selection query). As long as the hybrid-range partitioning strategy performs as well or better than the range partitioning strategy for the 0.001% range selection query, it has automatically satisfied the execution requirements of queries with an equality predicate and minimal resource requirements.

In order to demonstrate the superiority of the HRPS in those cases where the mix of queries have conflicting partitioning requirements, we chose a workload consisting of a mix of the 0.001% and 10% selection queries. Note that while the execution of the 0.001% selection must be localized to a single processor to minimize its execution time, the 10% selection should be executed by all the processors. Two cases were considered. In the first, we used an equal mix of both query types and varied the multiprogramming level. In the second, we fixed the multiprogramming level at 50 and varied the mix of the two query types. As we will demonstrate in Section 6, the HRPS provides superior performance in such situations.

5. Overview of the Gamma Database Machine

In this section, we present a brief overview of the Gamma database machine.[5] Gamma currently runs on a 32 processor iPSC/2 Intel hypercube [INTE88]. Each processor is configured with an Intel 80386 processor, 8 megabytes of memory, and a 330 megabyte MAXTOR 4380 (5 1/4") disk drive. Gamma is built upon a custom operating system that provides lightweight processes with shared memory. File services in Gamma are based on the Wisconsin Storage System (WiSS) [CHOU85]. These services include structured sequential files, B^+ tree indices, byte stream files as in UNIX, long data items, a sort utility, a scan mechanism, and concurrency control based on file and page locking.

Each disk drive has an embedded SCSI controller which provides a 45K byte RAM buffer that acts as a disk cache on read operations. In a single user environment, the SCSI cache speeds

[5] For a complete description of the Gamma database machine see [DEWI90].

up the execution of sequential scan queries by almost a factor of two. However, in a multiuser environment, its effect is minimal because the probability of two disk successive accesses being sequential is quite low. Furthermore, a disk request that results in a seek completely invalidates the contents of the cache. In the following section, we will repeatedly see the impact of this cache on the performance of the different selection queries.

For the performance evaluation presented in the following section, a 24 processor instantiation of Gamma was used. The remaining 8 processors in the hypercube were used to simulate users submitting queries.[6] Each node in Gamma was configured to use 8K byte disk pages and a buffer pool of 2 megabytes. The disk scheduler uses an elevator algorithm [TEOR72]. The buffer pool replacement policy is LRU.

6. Performance of Alternative Declustering Strategies

In this section, we evaluate the performance of the alternative declustering strategies using the workload described in Section 4. The alternative partitioning strategies declustered the relations across all the processors. The range and hash partitioning strategies distributed the tuples of each relation uniformly across the 24 processors, while the HRPS resulted in an approximately uniform distribution of the tuples across the processors. Before proceeding to the performance evaluation itself, we first must describe the execution paradigm for each of the alternative partitioning strategies.

In Figure 3, the BM process acts as a terminal generating queries which it submits to Gamma for execution. For each new query, the BM process randomly selects one of the 10 relations in the database and randomly generates a predicate for the selection query (both using a uniform random number generator). As described in Section 1, the Gamma query optimizer utilizes the information provided by the range, hash, and hybrid-range partitioning strategies in order to limit the number of processors to which each query is sent whenever possible.

As shown in Figure 3.b, when a query must be executed by multiple processors, it is first sent to a Query Manager (QM) process which assumes responsibility for its execution. The QM process sends the query to each processor that the query optimizer has indicated should participate in its execution. When each processor finishes executing the query, it sends a "query done" message back to the QM process. If the QM receives a "query done" message from each processor, it sends a "commit" message to each processor and closes all of the communication ports. Finally, it notifies the process that submitted the query of its successful execution. If, on the other hand, the query is aborted at a processor (generally because of a concurrency control deadlock) that processor sends an "abort" message to the QM. The QM, in turn, sends an "abort" message to each participating processor and then notifies the process that submitted the query. On the other

