A Performance Analysis of Alternative Multi-Attribute Declustering Strategies

Shahram Ghandeharizadeh
Department of Computer Science
University of Southern California

David J. DeWitt
Computer Sciences Department
University of Wisconsin-Madison

Waheed Qureshi
Department of Computer Science
University of Southern California

Abstract

During the past decade, parallel database systems have gained increased popularity due to their high performance, scalability and availability characteristics. With the predicted future database sizes and the complexity of queries, the scalability of these systems to hundreds and thousands of processors is essential for satisfying the projected demand. Several studies have repeatedly demonstrated that both the performance and scalability of a parallel database system is contingent on the physical layout of data across the processors of the system. If the data is not declustered properly, the execution of an operator might waste resources, reducing the overall processing capability of the system.

With earlier, single attribute declustering strategies, such as those found in Tandem, Teradata, Gamma, and Bubba parallel database systems, a selection query including a range predicate on any attribute other than the partitioning attribute must be sent to all processors containing tuples of the relation. By directing a query with minimal resource requirements to processors that contain no relevant tuples, the system wastes CPU cycles, communication bandwidth, and I/O bandwidth, reducing its overall processing capability. As a solution, several multi-attribute declustering strategies have been proposed. However, the performance of these declustering techniques have not previously been compared to one another nor with a single attribute partitioning strategy. This paper, compares the performance of Multi-Attribute GrId deClustering (*MAGIC*) and Bubba's Extended Range Declustering (*BERD*) strategies with one another and with the range partitioning strategy. Our results indicate that MAGIC outperforms both range and BERD in all experiments conducted in this study.

1 Introduction

During the past few years database management systems (DBMS) have become an essential components of many application domains (e.g., airline reservations, stock market trading etc). In the arena of high performance DBMS, multiprocessor database machines (Gamma [DGS$^+$90], Bubba [BAC$^+$90], XPRS [SPO88], Non-Stop SQL [Tan88], DBC/1012 [Ter85], Volcano [Gra89]) have become increasingly popular. In such systems, relations are generally horizontally declustered [RE87] [LKB87] across multiple processors. Hash and range are two widely used declustering strategies. In the first, a randomized function is applied to the partitioning attribute of each tuple to select a home processor for that tuple. This enables selection operators with an equality predicate on the partitioning attribute to be directed to a single processor. However operators with a range predicate must be sent to all the processors containing fragments of the relation. In the range partitioning strategy, the database administrator specifies a range of key values for each processor. This strategy provides a greater degree of control over the distribution of tuples across the processors and the execution of selection operators with either a range or an equality predicate on the partitioning attribute can be directed to those processors that contain the relevant fragments.

For a selection operator, the number of processors that a relation is declustered across determines the maximum degree of parallelism for the queries referencing it. However, if only a few tuples satisfy the predicate of a selection operator and its resource requirements(CPU or I/O) are minimal, then the execution of the operator should be localized by partitioning the tuples that satisfy the query across only a few processors [CABK88]. Doing so will minimize the execution time for the operator by minimizing the communication overhead associated with scheduling and terminating the operator on multiple processors. Using more processors simply wastes CPU cycles and I/O bandwidth and may actually reduce the throughput as these processors consume CPU cycles and I/O bandwidth only to determine that their fragments of the relation do not contain any relevant tuples. Minimizing the number of processors used to execute such operators frees the other processors in the system to execute other operators, increasing the overall processing capability of the system.

One major limitation of both the range and hash declustering strategies is that neither can decluster a relation on more than one attribute. Therefore, any selection operation whose predicate includes an attribute other than the partitioning attribute must be directed to all the processors that contain fragments of the relation. In a system consisting of hundreds to thousands of processors, the overhead of initiating a selection operator on processors that do not contain any relevant tuples can actually constitute a major fraction of the query's execution time. In addition, most of the processors will search their fragment of the relation to find no relevant tuples.

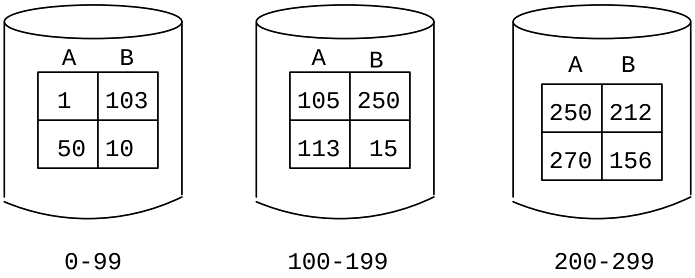

Figure 1: Range Partition R on Attribute A

As a solution to this limitation, several multi-attribute declustering strategies have been proposed. However, the performance of these declustering strategies have not been compared with one another nor with a single attribute declustering strategy. This paper compares the performance of two multi-attribute declustering strategies, BERD and MAGIC, with each other and with the range partitioning strategy. The rest of this paper is organized as follows. Sections 2 and 3 contain a brief overview of each of these partitioning strategies. In Section 4, we discuss the impact of high correlation of the partitioning attribute values on the execution paradigm provided by both multi-attribute partitioning strategies. Section 5 describes the simulator used for this performance evaluation. Sections 6 and 7 describe the workload and the performance of the alternative multi-attribute partitioning strategies for this workload, respectively. Our conclusions are contained in section 8.

