Data Placement In Bubba

George Copeland
William Alexander
Ellen Boughter
Tom Keller

MCC
3500 West Balcones Center Drive
Austin, Texas 78759

Abstract

This paper examines the problem of data placement in Bubba, a highly-parallel system for data-intensive applications being developed at MCC. "Highly-parallel" implies that load balancing is a critical performance issue. "Data-intensive" means data is so large that operations should be executed where the data resides. As a result, data placement becomes a critical performance issue.

In general, determining the optimal placement of data across processing nodes for performance is a difficult problem. We describe our heuristic approach to solving the data placement problem in Bubba. We then present experimental results using a specific workload to provide insight into the problem. Several researchers have argued the benefits of declustering (i.e., spreading each base relation over many nodes). We show that as declustering is increased, load balancing continues to improve. However, for transactions involving complex joins, further declustering reduces throughput because of communications, startup and termination overhead.

We argue that data placement, especially declustering, in a highly-parallel system must be considered early in the design, so that mechanisms can be included for supporting variable declustering, for minimizing the most significant overheads associated with large-scale declustering, and for gathering the required statistics.

1 Introduction

Throughput in parallel systems is determined both by the amount of total work to be done and by the total amount of processing capability wasted by poor load balancing. Maximizing the former while minimizing the latter leads to the following design dilemma. Minimizing total work usually leads to a sequential processing design, because parallel processing adds the overheads of communications, startup and termination. Contrarily, minimizing aggregate processor idle time is usually achieved by designs which spread each operation over all processing elements, usually guaranteeing excessive additional work. Neither of these extremes leads to maximum throughput in a parallel processing design. Instead, some compromise must be reached, which balances the gain in processing capacity due to parallel execution against the overhead caused by parallel execution.

Bubba is a highly-parallel architecture for data-intensive applications. By "data-intensive" we mean any system in which the large size of the base data causes significant storage and performance problems. In such systems, it is usually cheaper to move intermediate results over the interconnect between physical nodes than to move base data, because base data is typically much larger and because it is not always needed in its entirety. Thus, a Bubba design principle is to perform work involving base data at those physical nodes upon which the base data resides. This means that data placement is a critical factor in achieving load balancing and throughput.

This paper examines the problem of data placement in Bubba, which includes three related decisions: the number of nodes over which to partition base data, the particular nodes in which to place base data, and whether to place the data on disk or cache it permanently in memory. The difficulty of this problem dictates using a heuristic approach.

Effective data placement is crucial to performance for any database system having multiple disks, since it is an important lever for load balancing. This problem is usually addressed today by examining file reference frequencies on a periodic basis and manually moving files to achieve a more uniform load. An important exception to this conventional mode of operation is found in the Teradata system. Teradata is a highly-parallel database machine which employs **full declustering**, in which each relation is spread equally among all of the disk drives in the system, via a hash function. [Ter85] describes this scheme but does not describe its performance implications.

[DeW86] used full declustering in the GAMMA parallel database machine. [DeW87] measured single-transaction-at-a-time response time, but did not measure throughput.

[Liv87] studied the relative performance of a data declustering strategy versus no declustering on a conventional multiple-disk file system. They compared the strategy of leaving a file intact on one disk versus full declustering. From their investigation of both strategies across a spectrum of multi-transaction workloads, they concluded that, except under extremely high utilization conditions, full declustering was consistently a better approach to take.

[Tan87] measured the throughput of the DebitCredit [Ano85] workload using full declustering for all relations except for the history file and log, which used a *DegDecl* (the number of nodes containing a relation) of 4. They showed that throughput increased linearly with the total number of nodes for up to 32 nodes. This workload generated fairly uniform accesses to the entire data space.