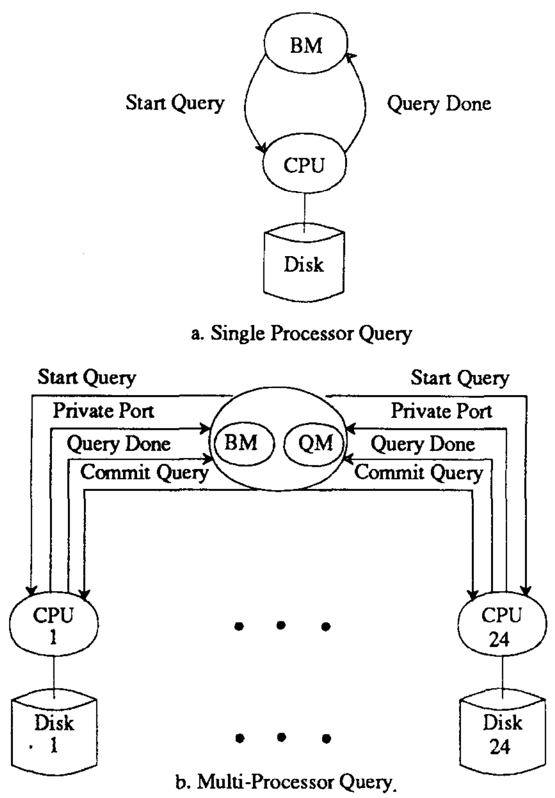

Figure 3

hand, queries that execute on only a single processor are sent directly to the proper processor for execution, bypassing the QM process (see Figure 3.a). As illustrated by Figure 3, significantly fewer messages are required to control the execution of a single processor query, and as we will see, this difference has a significant effect on the performance of the alternative partitioning strategies.

The benchmark process (BM) and the query manager process (QM) are two independent entities that are generally located on different processors, but they were placed on the same processor for these experiments. In order to simulate multiple concurrently executing users, one BM and one QM process was employed for each user. Eight processors were used for running the BM and QM processes in order to avoid a bottleneck from forming. For all the experiments presented below, we ensured that the CPU utilization of each of these 8 processors was less than 100%.

We measured *CostOfPart*, the overhead of using each additional processor to execute a query, to be 26 milliseconds in Gamma. Since operators are scheduled and terminated sequentially, the total overhead is a linear function of the number of participating processors. For the range and the hybrid-range partitioning strategies, the optimizer utilizes a binary search algorithm to search the range-table to determine which processors should participate in the execution of a query. We measured the overhead of searching the range table and initializing the query packet (i.e., CostOfSearch) to be 0.243 milliseconds per range table

[6] Communications between the hypercube and the Unix machine to which it is attached is too slow to allow a significant workload to be placed from the outside.

entry. These parameters are used by the HRPS to determine the number of fragments for a given relation and the cardinality of each fragment.

In the following sections, when we refer to the performance of a specific partitioning strategy for a particular type of query, we are implying that the selection predicate of the query is applied to the partitioning attribute.

6.1. 0.001% Selection Using a Clustered Index

With a clustered index, the order of the values in the B-tree corresponds to the order of the data records in the relation. Two types of disk requests are made by the range selection queries that use a clustered index structure: random and sequential. The traversal of the B-tree to locate the upper and lower limits of the query with respect to the actual data records is random, while the retrieval of data records between the lower and upper limit markings is sequential. In Figure 4, the throughput as a function of the multiprogramming level of a 0.001% selection query using a clustered index is presented for each of the alternative declustering strategies.

The execution time of this query using a single processor is 0.08 seconds. Using equation (4) (and the values of CostOfSearch and CostOfPart from the previous section), the value of M for this query was calculated to be 0.057 and hence only a single processor should be used. The HRPS will decluster each million tuple relation into 5,700 fragments with approximately 175 tuples per fragment. Since there are more fragments than processors, the fragments are distributed in a round-robin fashion among the processors.