2 Bubba's Extended-Range Declustering (BERD) Strategy

BERD declusters a relation by distinguishing between the primary and multiple secondary partitioning attributes for the relation. First, the relation is range partitioned using the primary partitioning attribute value. Then, for each of the secondary partitioning attributes, an auxiliary "relation" is formed from its attribute values, their associated tuple identifiers, and the processor on which the original tuple resides. The tuples in each of these relations are then range partitioned over multiple processors and the fragments at each processor are organized in the form of a B-tree index on the attribute value.

In order to illustrate how BERD partitions a relation, consider relation R with two attributes

B	Proc
103	1
250	2
212	3
10	1
15	2
156	3

Figure 2: Auxiliary Relation IndexB

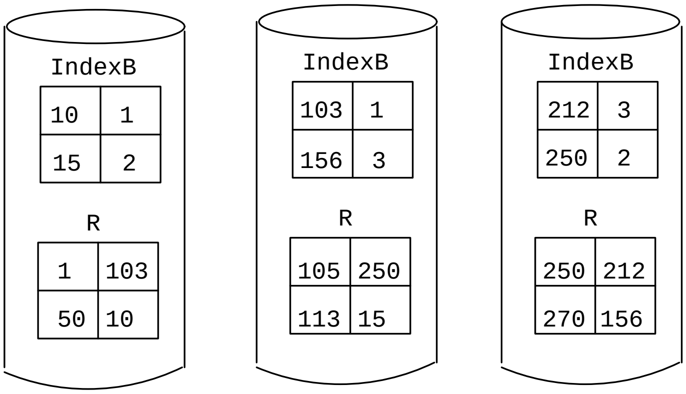

Figure 3: Range Partition IndexB on Attribute B

A and B, and a cardinality of six tuples. Assuming that the primary partitioning attribute is $R.A$, relation R is range declustered as shown in Figure 1. Next, each fragment of R is scanned and an auxiliary "relation" is constructed from the attribute values of the secondary partitioning attribute and a list of processors containing the original tuples. Assuming that the secondary partitioning attribute is $R.B$, Figure 2 presents this auxiliary relation (called IndexB). Next, this relation is range partitioned on attribute B, resulting in the assignment shown in Figure 3. If a query with a range predicate on attribute A (e.g., retrieve R.all where $R.A < 50$) is submitted, the query optimizer utilizes the partitioning information of this attribute to direct the query to those processors with the relevant fragments. If the query predicate references attribute B (e.g., retrieve R.all where $R.B < 50$), the system first uses the IndexB relation to determine which processors contain qualifying tuples (processors 1 and 2 for this query) and directs the query to these processors.

3 Overview of MAGIC Declustering

MAGIC partitions a relations by constructing a grid directory on a relation (see Figure 4 for a two dimensional grid directory) where each entry in the grid represents a fragment of the relation. Before describing MAGIC in detail, we start with an example in order to demonstrate the execution paradigm it provides for queries accessing different partitioning attributes. Assume a STOCK relation with the following attributes:

STOCK (ticker_symbol, name, price, closing, opening, P/E)

Assume that one half of the accesses (termed query type A) to the Stock relation use an equality predicate on the *ticker_symbol* attribute (e.g., select STOCK.all where STOCK.ticker_symbol = "AXP") and that the remaining queries (termed query type B) use a range predicate on the *price* attribute (e.g., select STOCK.all where STOCK.price > 10 and STOCK.price <= 20). Furthermore, assume that both queries produce only a few tuples and use an access method to retrieve the qualifying tuples. For this workload, MAGIC declustering would construct the two dimensional directory on the STOCK relation, shown in Figure 4, in which each entry corresponds to a **fragment** - i.e., a disjoint subset of the tuples of the relation. The rows of the directory correspond to ranges of values for the *price* attribute, while the columns correspond to the intervals of the *ticker_symbol* attribute. The grid directory consists of 36 entries (i.e., fragments), and assuming a system consisting of exactly 36 processors, each fragment will be assigned to a different processor. For example, tuples with *ticker_symbol* attribute values ranging from letters A through D and *price* attribute values ranging from values 21 to 30 are assigned to processor 13.

Next, consider the execution paradigm of query type A and B with MAGIC. Query type A is an exact match query on the ticker_symbol attribute and MAGIC will use six processors to execute this query because its selection predicate maps to one column of the two dimensional directory. As an example, consider the query that selects the record corresponding to American express (STOCK.ticker_symbol = "AXP"). The predicate of this query maps to the first column of the grid directory and processors 1, 7, 13, 19, 25, and 31 would be used to execute it. Similarly

Ticker_Symbol

		A-D	E-H	I-L	M-P	Q-T	U-Z
PRICE	0-10	1	2	3	4	5	6
	11-20	7	8	9	10	11	12
	21-30	13	14	15	16	17	18
	31-40	19	20	21	22	23	24
	41-50	25	26	27	28	29	30
	51-60	31	32	33	34	35	36

Figure 4: A two dimensional directory on the STOCK relation

MAGIC directs query type B to six processors[1] since its predicate maps to one row of the grid directory and the entries of each row have been assigned to six different processors. Consequently, the MAGIC partitioning strategy uses an average of six processors for both queries. In comparison, a one dimensional partitioning strategy such as range would use an average of 18.5 processors for a similar workload. This is because range partitioning can only decluster a relation on one attribute (say attribute *Price*) and will direct the query referencing that attribute (query type B) to one processor while type A queries must be directed to all the 36 processors.

3.1 Overview of the MAGIC Partitioning Strategy

In order to decluster a relation R using MAGIC, the database administrator must specify the K partitioning attributes, the resource requirements of each selection operation on the relation, and their respective frequency of execution. Using this information, MAGIC computes the number of processors (termed M) that should be used to execute the "average" query representative of

[1] Assuming that the range of query does not overlap two intervals of the grid directory.