Our study extends their findings. For some operations, the amount of work increases nonlinearly with *DegDecl*. This is the commonly encountered bane of parallel processing (e.g., [Hwa84], [Vrs85], [Cve87]). In these cases, less than full declustering outperforms both no declustering and full declustering. With full declustering, the placement problem is trivial because each base relation is spread across every available disk. [Ter87], [DeW87], [Liv87] and [Tan87] did not study the impact of different placements of the subfiles.

Section 2 describes the Bubba design environment. Section 3 describes how we deal with locality in Bubba. Section 4 describes the data placement problem in Bubba in more detail. Section 5 describes our data-placement heuristics. Section 6 describes our performance model, an experiment and the insight it provides. Section 7 provides a summary and our conclusions.

2 The Design Environment

This section describes the Bubba design environment, including our performance goals, our hardware organization, our techniques for efficiently supporting declustering, the constraints which our data recovery technique puts on data placement, and our benchmark workload.

Bubba is being designed to run the database and knowledgebase workloads that we envision for the mid 1990's. Its performance objective is to deliver knowledgebase management functionality with cost and performance improvements between one and two orders of magnitude relative to conventional general-purpose computers of the mid 1990s.

2.1 Hardware Organization

Bubba is a highly-parallel machine for data-intensive applications. It is designed to be scalable from 50 to 1,000 **intelligent repositories (IRs)**. Each IR has a main processor, a disk controller, a communications processor, a large main memory, and a disk. Its design philosophy is shared-nothing [Ter85] [Sto86] [DeW86]; neither memory nor disks are shared between IRs. Bubba also has several **interface processors (IPs)** to handle interaction with users and some centralized functions. The IRs and IPs are connected by an interconnect, so that any IR or IP may send messages to any other IR or IP. This interconnect is the only shared resource; it makes a network out of the set of IRs and IPs. This network is a single machine; the IRs and IPs are physically close and message delays are small. The high-level architecture is illustrated in Figure 2.1.

2.2 Architectural Support For Declustering

Bubba includes the following mechanisms to efficiently support declustering:

An efficient global directory mechanism replicated in

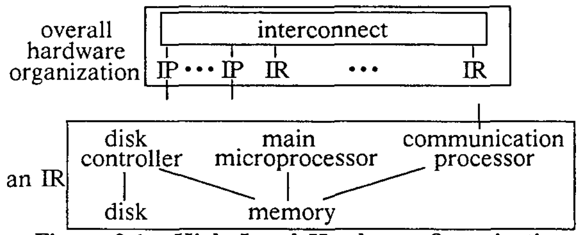

Figure 2.1: High-Level Hardware Organization

each IR and IP describes which IRs contain which parts of each base relation. This supports variable declustering, where each base relation has a different *DegDecl*. The global directory is used to route messages to only those IRs which could be involved in a particular operation.

Inverted files are declustered according to their inverted attribute and have global directory indexes. This allows a relation to be accessed using a single inverted attribute value without involving all *DegDecl* IRs. Instead, only a single inverted-file IR and only those IRs containing records with the inverted attribute are involved.

A dynamic loading and activation mechanism is employed [AlC88], which starts up a transaction on an IR only after the first message has arrived. This minimizes program loading, startup and termination overhead by involving only those IRs that are actually required by that transaction.

Multiple dataflow-control mechanisms are employed [Ale88, AlC88], which inform each dataflow operation that it has received all of its incoming data. The choice among these is made by the compiler based on which causes the least communications overhead.

Taken together, these mechanisms allow startup and termination cost to be O(1) for rifle-shot transactions using inverted attributes as keys, as well as clustered attributes, rather than O(*DegDecl*).

We have put considerable effort into implementing a streamlined communications protocol which reduces the cost of each message over the Bubba interconnect.