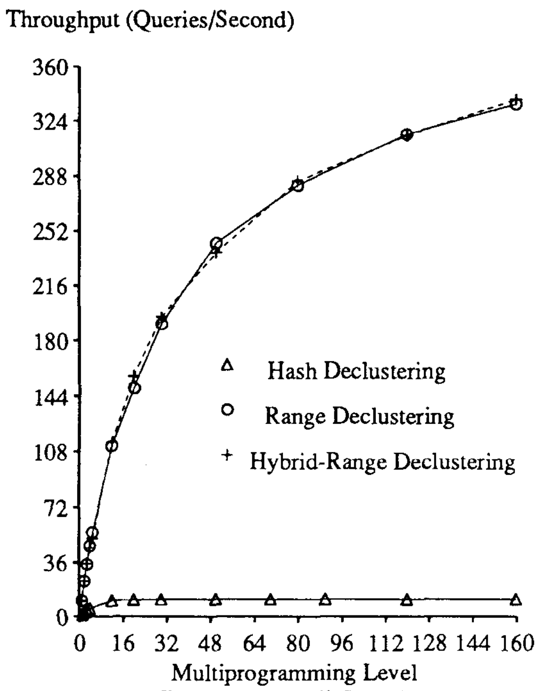

Figure 4: 0.001% Selection

While the optimizer has enough information for the hybrid-range and range declustering strategies to localize the execution of this query to 1 or 2 processors, since the query involves a range predicate, the hash partitioning strategy must direct this query to all the processors. Consequently, at a multiprogramming level of one, the throughput with the range and hybrid-range declustering strategies is almost eight times higher than that of the hash partitioning strategy since the hash partitioning strategy incurs the overhead associated with a multi-processor query. At a multiprogramming level of 20, the CPU of each individual processor becomes 100% utilized causing the throughput of the system to level off.

With the range and hybrid-range partitioning strategies, one might have expected the throughput to increase linearly from a multiprogramming level of one to twenty four. However, beginning around a multiprogramming level of 12, the increase in throughput is no longer a linear function of the multiprogramming level. This is primarily because the random nature of the workload cannot guarantee a perfectly uniform distribution of the concurrently executing queries among the processors. For example, while 24 processors are being used, at a multiprogramming level of 12 two or more queries may be executing concurrently on a single processor because their randomly generated predicates may overlap [GHAN90].

The throughput with the range and hybrid-range partitioning strategies is almost identical at all multiprogramming levels (the maximum difference is less than 2.5%) because both partitioning strategies localize the execution of the query to a single processor most of the time. While the hybrid-range partitioning strategy must search a significantly larger directory (5700 instead of 24 entries) the execution time of the query is high enough to render this overhead insignificant.

It is essential to observe the impact of the overhead associated with a multi-processor query. At a multiprogramming level of 160, the throughput for the range and hybrid-range partitioning strategies is thirty times higher than that of the hash partitioning strategy. Finally, as these results demonstrate the hybrid-range partitioning strategy performs as well as the range partitioning strategy for queries with minimal resource requirements.

6.2. 10% Selection Using a Clustered Index

The throughput for the alternative declustering strategies for a 10% selection using a clustered index is presented in Figure 5. Using equation 4, the value of M is 90 for this query and the HRPS declusters each million tuple relation into 895 fragments. Ideally, 90 processors should be used to execute this query but since only 24 processors are available, each processor must perform approximately 3.7 (i.e., $\frac{90}{24}$) times the optimal amount of work (see Section 3.2, Figure 1.b). The HRPS directs this query to all the processors.

The range partitioning strategy declusters each relation into 24 fragments, each containing a distinct range of partitioning attribute values. Since the range of a 10% selection query will overlap at most the range of 3 fragments, it will use at most 3 processors to execute this query.

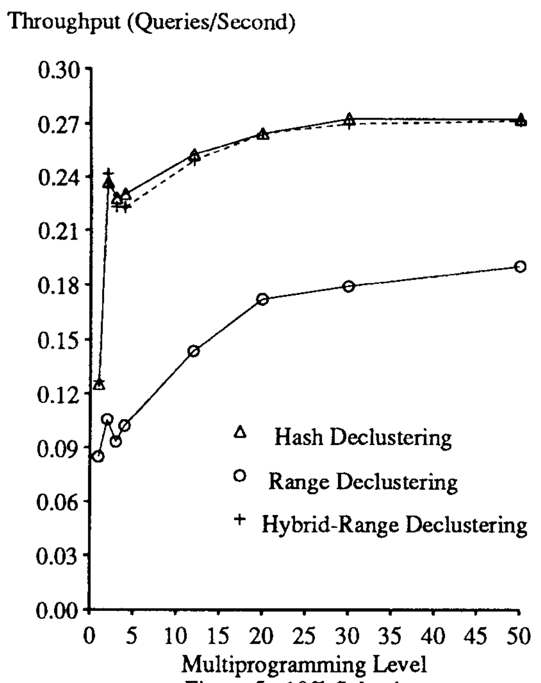

Figure 5: 10% Selection

At a multiprogramming level of one, the throughput with the hash and hybrid-range declustering strategies is 16% higher than that of the range partitioning strategy because the hash and hybrid-range declustering strategies utilize intra-query parallelism effectively while the range partitioning strategy directs the query to the absolute minimum number of processors and performs the majority of the work in a sequential manner.