Term	Definition
K	Number of partitioning attributes (i.e., the number of dimensions in a directory)
P	Number of processors in the system
Q_{Ave}	Average query representative of the individual queries in the workload
M	Ideal number of processors to execute Q_{Ave}
M_i	Ideal number of processors to execute queries whose predicate includes attribute i
N_i	Number of range intervals associated with dimension i
CPU_i	The CPU processing requirement of query Q_i
$Disk_i$	The Disk processing requirement of query Q_i
Net_i	The Network processing requirement of query Q_i
$FreqQ_i$	Frequency of occurrence of Q_i in the workload
CP	Cost of Participation - overhead of using an additional processor to execute a query
FC	Cardinality of a fragment
CS	Cost of searching the grid directory constructed by MAGIC declustering

Table 1: List of terms used repeatedly in this paper and their respective definitions

the workload (termed Q_{Ave}). In order to ensure that M processors are used to execute Q_{Ave}, the cardinality of each fragment of R is set to $\frac{1}{M-1}$ th of the number of tuples processed by Q_{Ave}. Thus, the selection predicate of Q_{Ave} will cover M fragments. If each of the M fragments is assigned to a different processor, the correct number of processors will be used to execute Q_{Ave}. In addition to M, MAGIC declustering also computes M_i, the number of processors that should be used to execute those selection operators whose predicate includes attribute i based on the resource requirements of the corresponding subset of selection operations. M_i is used to determine the splitting strategy when constructing a grid directory on the relation.

Once these values have been determined, MAGIC provides the insertion phase of the grid file algorithm [NHS84] with the following information: 1) the cardinality of each fragment, 2) the splitting strategy, 3) the relation to be declustered, and 4) the K partitioning attributes. The grid file algorithm scans the relation and constructs a K dimensional grid directory whose ith dimension consists of N_i ranges of partitioning attribute values. Each such range is termed a *slice*. For example, the *ticker_symbol* attribute in Figure 4 is composed of six slices (A-D, E-H, etc.). The MAGIC partitioning strategy then analyzes the grid directory and assigns its entries (corresponding to fragments of the relation) to the processors.

During the final stage of the partitioning process, the relation is scanned a second time and tuples are assigned to processors based on the contents of the grid directory. The grid directory is then stored in the database catalog and is used by the query optimizer to localize the execution of those selection operators whose predicate involves one or more of the partitioning attributes.

3.2 Computing the Cardinality of a Fragment

Assume a mixed workload consisting of n selection operations. Each operation Q_i retrieves and processes $TuplesPerQ_i$ tuples from the database, has a frequency of execution of $(FreqQ_i)$, and consumes three principle resources during its execution: 1) CPU cycles, 2) disk I/Os, and 3) communications bandwidth. For each operation 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 the operation. Q_{Ave} is representative of all the queries in the workload. The number of tuples retrieved by Q_{Ave} are $(TuplesPerQ_{Ave} = \sum_{i=1}^{n}(TuplesPerQ_i * FreqQ_i))$. Similarly, each of the CPU, disk and the network processing quanta for this query is the weighted sum of the quanta of the individual operations (e.g., $CPU_{Ave} = \sum_{i=1}^{n}(CPU_i * FreqQ_i))$. Observe that these times are determined based on the resource requirements of the operation and the processing capacity of the system and thus, we derive the cardinality of the fragment based on the resource requirement of the average query, Q_{Ave}.

As a relation is partitioned over additional processors, the response time of the query decreases proportionally[2]. However, the overhead (termed CP) incurred for each additional processor must be taken into account. Assume a linear increase of this overhead as a function of the processors employed (as is the case in the Gamma database machine)[3]. In addition, if the resource requirements of the query are extremely low, then searching the one dimensional directory can constitute a significant portion of the query's execution time. The number of entries in the grid directory can be at most $\frac{(M-1)*Cardinality(Rel)}{TuplesPerQ_{Ave}}$. If a linear search is performed then one half of the entries must be searched to find the qualifying entries. Thus the response time of the average query Q_{Ave} is

[2] The linear speedup results presented in [DGS$^+$90] justify this claim.

[3] The algorithm used to schedule and commit a multiprocessor query defines this function.

defined by:

$$RT(M) = \frac{CPU_{Ave} + Disk_{Ave} + Net_{Ave}}{M} + M * CP + \frac{(M-1) * Card(Rel) * CS}{2 * TuplesPerQ_{Ave}} \qquad (1)$$

where **CS** represents the cost of searching a single entry in the directory. Hence the desired number of processors (i.e., the value of M that minimizes RT(M)) to execute Q_{Ave} is specified by setting the first derivative of equation 1 to zero and solving the value of M:

$$M = \sqrt{\frac{CPU_{Ave} + Disk_{Ave} + Net_{Ave}}{CP + \frac{Cardinality(Rel) * CS}{2 * TuplesPerQ_{Ave}}}}$$

M represents the number of processors that should be employed to execute Q_{Ave}. Since Q_{Ave} processes $TuplesPerQ_{Ave}$ tuples, each fragment of the relation should contain **FC** $= \frac{TuplesPerQ_{Ave}}{M-1}$ tuples[4].