2.3 Recovery And Availability Constraints

Recovery from media or IR failure requires maintaining data redundantly on different media or IRs. Many systems maintain a checkpoint and log for this purpose [Gra78]. For higher availability, some systems maintain two identical on-line copies of each conceptual relation and their indexes, where updates are sent to both copies and reads can use either copy. Mirroring as used by Tandem [Kat78] is one example of this, where each disk has a twin containing the same data and indexes. Teradata [Ter85] uses a hash technique to decluster two copies of each relation across the same set of nodes, which has the property that the two copies of each record are guaranteed to each be in different nodes. Both of these identical-copy techniques double the size of both the base data and their indexes and require updates to be applied to both copies.

Each of these three recovery techniques can be supported on Bubba, the choice being determined by the availability requirements of the application. In addition, we support a novel inverted-file (IF) technique that exploits the data redundancy already present in the inverted files. One copy of the conceptual relation is a base relation called the **direct copy**, which has the same structure as its conceptual relation. Another copy of the conceptual relation is its set of inverted files and a remainder relation (containing all data which is not inverted), which together we call the **IF copy**. Either copy can be created from the other. The IF technique strikes a compromise between the previous techniques by requiring less storage and updating than the identical-copy techniques provided there is at least one inverted file, but having a recovery time between that of the checkpoint-and-logging and the identical-copy techniques.

Each of these techniques constrains declustering differently. The two identical-copy techniques require different transaction scheduling and execution strategies to achieve load balancing of reads on the two copies. The IF technique requires that the two copies be declustered over different sets of IRs, which puts the following constraint on the placement of each conceptual relation: $CDegDecl = DDegDecl + IFDegDecl \leq$ total number of IRs, where *CDegDecl*, *DDegDecl* and *IFDegDecl* are the *DegDecl* for the conceptual relation, the direct copy and the IF copy.

Our treatment of data placement in this paper assumes the IF technique, although the results also apply qualitatively when other recovery techniques are used.

2.4 The Order_Entry Workload

We use database application programs based on a conventional order-entry system as our workload. Five transactions comprise our workload: **Payment**, **Order-Shipped**, **New-Order**, **Suggested-Order** and **Store-Layout**, plus a mix of the five transactions, **Mixed**, forming a composite workload. These transactions and their workload characterizations are described in [Ale87]. The order-entry database consists of 36 base relations; 8 of these are direct-copy relations and the remainder are their IF-copy relations. The total size of the database is 160 Gbytes. The five transactions may be summarized as follows:

1) New_Order records a customer's order for an average of 10 different items after the customer's new outstanding balance is checked against his credit limit.
2) Order_Shipped generates an invoice for the customer after an order is filled.
3) Payment retrieves and updates the date-paid for an order and adjusts the associated customer, salesman, district and company sales totals.
4) Suggested_Order infers the number of items to order from suppliers in order to keep a warehouse sufficiently well-stocked.
5) Store_Layout assists a customer in configuring the layout of items on the shelves in the store in an attempt to maximize customer profit.

New_Order, Order_Shipped and Payment represent conventional database transactions, whereas Suggested_Order and Store_Layout represent knowledgebase queries. These transactions require roughly from 2x to 1,000x the work involved in the DebitCredit transaction. *All five transactions are executed on the same database using the same data placement, so that only a compromise data placement is possible.* We think that workloads with the kind of mix of simple and complex transactions seen in Order_Entry will gradually replace simpler workloads like DebitCredit, largely because of the availability of reasonably priced systems with the performance needed to support them [Sam87].

For each transaction type in the Order_Entry workload, Figure 2.2 provides the relative frequency of each transaction in the Mixed workload (*TransFreq*) and the complexity measures (as a function of *DegDecl*) of the costs of the database operations (e.g., joins), communication, and startup and termination as implemented on Bubba. The first three transactions contain "rifle shot" operations (i.e., select operations involving a small number of records). Suggested_Order and Store_Layout contain 1-M and N-M joins, where N and M are the number of records in the two joined relations. Both Suggested_Order and Store_Layout require exhaustive scans of a large percentage of the database without using an inverted file. Their *TransFreq*s are set artificially high because they are intended to be representative of many different "large" transactions, each of which we expect to be run infrequently. Commit and logging overhead are included in each of these five transactions.