With the range partitioning strategy, the throughput increases only slightly from a multiprogramming level of one to two because every time two queries are directed to the same processor the response time of each query increases significantly as the SCSI cache becomes ineffective. Since a large number of disk pages are processed by a query at a single processor, when the SCSI cache becomes ineffective, the performance of the system degrades significantly. The throughput of the system increases at higher multiprogramming levels as previously idle resources become more fully utilized.

For the hash and hybrid-range partitioning strategies, the throughput of the system increases significantly from a multiprogramming level of one to two and then drops at a multiprogramming level of three. This affect is again due to the SCSI cache. At multiprogramming levels higher than three, the throughput of the system increases due to processor sharing and utilization of the elevator algorithm at the disk controller [GHAN90].

Why does not the SCSI cache have an impact on the performance of the 0.001% selection query (presented in Section 6.1)? The primary reason is that the 0.001% selection query retrieves and processes only 10 tuples (compared with 100,000 tuples for the 10% selection query). Each 8K byte disk page contains 36 tuples. Thus, the 0.001% selection query processes a single disk page most of the time and two disk pages some of the time[7]. Thus, the impact of the SCSI cache is minimal for this query.

The 10% range selection query represents the best case scenario for the hash partitioning strategy and the worst case scenario for the range partitioning strategy as the hash partitioning strategy utilizes intra-query parallelism effectively to produce the best response time and throughput. The hybrid-range declustering strategy performs as well as the hash partitioning strategy because it also utilizes parallelism to execute this query.

6.3. A Mixed Workload

For our final experiment, we used a mix workload of queries consisting of one-half 0.001% selection queries and one-half 10% selection queries. **It is important to note that the partitioning requirements of these two queries conflict with one another.** While the 0.001% selection should be directed to a single processor, the response time of the 10% selection will be minimized if all the processors are used. While neither the range nor hash partitioning strategies can resolve the conflicting demands of these two queries, the HRPS can. The performance of the different declustering strategies for this mix of queries is presented in Figure 6.

The hybrid-range partitioning strategy declusters each relation into 1688 fragments (using equations 5 and 6) which were distributed among the processors in a round-robin fashion. Thus, the

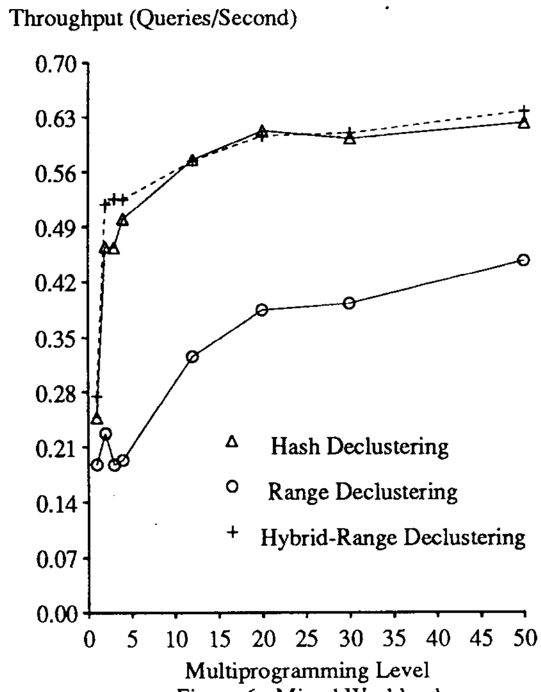

Figure 6: Mixed Workload

[7] This count of disk pages ignores the random disk requests for the B-tree index pages since the SCSI cache is ineffective for random disk requests.

optimizer has enough information with this partitioning strategy to direct the 0.001% selection query to a single processor while using all the processors for the execution of the 10% selection query. On the other hand, the hash partitioning strategy uses all the processors for both queries. The range partitioning strategy directs the execution of the 0.001% selection query to a single processor and uses at most 3 processors for the execution of the 10% selection query.