In addition to M, M_i, which specifies the number of processors that should be employed to execute queries accessing attribute i is also determined. Thus, ideally, M_i different processors should appear in each slice of dimension i. Assuming s represents the number of different queries whose selection predicate includes attribute i, we first compute the frequency of occurrence of each query relative to the total frequency of all queries whose predicate includes this attribute (termed RelFreqQ). For each query j, the relative frequency of that query is:

$$RelFreqQ_j = \frac{FreqQ_j}{\sum_{r=1}^{s} FreqQ_r} \qquad (2)$$

Then, the value of M_i for attribute i is:

$$M_i = \sqrt{\frac{\sum_{r=1}^{s}(CPU_r + Disk_r + Net_r) * RelFreqQ_r}{CP}} \qquad (3)$$

The value of M_i is used both to determine the splitting strategy when constructing the grid directory and while assigning the elements of the grid directory to processors.

3.3 Creating the Grid Directory

In order to construct a grid directory on a relation, the grid file algorithm requires the frequency with which each dimension of the grid file should be split. This determines the shape of the grid directory, in particular, the number of range intervals (N_i) each dimension i is divided into.

[4] If $M < 1$, then $FC = \frac{TuplesPerQ_{Ave}}{M}$.

Ticker_symbol

	1	4	7	10	13	16	19	22	25	28	31	34
PRICE	2	5	8	11	14	17	20	23	26	29	32	35
	3	6	9	12	15	18	21	24	27	30	33	36

Figure 5: Different values for M_i

Since the M_i value for each dimension will normally be different, the values of M_i, for $i = 1$ to K, must be taken into consideration when splitting the directory in order to ensure that each slice of dimension i has at least M_i elements (so that M_i different processors can be assigned to each interval or slice of dimension i). To achieve this, the number of elements in each dimension should be proportional to the $Fraction\_Splits_i$ defined as:

$$Fraction\_Splits_i = FreqQ_i * (\frac{(\sum_{j=1}^{K} M_j) - M_i}{\sum_{j=1}^{K} M_j}) \tag{4}$$

This results in a directory with at least M_i elements for each element of dimension D_i. For example, assume $M_{ticker-symbol} = 3$ and $M_{price} = 1$. Also, assume that 90% of the queries access the *ticker_symbol* attribute and 10% access the *price* attribute. Using Equation 4, the splitting frequencies of the *ticker-symbol* and *price* attributes can be calculated to be 22.5% and 7.5%, respectively[5]. Consequently, the *ticker-symbol* attribute will have three times as many elements as the *price* attribute, resulting in the directory shown in Figure 5 [6]. Using the cardinality of each fragment and the splitting frequencies for each dimension, the grid file algorithm scans the relation and constructs a grid directory on the relation.

In the context of MAGIC declustering, each element of the directory corresponds to one fragment of the relation. The output of the grid file algorithm is a K dimensional directory whose ith dimension consists of N_i ranges of partitioning attribute values.

[5] Note that $\sum_{i=1}^{K} Fraction\_Splits_i$ does not equal 1.0. This is not important because our objective is to compute the ratio of splits along each dimension. In this example, the *ticker-symbol* attribute should be split three times as frequently as the *price* attribute.

[6] In this figure, the number of slices along the *ticker-symbol* is actually four times that of the *price* attribute. With the computed ratio of *fraction_splits* (using Equation 6), the grid file algorithm will construct either a 3x12 or a 4x9 directory. In either case, the number of elements for each slice of the *ticker-symbol* attribute will be equal to or greater than $M_{ticker-symbol}$.

3.4 Assigning the Grid Directory Entries to Processors

Once the construction of the grid directory has been completed, MAGIC analyzes the grid directory and assigns its elements (i.e., fragments of the relation) to the processors. This is a complex task since two conflicting goals must be satisfied. On one hand, M_i different processors should be assigned to each dimension D_i, while on the other hand, the elements of the grid directory should be distributed evenly among the processors in order to: 1) use the full processing capacity of the system and, 2) balance the load evenly across the processors. When the fragments in the grid directory are less than or equal to the number of processors in the system , the assignment of the fragments to the processors is straightforward; each fragment (or element of the grid directory) is assigned to a different processor. When the number of fragments is greater than the number of processors in the system [7] , the problem can be posed as a integer programming problem which can be solved for a optimal solution [GMSY90]. However, this requires an exhaustive search of the solution space and even for small grids the time requirements are unacceptable. [Gha90] presents a heuristic that approximates the optimal solution and evaluates against the theoretical lower bound computed in [GMSY90]. The results obtained demonstrates that the heuristic can approximate an assignment that corresponds to the theoretical lower bound. The interested reader is referred to [Gha90] for complete details of the heuristic and its evaluation.

4 Correlation of the Partitioning attribute values

When the values of partitioning attributes of a relation are highly correlated, it can have a significant impact on the performance of the alternative multi-attribute declustering strategies. Selection queries on highly correlated attributes can be localized to a single processors (or fewer processors than what was originally anticipated). As an example, consider the relation:

Emp (ss#, name, age, salary, dept_no)

Assume that the age and salary attributes are the partitioning attributes. Furthermore, assume a

[7] When $K = 1$, the assignment of elements to processors in a round robin fashion satisfies both of the constraints [Gha90].