transaction	*Trans Freq*	operation type	commun. cost	start/term. cost
New_Order	0.332	rifle shot	O(1)	O(1)
Order_Ship.	0.332	rifle shot	O(1)	O(1)
Payment	0.332	rifle shot	O(1)	O(1)
Sugg._Order	0.001	1-M join	O(*DegDecl*)	O(*DegDecl*)
Store_Layout	0.003	N-M join	O($DegDecl^2$)	O(*DegDecl*)

Figure 2.2: Characterization Of Order_Entry Workload

3 Locality

This section describes how we address locality in Bubba and describes the locality in the Order_Entry workload.

3.1 What Is Locality?

Various formal definitions of locality have been proposed in the literature [Bun84]. They all pertain to the *shape* of the curve of cumulative access frequency of objects (ordered by decreasing frequency) plotted against the cumulative fraction of the objects. This fraction can be based on either object size in bytes (which we call **temperature locality**) or number of objects (which we call **heat locality**). We refer to this type of curve as a **locality curve**. Examples are Figures 3.1 and 3.2, which are described more fully later. **High** locality means a steeply rising curve, and **low** locality means a slowly rising curve. We use the following definitions:

- The **heat** of an object is the access frequency of the object over some period of time.
- The **size** of an object is the number of bytes of the object.
- The **temperature** of an object is the object's heat divided by the object's size.
- **Record heat** (or **temperature**) **locality** is a measure of the nonuniformity in heat (or temperature) among individual records within a base relation.
- **Block heat** (or **temperature**) **locality** is a measure of the nonuniformity in heat (or temperature) among individual storage blocks within a base relation.
- **Relation heat** (or **temperature**) **locality** is a measure of the nonuniformity in heat (or temperature) among individual base relations.

To help provide some intuition, we use an analogy between our terms (heat, size and temperature) and terms from physics (heat, mass and temperature). Heats and sizes are each additive. That is,

$$H_{total} = H_1 + H_2 \quad \text{and} \quad S_{total} = S_1 + S_2 \ .$$

However, because temperatures measure intensity, they are not directly additive. Two temperatures are added by adding their corresponding heats and sizes, so that

$$\text{if} \quad T_1 = \frac{H_1}{S_1} \text{ and } T_2 = \frac{H_2}{S_2} \quad \text{then} \quad T_{total} = \frac{H_1 + H_2}{S_1 + S_2} \ .$$

Both record and relation localities vary widely depending on the application but are independent of database system architecture. Block locality is dependent on both record locality and the strategy used by the database system architecture to place records onto blocks. The strategies used by a database system for placing records onto blocks and for placing blocks onto IRs are linked, so that these choices must be made together.

3.2 Dealing With Locality In Bubba

Bubba uses four techniques for data placement: declustering, relation assignment, our clustering and indexing strategy, and caching.

Declustering [Liv87] partitions the records of each relation into *DegDecl* segments. We decluster records into segments according to either the key value or its hash. When using the key value, we equally partition the sorted relation across its *DegDecl* IRs based on balanced heat rather than size. The advantage of declustering is improved throughput due to load balancing. In general, as *DegDecl* is increased, load is more balanced (increasing throughput) but with diminishing returns, and the overhead due to communications, startup and termination increases (reducing throughput).

Relation assignment places the set of relations, each having *DegDecl* segments, onto the IRs in such a way that the total heat of each IR is roughly equal. Relation assignment and declustering are complementary techniques. Relation assignment becomes easier as *DegDecl* is increased and is trivial with full declustering.

Random-based declustering *deliberately destroys* most potential block locality due to record locality in order to buy better load balancing, so that only relation locality matters. The only cases for which block locality is not destroyed are 1) for records which are so hot that they cause their block to have a high temperature, and 2) for relations indexed on a key value that is strongly correlated with heat (i.e., history files [Ano85]).