At a multiprogramming level of one, the throughput of the HRPS partitioning strategy is 11% higher than the hash partitioning strategy and 46% higher than the range partitioning strategy. The principal reason why the hash and hybrid-range partitioning strategies outperform the range partitioning strategy is that the execution of the 10% selection dominates the computation of the average throughput. Since both queries occur with equal frequency in this experiment and since the response time of the 10% query is significantly higher than that of the 0.001% selection query, those partitioning strategies that maximize the throughput for the 10% selection also maximize the throughput for the whole workload. This also explains why the throughput of the hash and hybrid-range declustering strategies converge at multiprogramming levels higher than four. The impact of the 10% selection query on the workload is so significant that the savings provided by the HRPS for the 0.001% selection query is not significant.

We can generalize on the results obtained for this mix of queries and speculate on workloads consisting of different mixes of queries. As the frequency of occurrence of queries with minimal resource requirements (e.g., the 0.001% selection) is increased, the throughput of the range and hybrid-range partitioning strategies converge and outperform the hash partitioning strategy. Conversely, as the frequency of queries with high resource requirements (e.g., the 10% selection query) is increased, the throughput of the hash and hybrid-range partitioning strategies converge and outperform the range partitioning strategy.

To support this hypothesis, we fixed the multiprogramming level of the system at 50 and varied the mix of 10% and 0.001% selection queries in the workload. The results are presented in Figure 7. In this figure, the X axis represents the percentage of 10% selection queries in the workload. For example, at point 80 on the X axis, eighty percent of queries in the workload were 10% selection queries and the remaining twenty percent were 0.001% selection queries. The performance of the hash and hybrid-range partitioning strategies are almost identical when the 10% selection query constitutes more than ten percent of the workload. The HRPS outperforms the hash partitioning strategy by a slight margin since it localizes the execution of the 0.001% selection queries to a single processor.

At first glance, the hash and hybrid-range partitioning strategies appear to outperform the range partitioning strategy by a wider margin when the 10% selection query constitutes ten percent of the workload than when it constitutes more than fifty percent of the workload. This misconception is due to the scale of the Y axis. In Figure 8, we present the percentage improvement in throughput provided by the hash and hybrid-range partitioning strategies relative to the range partitioning strategy. As the percentage of the 10% selection queries is increased, the hash and

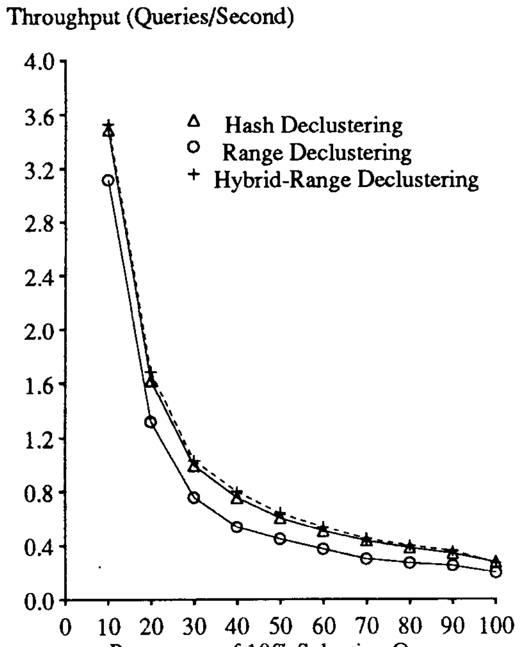

Figure 7: Multiprogramming Level = 50

hybrid-range partitioning strategies outperform the range partitioning strategy by an increasingly widening margin. The percentage improvement levels off when the 10% selection query constitutes forty percent of the queries in workload because the disk has become 100% utilized at this point.

In Figure 9, we present the throughput of the alternative partitioning strategies at a multiprogramming level of fifty for a variety of workloads consisting of a very low percentage of 10% selection queries. (This figure is just an enlargement of the lower limit of Figure 7.) In this figure, the range partitioning strategy begins to outperform the hash partitioning strategy when the 10% selection query constitutes less than five percent of the queries in the workload because the range partitioning strategy can localize the execution of the 0.001% selection queries to a single processor (these queries now constitute more than 95% of the queries in the workload).