Figure 6: Distribution of tuples in the presence of highly correlated partitioning attribute values

high correlation between these attribute values, where the salary of an employee increases proportionally to his/her age. Assuming that the salary attribute is the primary partitioning attribute, BERD range partitions the relation using the salary attribute values and constructs an auxiliary relation using the age attribute value. Range queries that reference the partitioning attribute (termed Q_{Salary}) will be executed by a single processor. However, the number of processors used to execute queries that reference the secondary partitioning attribute (termed Q_{age}) depends on: 1) the degree of correlation between the age and salary attributes, and 2) the selectivity factor of the query. Assuming that Q_{age} retrieves 10 tuples and that there is a low correlation between the two attributes values, Q_{age} is executed by at most eleven processors. This is because with BERD partitioning, this query is executed in two steps: First, Q_{age} is directed to the processor with the relevant fragments of the auxiliary relation to determine the location of the 10 qualifying tuples. During the second step, Q_{age} is directed to the processors found in the first step. When there is a low correlation between the partitioning attributes values, the first step is likely to determine that the 10 qualifying tuples are distributed across ten distinct processors.

However, when there is a high correlation between the partitioning attribute values, the 10 qualifying tuples could be located on the same processor that contains the relevant fragment of the auxiliary relation, localizing the execution of Q_{age} to a single processor.

With MAGIC, when the partitioning attributes values of a relation are highly correlated, the tuples of a relation may not be evenly distributed across the entries of its grid directory (see Figure 6). This is a mixed blessing. On one hand, the optimizer will direct queries that reference either partitioning attributes to fewer processors than that originally obtained by the assignment procedure. This is because while the assignment procedure assigns the empty entries of the grid directory to processors (i.e., an entry corresponds to a fragment), if an entry contains no tuples, then the query optimizer does not have to direct the query to that processor. This improves the performance of the system for queries with minimal resource requirements (i.e., those that should be directed to one or two processors) because the assignment procedure generally over-estimates the value of M_i (the ideal number of processors that should be employed to execute queries referencing attribute i).

On the other hand, when partitioning attribute values are highly correlated, the original assignment step of MAGIC will almost always result in a skewed distribution of the tuples across processors (see Figure 6) because: 1) the assignment step assumes a uniform distribution of tuples across the entries of the grid directory, and 2) it attempts to approximate an even distribution of tuples by assigning the same number of grid entries to each processor. In [Gha90], we describe a heuristic for approximating a uniform distribution of tuples. Briefly, the heuristic determines the processors with the fewest and the one with the most tuples. Next, it switches the assignment of either two rows or two columns (i.e., two slices in a dimension K) in order to reduce the weight difference between these two processors. It uses a hill climbing search technique and swaps the assignment of those two slices that minimizes the weight difference by the greatest margin. It is important to note that by swapping two slices of a dimension, the number of unique processors that appear in each dimension does not change. The heuristic is very effective and [Gha90] contains an evaluation of it. As an example, in the worst case scenario where the value of the two partitioning attributes is identical for each tuple of a relation, for a 32 processor system, the original assignment of entries would have resulted in a very skewed distribution with 12 processors containing no tuples of the relation. After applying the heuristic, there was only a 20% difference between any two processors.

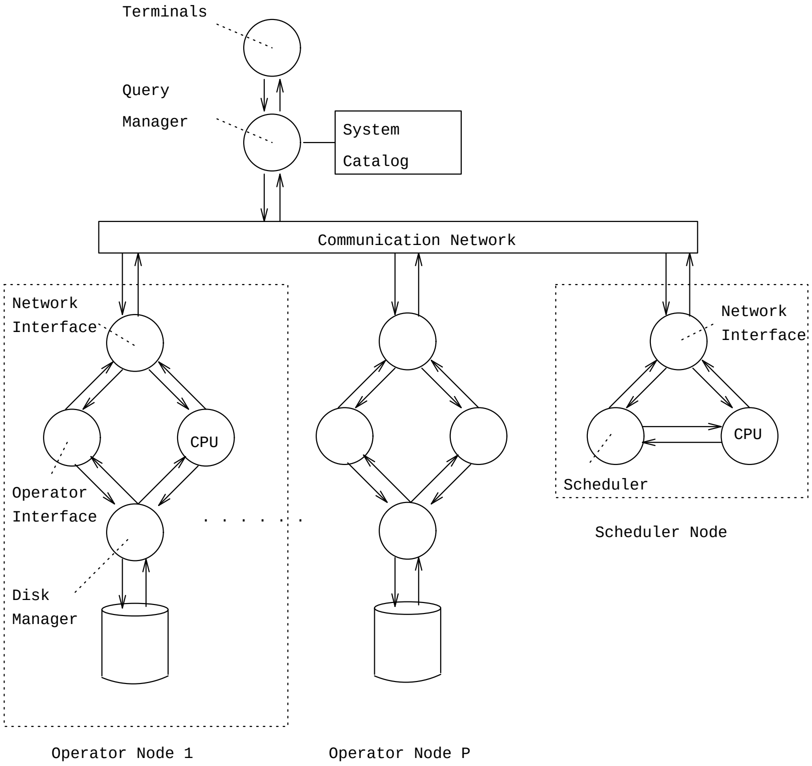

Figure 7: Simulation Model

5 Simulation Model

We used a simulation model of Gamma to implement and evaluate the alternative declustering strategies. This simulation model is organized as follows. Each node in the multiprocessor is composed of a Disk Manager, an Operator Manager, and a Network Interface manager. Additionally, five stand-alone modules are provided: a Communication Network manager, a Query Manager, a Terminal module, a Query Scheduler and the System Catalog manager. See Figure 7 for a picture of the entire simulator. The DeNet simulation language [Liv88] was used to construct the simulator.