Some clustering and indexing strategies place records onto blocks according to relative heat [Flo78, Jak80, Omi83, Yu85]. These strategies more fully exploit record locality by concentrating the hottest records into relatively few, cacheable blocks. However, these schemes would improve data cache efficiency at the expense of load balancing in a parallel environment and of index cache efficiency. To whatever extent they are successful in clustering hot records in the same storage block for better data cache efficiency, load balancing across multiple IRs will be hurt. In addition, index cache efficiency is significantly reduced because much larger indexes must be used. Placing data into IRs and blocks according to key values allows cluster indexes to be used for key attributes instead of inverted indexes. Cluster indexes are much smaller than inverted indexes, because they require only IR-level or block-level resolution instead of record-level resolution, with the advantage that they can be cached.

Our **clustering and indexing strategy** within an IR is to use a cluster index using either the key value or its hash. With random-based declustering, the advantage of heat clustering within each IR is greatly diminished, but its cost in terms of index size is not diminished. This approach reduces data cache efficiency to the extent there is still some record locality within each IR after declustering, but significantly increases index cache efficiency because indexes are reduced by several orders of magnitude.

Caching has the obvious advantage of reducing disk IO, given that temperature locality exists. Bubba adopts the approach that a base relation is either entirely cache resident or entirely disk resident. The reasons are 1) that we force block locality to be low via declustering to achieve load balancing, and 2) that an optimizer can exploit this knowledge of cache residency by having more precise cost functions in its optimizer and by using memory-resident algorithms. Relations are cached based on temperature rather than heat, so that memory space is considered in addition to memory bandwidth. The optimum total amount of cache memory for a Bubba configuration can be determined by an analysis similar to the "5-minute rule" in [Gra87]. For a given configuration, the highest temperature relations are cached. We use conventional LRU to buffer blocks of non-cached relations, which handles the two cases described above (i.e., extremely hot records and correlations between key value and heat), as well as transient hot spots. Buffering of temporary data is handled using a separate mechanism.

Although Bubba destroys most block locality, relation locality is still quite significant in Bubba. In general, high relation temperature locality helps data cache efficiency, whereas high relation heat locality hurts load balancing among IRs.

3.3 Locality In Order_Entry

Figures 3.1 and 3.2 indicate that both temperature and heat locality for the Order_Entry Mixed workload are quite high. High temperature locality means that only a small data cache is needed to reap most of the benefits of caching. For example, a hit ratio of about 93% is possible with only about 0.6% of the database cached. High heat locality means that load balancing will be quite difficult, because the relatively high heats of some relations mean that placement of these very hot relations will determine overall load balancing. If these relations are declustered over a small number of IRs, then load balancing will be more difficult to achieve. Thus, for the Order_Entry workload, it is quite easy to obtain efficient caching but quite difficult to achieve efficient load balancing.

For the sake of comparison, Figures 3.1 and 3.2 also contain "theoretical" curves for three localities: 50/50 (no locality), 70/30 and 90/10. For example, 70/30 locality means that 70% of the logical accesses are to 30% of the relations. Numerous formulas can satisfy an "α/β" rule, where $0 \le \beta \le 0.5 \le \alpha \le 1$. We plotted the following formulas for α/β temperature and heat localities, which have the property that the α/β rule can be applied at any point on the curve:

$$\frac{\text{CumulativeHeat}}{\text{Total Heat}} = \left(\frac{\text{CumulativeSize}}{\text{Total Size}}\right)^{\frac{\log \alpha}{\log \beta}}$$

$$\frac{\text{CumulativeHeat}}{\text{Total Heat}} = \left(\frac{\text{CumulativeNumber}}{\text{Total Number}}\right)^{\frac{\log \alpha}{\log \beta}}.$$