The HRPS outperforms both the range and hash partitioning strategies by close to thirty percent when the 10% selection query constitutes five to seven percent of the queries in the workload. For this range of workloads, neither the range nor the hash partitioning strategy is appropriate. Since the HRPS provides the appropriate execution paradigm for both queries in the workload, it can outperform the other two partitioning strategies.

7. Other Advantages of the HRPS

7.1. Support for Small Relations

In database machines with hundreds to thousands of processors, relations with low cardinalities must be partially declustered across a subset of processors [COPE88]. The HRPS does a very good job at supporting small relations since the number of

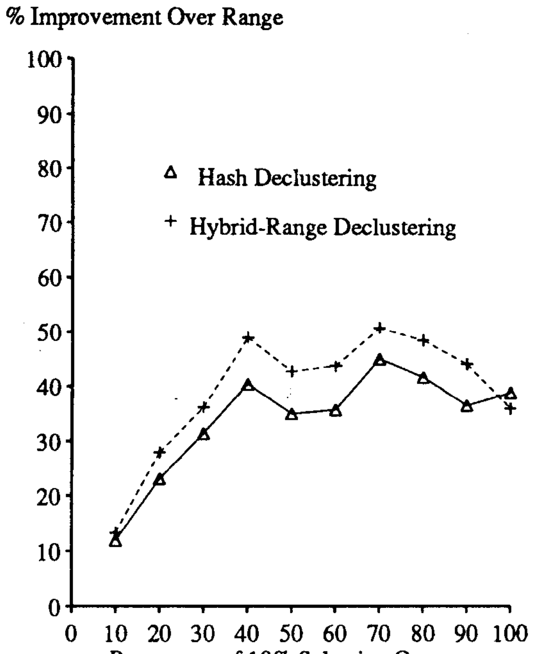

Figure 8: Multiprogramming Level = 50

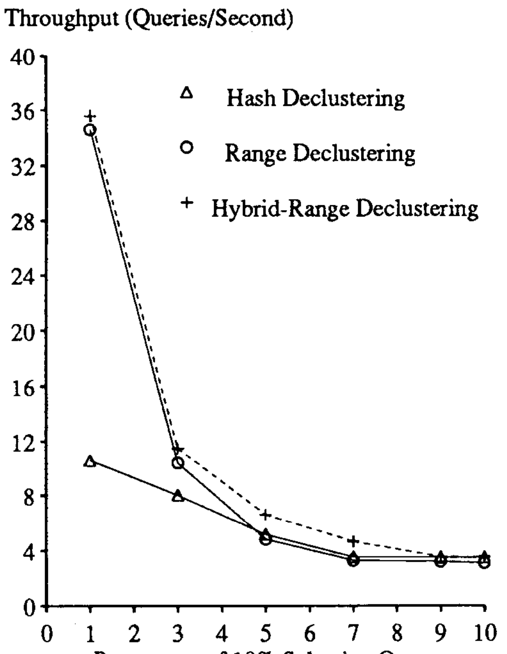

Figure 9: Multiprogramming Level = 50

fragments created by the HRPS is dependent on the processing capability of the system and the resource requirements of the workload and is independent of the number of processors in the multiprocessor. If the number of fragments of a relation is less than the number of processors, then the relation will automatically be partitioned across a subset of the processors.

What about the case of a database that consists of many small relations? In this case, the relations must be declustered such that there are approximately the same number of fragments at each processor with the restriction that each fragment of a relation be assigned to a different processor. Furthermore, the issue of disk space utilization also arises as we would like the database uniformly distributed among all the processors. Since this is clearly a bin packing problem (which is NP complete [GARE79]), one can decluster a relation using a number of alternative heuristics that approximate the optimal solution. One simple heuristic might be to assign the next fragment of a relation to that processor that has the most free disk space available and which does not yet contain any fragments of the relation.

The temperature of an individual fragment of the relation must also be taken into consideration when assigning the fragments of a relation to the processors. The heuristic solutions proposed by [COPE88] appear most promising at this point in time and we refer the interested reader to that study.