The Disk Manager (DM) schedules disk requests to an attached disk according to the elevator algorithm [TP72]. In order to accurately reflect the hardware currently being used by Gamma [DGS$^+$90], the disk manager interrupts the CPU when there are bytes to be transferred from the I/O channel's FIFO buffer to memory or vice versa. The CPU module enforces a FCFS non-preemptive scheduling paradigm on all requests, except for byte transfers to/from the disk's FIFO buffer. An Operator manager is responsible for modeling the relational operators (e.g., se-

Disk Parameters	
Average Settle Time	2 msec
Average Latency	0-16.68 msec (Unif)
Transfer Rate	1.8 MBytes/sec
Seek Factor	0.78 msec
Disk Page Size	8 Kbytes
Xfer Disk page from SCSI to memory	4000 instructions
Network Parameters	
Maximum Packet Size	8 Kbytes
Send 100 bytes	0.6 msec
Send 8192 bytes	5.6 msec
CPU Parameters	
Instructions/Second	3,000,000
Read 8K Disk Page	14600 instructions
Write 8K Disk Page	28000 instructions
Miscellaneous	
Tuple Size	208 bytes
Tuples/Network Packet	36
Tuples/Disk Page	36
Number of Processors	32

Table 2: Important Simulation Parameters

lect). This manager repeatedly issues requests to the CPU, Disk and Network Interface managers to perform its particular operation. The Network Interface manager enforces a FCFS protocol for access to the global communications network. The Network module currently models a fully connected network and the Terminal module provides the entry point for new queries. The Query Manager constructs a query plan for executing a multi-site query. The Query Scheduler coordinates the execution of the operators of a multi-site query. Finally, the System Catalog manager keeps track of how many relations are defined, what disk each relation is declustered across, which partitioning strategy is used to decluster a relation, and the number of pages of each relation on each disk. For each relation, a mapping from logical page numbers to physical disk addresses is also maintained. This physical assignment of pages allows for accurate modeling of sequential as well as random disk accesses. Indices, including both clustered and non-clustered B^+ trees can be constructed on a relation. Table 5 summarizes the key parameters of the model.This simulation model was validated against the Gamma database machine (see [Sch90, Hsi90] for details).

6 Workload Definition

The goal of this performance evaluation was to compare the performance of MAGIC and BERD declustering with one another and the range partitioning strategy. In order to achieve this objective, we designed a multiuser workload and used the throughput of the system as the evaluation criteria. The database consisted of a relation R and its workload consisted of two query types (Q_A and Q_B). Q_A is an average query representative of the queries referencing attribute $R.A$ of relation R while Q_B is representative of those that reference attribute $R.B$. Attribute $R.A$ serves as the partitioning attribute for the range partitioning strategy and as the primary partitioning attribute for Bubba's declustering strategy. Attribute $R.B$ serves as a secondary declustering attribute for Bubba's partitioning strategy. MAGIC constructs a two dimensional grid directory using attributes A and B.

With respect to their utilization of resources, Q_A and Q_B can be categorized as: "low", "moderate", or "high". Queries with low resource requirements should be directed to only 1-2 processors in order to minimize the overhead of parallelism. On the other hand, all processors should be employed to execute queries with high resource requirements in order to avoid the formation of hot spots and bottleneck processors [GD90]. Finally, queries with moderate resources requirements should be directed to a subset of processors. We eliminated queries with high resource requirements from our workload because such queries can always be directed to all the processors by simply ignoring the partitioning information. Consequently, there are four possible query mixes:

Q_A	Q_B
low	low
low	moderate
moderate	low
moderate	moderate

For this performance evaluation, we simulated a 32 processor configuration of the Gamma database machine. The database consisted of a 100,000 tuple relation (relation R). The characteristics of this relation is based on the standard Wisconsin benchmark relations [BDC83] and consists of thirteen attributes. Two of its attributes are termed unique1 and unique2, and their values are uniformly distributed between 0 and 100,000. In our workload, attribute A corresponds to unique1 and attribute B to unique2. In all experiments, there is a non-clustered index on attribute A and a clustered index on attribute B. Queries with low resource requirements that access attribute A are modeled as a single tuple retrieval query using a non-clustered index while those that access attribute B are modeled as a 0.01% clustered-index range selection query (retrieves 10 tuples). Queries with moderate resource requirements that access attribute A are modeled as a 0.03% non-clustered range selection query (retrieves 30 tuples) while those that access attribute B are modeled as a 0.3% clustered-index range selection query (retrieves 300 tuples). The different queries chosen to represent a query with either low or moderate requirements have almost identical execution times even though they reference different attributes and use different types of indexes. For example, the 0.03% non-clustered range selection query referencing attribute A has almost the same execution time as the 0.3% clustered index selection query referencing attribute B. Ideally, both of these queries should be directed to nine processors. In each experiment, the workload consisted of 50% of each query type Q_A and Q_B.

The queries used here are representative of our target class of queries. Each query type could have been modeled as a set of queries with different execution times. However, as long as the average query representative of this set of queries retrieves the same number of tuples and has an identical execution time, the performance results and conclusions drawn in the following sections would be applicable.

For each query mix, we analyzed the performance of the system in the presence of both a low and a high correlation of the partitioning attribute values.