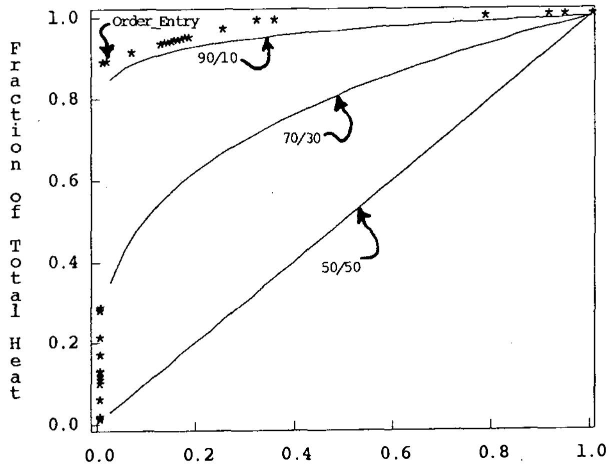

Figure 3.1: Temperature Locality of the Mixed Workload

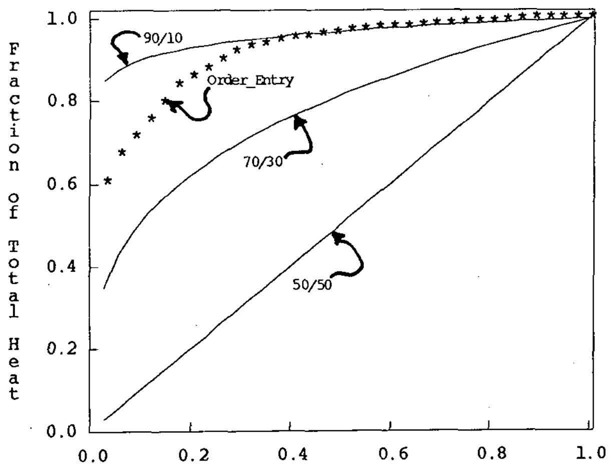

Figure 3.2: Heat Locality of the Mixed Workload

4 Problem Statement

Informally, the problem is to choose how many IRs to spread each relation over (i.e., its *DegDecl*), choose which IRs to place each relation on (i.e., relation assignment), and decide which relations to cache, so as to maximize overall throughput in the system.

Our measure of throughput in this study is total transactions completed per second. We will calculate the throughput on the Mixed workload and on "pure" workloads consisting of only one of the Order-Entry transaction types *given a placement aimed at maximizing throughput on the* **Mixed** *workload*. Although transaction response time is a very important performance metric, we ignore it in this study except to constrain processor utilization at each IR to be no greater than 0.95 and disk utilization to be no greater than 0.50 in our throughput model. We did not including response time because response time distributions cannot be obtained from an analytic model.

4.1 Initial Placement

The placement problem could be stated formally as an optimization problem. The objective is to maximize throughput as defined by a certain analytic queueing model (described in Section 6). There are six constraints:

1) All of the inverted files and the remainder relation of the same conceptual relation should be on the same IRs. This placement of the IF copy as a unit helps make the IF recovery technique used on Bubba more efficient.
2) No part of the direct copy of a relation may be on the same IR as its IF copy. This is required by the IF recovery technique.
3) No relation may be declustered over more than *CDegDecl* IRs. *CDegDecl* is an experiment parameter.
4) No relation may be declustered over more IRs than its number of records.
5) The number of relations that may be cached is limited by the size of the memory at each IR.
6) The number of relations that may be placed on the disks is limited by the size of the disk at each IR.

[Muk87] shows that many similar problems (e.g., [Eas74], [Chu69] and [Mah76]) are NP-complete. Consequently, we attack the problem by heuristic methods and investigate their performance using a model.

4.2 Reorganization

There is always some ideal data placement for the workload which is executing at a particular instant in time. Usually, this ideal placement changes continuously as the workload changes in time. Whenever the actual placement is not ideal, there is a benefit in reorganizing. Unfortunately, there is a cost involved in reorganizing. Obviously, reorganization should take place only when the benefit outweighs the cost. The reorganization problem is to decide what and when to reorganize. The only constraint, in addition to those listed for initial placement, is that the reorganization costs should be less than the anticipated gain in performance.