7.2. Support for Relations with Non-Uniform Distributions of the Partitioning Attribute Values

The HRPS is also capable of declustering relations with non-uniformly distributed partitioning attribute values since the cardinality of each fragment is not based on the value of the partitioning attribute value. Once the HRPS determines the cardinality of each fragment, it will decluster a relation based on that value. For example, assume relation R has a cardinality of 100,000 tuples and assume that 4,000 of these tuples have 2 as their partitioning attribute value. In addition, assume that the HRPS determines the cardinality of each fragment should be 1000 tuples. After sorting the relation on the partitioning attribute, the tuples with a partitioning attribute value of 2 will be distributed among 4-5 fragments which will each be assigned to a different processor. Thus, if a query with an exact match predicate for value 2 is submitted to the system, the query will be directed to 4 processors (rather than one processor, had the relation been declustered using either the hash or the range partitioning strategy).

8. Conclusions and Future Research Directions

In this paper we have described the design of the hybrid-range partitioning strategy. This new partitioning strategy declusters a relation by analyzing the resource requirements of the queries accessing the relation, the processing capability of the processors in the multiprocessor configuration, and the overhead of using additional processors to execute a query. The goal of this partitioning strategy is to decluster a relation such that the appropriate degree of intra-query parallelism for the queries accessing the relation is obtained. The design provides effective support for small relations and relations with skewed distributions of the partitioning attribute value.

We implemented the hybrid-range partitioning strategy on the Gamma database machine and compared its performance to that of the range and hash partitioning strategies. For query types where either the range or hash partitioning strategies result in the best response time and throughput, we demonstrated that the hybrid-range partitioning strategy has equivalent performance. For a mixed workload of queries with conflicting partitioning requirements, we demonstrated that the hybrid-range partitioning strategy is a better alternative.

It is important to distinguish this study from [COPE88]. The hybrid-range partitioning strategy and the heuristic solutions presented in [COPE88] are orthogonal to each other. In this study we proposed a new partitioning strategy which declusters a relation into an optimal number of fragments while [COPE88] attempted to strike a compromise between load balancing and the overall load reduction in the presence of data locality. Once a relation is declustered into fragments using the hybrid-range partitioning strategy, the fragments can be assigned to the processors using the heuristic solutions proposed by [COPE88]. In the presence of update queries and changing workloads, the heuristic used for file reorganization proposed in [COPE88] can also be used.

A number of open issues remain. First, a major assumption of the hybrid-range partitioning strategy is that the work performed by a query can always be parallelized. This assumption might not be true for all queries. Second, a major limitation of the hybrid-range and the existing partitioning strategies is that they are one dimensional. If the selection predicate of a query is on an attribute other than the partitioning attribute, it must be sent to all the processors containing the fragments of the relation. This is clearly a major limitation that requires additional study.

9. Acknowledgments

We wish to acknowledge the contributions of several people. We would like to thank Miron Livny for the lengthy discussions over the performance numbers presented in Section 6. These discussions significantly enhanced the presentation of this section of the paper. We also wish to acknowledge Rick Rasmussen who provided technical support for our research vehicle, the Intel iPSC/2 hypercube. The performance numbers obtained from the hypercube would not have been possible without his efforts. We would also like to thank Mike Carey for reading and providing useful comments on an earlier draft of this paper.

10. References

[ALEX88] Alexander, W., et. al., "Process and Dataflow Control in Distributed Data-Intensive Systems," Proc. ACM SIGMOD Conf., 1988.

[BITT83] Bitton D., D.J. DeWitt, and C. Turbyfill, "Benchmarking Database Systems - A Systematic Approach," Proceedings of the 1983 VLDB Conference, October, 1983.

[BORA84] Boral, H, and DeWitt, D. J., "A Methodology for Database System Performance Evaluation," Proc. ACM SIGMOD Conf., 1984.

[BORA88] Boral, H., "Parallelism and Data Management," Proceedings of the 3rd int'l. Conf. on Data and Knowledge Bases, June 1988.

[BULT89] Bultzingsloewen, G, V., "Optimizing SQL Queries for Parallel Execution," ACM Sigmod Record, Vol. 18, No. 4, Dec., 1989.