7 Performance Evaluation of Alternate Partitioning Strategies

7.1 Low-Low Query Mix

Figure 8.a presents the throughput of the system with alternative declustering strategies for the low-low query mix as a function of the multiprogramming level when there is a low correlation between the partitioning attributes. For this workload, MAGIC constructs a 62x61 grid directory

Figure 8: Low-Low Query Mix

and, on the average, uses 6.39 processors. The range partitioning strategy directs Q_A to one processor and Q_B to all 32 processors, utilizing 16.5 processors on average. With BERD, Q_A is directed to a single processor while Q_B is directed to at most eleven processors, utilizing 6 processors on the average. MAGIC, however, outperforms BERD by approximately 7% even though, on the average, more processors are used than with BERD. This is because BERD executes Q_B in two sequential steps: First, Q_B is directed to a single processor containing the relevant fragments of its auxiliary relation in order to determine which processors contain the 10 qualifying tuples. Next, Q_B is directed to these processors found in the first step. The first step of BERD is a sequential process that is likely to compete for resources with the other queries executing concurrently on the same processor. Every time this happens, the execution time of the query increases significantly, reducing the throughput of the system. MAGIC, however, analyzes the grid directory and directs Q_B to 8 processors and each processor retrieves approximately $\frac{1}{8}$ of the qualifying tuples in parallel.

Both BERD and MAGIC provide a higher throughput than range partitioning because they direct each query to fewer processors, freeing other processors to execute other queries. The number of processors used by BERD increases as the number of tuples that satisfy the predicate of Q_B (the query referencing the secondary partitioning attribute) increases. To demonstrate this, we increased the number of tuples that satisfy the predicate of Q_B from 10 to 20 and measured the

Figure 9: Low-Low Query Mix with a Higher Selectivity

throughput of both BERD and MAGIC. As demonstrated in Figure 9, MAGIC outperforms BERD by 50% at a multiprogramming level of 64.

When there is a high correlation between attribute values A and B, the throughput of the system with both BERD and MAGIC is significantly enhanced (see Figure 8.b). In this case, both MAGIC and BERD direct each query to a single processor, resulting in an ideal execution paradigm for this workload. MAGIC outperforms BERD by approximately 45% at high multiprogramming levels because it does not incur the overhead of accessing the auxiliary relation to determine which processor contains the relevant fragments (it uses the grid directory to compute these processors).

7.2 Low-Moderate Query Mixes

Figure 10.a presents the throughput of the various declustering strategies for the low-moderate query mix when there is low correlation between the partitioning attributes values. In the case of MAGIC, the value of M_i for Q_B is 9 processors (i.e., ideally 9 processors should be utilized to execute Q_B) while M_i for Q_A is one. Using Equation 4 of Section 3.3, the grid file algorithm splits the dimension corresponding to attribute B nine times more frequently than the dimension corresponding attribute A.

Figure 10: Low-Moderate Query Mix

When there is a low correlation between the partitioning attribute values, MAGIC constructs a 23x193 directory. The assignment procedure causes MAGIC to direct Q_A to two and Q_B to sixteen processors. BERD partitioning directs Q_A to 1 processor and Q_B to all 32 processors. (Q_B retrieves 300 tuples and, due to the low correlation between the partitioning attribute values, these tuples are scattered across all the processors.) The range partitioning strategy provides an identical execution paradigm. BERD provides a lower throughput (Figure 10.a) than range declustering because it incurs the overhead of accessing and searching the auxiliary relation.

When there is a high correlation between the partitioning attribute values, both MAGIC and BERD direct each query type to a single processor (see Figure 10.b). Consequently, most resources in the system remain idle at low multiprogramming levels, resulting in a low system throughput. At high multiprogramming levels, both MAGIC and BERD outperform range by a significant margin since they direct both queries to fewer processors and do not waste any system resources. MAGIC outperforms BERD because it does not incur the overhead of searching the auxiliary relation to determine the processors that contain the relevant fragments for executing Q_B.

Figure 11: Moderate-Low Query Mix

7.3 Moderate-Low Query Mixes

Figure 11 presents the throughput of the alternative declustering strategies for the moderate-low query mix. For this query mix, the value of M_i for Q_A is nine processors while that of Q_B is one processor. Thus, MAGIC splits the dimension corresponding to attribute A nine times more frequently than attribute B.

When there is a low correlation between the partitioning attribute values, MAGIC constructs a 193x23 directory. This causes the query with low resource requirement (Q_B) to be directed to two processors and the query with moderate resource requirement (Q_A) to sixteen processors. The performance results obtained by the alternative declustering strategies for this case is almost identical to that of Section 7.2. One difference is that BERD outperforms range when the partitioning attribute values are not strongly correlated. For this query mix, the low resource intensive query referencing attribute B retrieves only 10 tuples (compared with 30 in the low-moderate query mix). While the range declustering strategy directs this query to all 32 processors, BERD directs this query to at most 11 processors. Consequently, it minimizes the number of processors utilized for Q_B and provides an overall higher throughput.

The performance of the alternative declustering strategies when there is a high correlation

Figure 12: Moderate-Moderate Query Mix

between the partitioning attribute values (see Figure 11.b), is almost identical to that of Section 7.2 and is not described further.

7.4 Moderate-Moderate Query Mix

Figure 12 presents the performance of the alternative declustering strategies for the moderate-moderate workload as a function of the multiprogramming level. When there is a low correlation between the partitioning attribute values, MAGIC outperforms both range and BERD partitioning because MAGIC constructs a 101x91 directory and utilizes an average number of 6.5 processors while both BERD and range partitioning direct Q_A to one processor and Q_B to all thirty-two processors, utilizing 16.5 processors. Moreover, BERD incurs the additional overhead of accessing the auxiliary relation to determine the processors that contain the original tuples.