5 Placement Heuristics

This section describes our heuristics for initial data placement and for reorganization.

5.1 Motivation

Ideally, in order to balance load by placing data, we need to know the computational load that will be associated with each relation. For each conceptual relation, we need to know the relative frequency of accesses to each copy of that relation by the workload. Then, for example, given the copy's placement in cache or on disk, the ratio of reads to writes, the physical schema and some characteristics of the hardware, we can compute the rate of disk accesses associated with that copy of that relation.

However, disk IO is not the only component of work that needs to be balanced among IRs. In shared-nothing systems, we must also balance processor load and inter-IR messages. To compute these from the relative access frequency on a relation, we also need to know the *kind* of access. For example, a join on a relation typically requires more processor instructions than a select, and an M-to-N join generates more messages over the interconnect than a 1-to-N join.

Further, the relations will have associated with them different kinds of load in different proportions, so that placing data to balance one kind of load will often unbalance others. We would therefore have to associate with each copy of each conceptual relation a composite load defined as a weighted sum of the different kinds of load.

While it might be possible to define such a composite load and to experimentally arrive at the best weights to maximize throughput, in practice it would be very expensive to collect and maintain workload statistics on the type of database operations performed on each relation. In contrast, it would be relatively easy to record only accesses. In fact, an estimate of the number of accesses to each copy of a relation by a given transaction could be supplied by a transaction compiler.

These considerations led us to investigate whether satisfactory initial placements could be found using only statistics that are inexpensive to gather.

5.2 Gathering The Required Statistics

Our algorithms use the following statistics:

1) The frequency of each transaction t ($TransFreq_t$) is measured directly in the IPs.
2) For each transaction t, the compiler's optimizer provides estimates of the heat for each base relation r ($TransRelHeat_{tr}$).
3) The size of each base relation r ($RelSize_r$) is measured directly in the IRs.
4) For the Mixed workload, the heat ($MixedRelHeat_r$) and temperature ($MixedRelTemp_r$) of each base relation r are calculated by the following formulas:

$$MixedRelHeat_r = \sum_t (TransRelHeat_{tr} * TransFreq_t)$$
$$MixedRelTemp_r = MixedRelHeat_r \,/\, RelSize_r \,.$$

5) For each IR i, the utilization of the processors ($IRProcUtil_i$) and the disk ($IRDiskUtil_i$) are measured directly in the IR.

All of these statistics must be periodically communicated to a central placement node.

5.3 An Algorithm For Initial Placement

The unit of placement is an entire copy of a relation. We base placement of relations solely on their heat, temperature and size.

Recall that all base relations of the IF copy of a conceptual relation are treated as a unit and placed together. The heat of the unit is the sum of the individual heats, and the temperature of the unit is the sum of the heats divided by the sum of the sizes.

$$\begin{aligned} MixedRelHeat_{IFCopy} &= \sum MixedRelHeat_{IF} \\ &\quad + MixedRelHeat_{remainder} \\ RelSize_{IFCopy} &= \sum RelSize_{IF} + RelSize_{remainder} \\ MixedRelTemp_{IFCopy} &= \frac{MixedRelHeat_{IFCopy}}{RelSize_{IFCopy}} \,. \end{aligned}$$

First, we decide how many IRs to decluster each relation over. There is a *CDegDecl* for each conceptual relation, which is the maximum number of IRs over which any conceptual relation's direct copy plus its IF copy may be declustered. Within the *CDegDecl* IRs, the direct copy and the IF copy are declustered according to the ratio of their heats:

$$\begin{aligned} DDegDecl &= \frac{MixedRelHeat_{IFCopy}}{MixedRelHeat_{direct}} * CDegDecl \\ IFDegDecl &= CDegDecl - DDegDecl \,. \end{aligned}$$

This declustering calculation is made for each conceptual relation independently.