[CHOU85] Chou, H-T, DeWitt, D. J., Katz, R., and T. Klug, "Design and Implementation of the Wisconsin Storage System (WiSS)" Software Practice and Experience, Vol. 15, No. 10, October, 1985.

[COPE88] Copeland, G., Alexander, W., Boughter, E., and T. Keller, "Data Placement in Bubba," Proc. ACM SIGMOD Conf., 1988.

[DEWI86] DeWitt, D., Gerber, B., Graefe, G., Heytens, M., Kumar, K. and M. Muralikrishna, "GAMMA - A High Performance Dataflow Database Machine," Proceedings of the VLDB Conf., 1986.

[DEWI88] DeWitt, D., Ghandeharizadeh, S., and D. Schneider, "A Performance Analysis of the Gamma Database Machine", Proc. ACM SIGMOD Conf., June 1988.

[DEWI90] DeWitt, D., Ghandeharizadeh, S., Schneider, D., Bricker, A., Hsiao, H., R. Rasmussen, "The Gamma Database Machine Project," IEEE Transaction on Knowledge and Data Engineering, March, 1990.

[DU82] Du, H.C. and Sobolewski, J.S., "Disk Allocation for Cartesian Product Files on Multiple-Disk Systems," ACM Trans. Database Systems, Vol. 7, No. 1, March 1982, pp. 82-101.

[GARE79] Garey, M. R., Johnson D. S., **Computers and Interactability: A Guide to the Theory of NP-Compelteness**, New York, 1979.

[GHAN90] Ghandeharizadeh, S., and D. J. DeWitt, "Performance Analysis of Alternative Declustering Strategies," Proceedings of the 6th International Conference on Data Engineering, February 1990.

[INTE88], Intel Corporation, **iPSC/2 User's Guide**, Intel Corporation Order No. 311532-002, March, 1988.

[KIM88] Kim, M.H., and Pramanik, S., "Optimal File Distribution for Partial Match Retrieval," Proc. ACM-SIGMOD Conf., June 1988.

[LIVN87] Livny, M., et al., "Multi-Disk Management Algorithms," Proc. ACM-SIGMETRICS Conf., May 1987.

[LORI88] Lorie, R., et al., "Adding Intra-Transaction Parallelism to an Existing DBMS: Early Experience" IBM Research Report RJ6165. Almaden Research Center, March 1988.

[PATT88] Patterson, D., et al., "RAID: Redundant Arrays of Inexpensive Disks," Proc. ACM-SIGMOD Conf., June 1988.

[PRAM89] Pramanik, S., and M. H. Kim, "Database Processing Models in Parallel Processing Systems," **Database Machines**, Boral, H., and P. Faudemay, eds., Springer-Verlag, 1989.

[RIES78] Ries, D. and R. Epstein, "Evaluation of Distribution Criteria for Distributed Database Systems," UCB/ERL Technical Report M78/22, UC Berkeley, May, 1978.

[SCHN89] Schneider, D. and D. J. DeWitt, "A Performance Evaluation of Four Parallel Join Algorithms in a Shared-Nothing Multiprocessor Environment," Proc. ACM-SIGMOD Conf., June 1989.

[SMIT89] Smith, M., et al., "An Experiment on Response Time Scalability in Bubba," **Database Machines**, Boral, H., and P. Faudemay, eds., Springer-Verlag, 1989.

[STON86] Stonebraker, M., "The Case for Shared Nothing," Database Eng. 9, 1, March 1986.

[STON88] Stonebraker, M., et al., "The Design of XPRS," Proceedings of the 1988 VLDB Conference, Sept, 1988.

[TAND88] Tandem Performance Group, "A Benchmark of Non-Stop SQL on the Debit Credit Transaction", Proc. ACM-SIGMOD Conf., June 1988.

[TANE81] Tanenbaum, A. S., **Computer Networks**, Prentice-Hall, 1981.

[TEOR72] Teorey T. J., and Pinkerton T.B., "A Comparative Analysis of Disk Scheduling Policies", Commun. of ACM, 15:3, March 1972.

[TERA85] Teradata Corp., "DBC/1012 Data Base Computer System Manual, Rel. 2.0," Teradata Corp. Document No. C10-0001-02, November 1985.