When there is a high correlation (see Figure 12.b) between the partitioning attribute values, both MAGIC and BERD direct all queries to a single processor. At low multiprogramming levels most resources in the system remain idle resulting in a low system throughput. In fact, range outperforms both MAGIC and BERD at multiprogramming level of one because it distributes the CPU processing of a query across multiple processors reducing the response time of the system.

At high multiprogramming levels, BERD and MAGIC maximize the throughput of the system because: 1) they do not incur the overhead of parallelism when executing each query, and 2) the idle resources are utilized by the additional queries in the system. At multiprogramming level of 64, MAGIC outperforms BERD by approximately 25% because it does not incur the overhead of searching the auxiliary relation.

8 Conclusion

In this paper we compared the performance of two alternative multi-attribute declustering strategies, MAGIC and BERD, with respect to each other and the single attribute range declustering strategy. Three major conclusions can be drawn from this study. First, for a workload consisting of selection queries with range predicates with either a low or moderate resource requirements, MAGIC always provides better throughput than BERD partitioning. This is because BERD incurs the overhead of accessing an auxiliary relation to determine which processors contain relevant fragments. Second, the correlation between the partitioning attributes of a relation can have a significant impact on the performance of each multi-attribute declustering strategy. If there is a high correlation between the partitioning attribute values, then the execution of queries that reference different attributes can be localized to a single processor (or fewer processors than that originally anticipated). Localizing the execution of queries with minimal resource requirements to a single processor increases the overall processing capability of the system as the "freed" processors can execute other queries. In addition, such localization can reduce the response time of a query as the overhead of scheduling and terminating the query on multiple processors is reduced. Finally, MAGIC outperformed the range declustering strategy in all the conducted experiments.

9 Acknowledgments

We would like to thank Michael Carey both for his very constructive comments on earlier (and less readable) drafts of this document and for his help in giving MAGIC declustering its name. We would also like to thank Miron Livny for the DeNet simulation language and Hui Hsiao for his validated simulation model of the Gamma database machine. This research was partially supported by the USC Faculty Research and Innovation funds, by the National Science Foundation under grant IRI-9110522, 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.

References

[BAC+90] H. Boral, W. Alexander, L. Clay, G. Copeland, S. Danforth, M. Franklin, B. Hart, M. Smith, and P. Valduriez. Prototyping Bubba, a highly parallel database system. *IEEE Transactions on Knowledge and Data Engineering*, March 1990.

[BDC83] C. Bitton, D. DeWitt, and C.Turbyfill. Benchmarking database systems - a systematic approach. In *Proceedings of the 1983 VLDB Conference*, October 1983.

[CABK88] G. Copeland, W. Alexander, E. Boughter, and T. Keller. Data placement in bubba. *Procedings of the ACM-SIGMOD International Conference on Management of Data*, June 1988.

[DGS+90] D. DeWitt, S. Ghandeharizadeh, D. Schneider, A. Bricker, H. Hsiao, and R. Rasmussen. The Gamma database machine project. *IEEE Transactions on Knowledge and Data Engineering*, March 1990.

[GD90] S. Ghandeharizadeh and D. DeWitt. A multiuser performance analysis of alternative declustering strategies. In *Proceedings of the 6th Data Engineering Conference*, February 1990.

[Gha90] S. Ghandeharizadeh. *Physical Database Design in Multiprocessor Systems.* PhD thesis, University of Wisconsin - Madison, 1990.

[GMSY90] S. Ghandeharizade, R. R Meyer, G. L. Schultz, and J. Yackel. Optimal balance partitions and a parallel database application. *Submitted for publication*, 1990.

[Gra89] G. Graefe. Volcano: An extensible and parallel dataflow query processing system. Computer science technical report, Oregon Graduate Center, Beaverton, OR, June 1989.

[Hsi90] Hui Hsiao. *Availibility in Multiprocessor Database machines.* PhD thesis, University of Wisconsin - Madison, 1990.

[Liv88] M. Livny. *DeNet User's Guide.* Computer Sciences Department, University of Wisconsin, Madison, 1988.

[LKB87] M. Livny, S. Khoshafian, and H. Boral. Multi-disk management algorithms. In *Proceedings of the 1987 ACM SIGMETRICS Int'l Conf. on Measurement and Modeling of Computer Systems*, May 1987.

[NHS84] J. Nievergelt, J. Hinterberger, and K.C Sevcik. The grid file: An adaptable, symetric multikey file structure. In *ACM transaction on Databases*, volume 9:1, March 1984.

[RE87] D. Ries and R. Epstein. Evaluation of distribution criteria for distributed database systems. UCB/ERL Technical Report M78/22, UC Berkeley, May 1987.

[Sch90] Donovan Schenider. *Complex Query Processing in Multiprocessor Database machines.* PhD thesis, University of Wisconsin - Madison, 1990.

[SPO88] M. Stonebraker, D. Patterson, and J. Ousterhout. The design of XPRS. In *Proceedings of the 1988 VLDB Conference*, Los Angeles, CA, September 1988.

[Tan88] Tandem Performance Group. A benchmark non-stop SQL on the debit credit transaction. In *Proceedings of the 1988 SIGMOND Conference*, Chicago, IL, June 1988.

[Ter85] Teradata Corp. *DBC/1012 Data Base Computer System Manual*, November 1985. Teredata Corp. Document No. C10-0001-02, Release 2.0.

[TP72] T. Teorey and T. Pinkerton. A comparative analysis of disk scheduling policies. In *Communications of ACM*, volume 15:3, March 1972.