Next, we decide on which IRs to place relations (i.e., relation assignment). We do this in two steps, first placing hot base relations in caches until the caches are full, and then placing the remaining base relations on the disks. We order the base relations according to temperature, again treating all base relations of an IF copy as a single unit. Suppose that the highest-temperature unplaced base relation, u, is to be declustered over $DegDecl_u$ IRs. We try to find $DegDecl_u$ IRs that have room in their cache for $1/DegDecl_u$ of u and that do not already contain the other copy of u; if there are more than $DegDecl_u$ such IRs, we choose the ones with the least accumulated heat in the data already placed there. If there are less than $DegDecl_u$ IRs that can take u, we continue processing the list of unplaced relations, looking for other base relations that might fit in the remaining unused cache space.

When no more base relations can be placed in cache, we place the remaining ones on disk. First, we order these remaining base relations according to heat, again treating all base relations of an IF copy as a single unit. Suppose that the highest-heat unplaced base relation, u, is to be declustered over $DegDecl_u$ IRs. We look for $DegDecl_u$ IRs that have room on their disks and that do not already contain the other copy of u. If there are more than $DegDecl_u$ such IRs, we choose the ones with the least accumulated heat in the data already placed there, either in cache or on disk. If there are less than $DegDecl_u$ IRs that can take u, the placement algorithm fails.

In both cache and disk relation assignment, we try to balance for each single transaction when doing so makes little difference to Mixed-workload balancing.

The algorithm described above is extremely simple and computationally very cheap. It involves no backtracking. It requires only two facts about a base relation, its size and heat. The algorithm does not guarantee perfect balancing of processing work, disk IOs or message traffic; however, it has done reasonably well in all the configuration combinations of hardware and workload that we have modeled.

5.4 An Algorithm For Reorganization

Several researchers have investigated the optimal reorganization interval for structures in database systems that grow in size, such as checkpoint files and index files, causing performance to deteriorate monotonically in time [Shn73, Yao76, Mar76, Tue78, Soc79, Bat82]. The data-placement reorganization problem in Bubba differs from these in two ways. First, performance due to poor data placement does not necessarily deteriorate monotonically; it can sometimes be temporary and even cyclic. Secondly, reorganization costs are quite different in nature and typically higher. Thus, we cannot determine an optimal reorganization interval.

We are concerned with both cache and distributed (i.e., *DegDecl* and relation assignment) reorganization. The cost of cache reorganization can be reduced using a "lazy" technique, which exchanges blocks only when blocks of the promoted data are actually needed or when the system is idle. A potentially high cost of cache reorganization is that our optimizer exploits the knowledge of which relations are cached, so that programs which reference a reorganized relation must either be recompiled or suffer degraded performance. The cost of distributed reorganization is also quite high, requiring the movement of large amounts of data. While this data movement could be done in the background using the copying scheme proposed by [Att84], throughput would be severely degraded in a heavily-loaded system. These high costs and the fact that performance does not deteriorate monotonically dictate that data-placement reorganization should be infrequent and should involve only a few bottleneck relations, and that statistics should be averaged over a relatively long time period.

Bubba's reorganization algorithm attempts to balance the cost of reorganization with the expected performance improvement. The first step in our algorithm is to choose candidate relations for reorganization. To do this, we examine the IRs with highest *IRProcUtil* and *IRDiskUtil*. Among these bottleneck IRs, we pick the relations with highest MixedRelTemp using the most recent statistics. We use temperature for parallel reorganization, as well as for caching, because smaller relations require less data movement.

The next step is to determine the new candidate placement. To do this, we run the same algorithm described above for initial placement using the most recent statistics, except that all relations other than the candidate relations are already placed.

The final step is to decide whether to reorganize. To do this, we run our throughput model using the most recent statistics for both the current placement and the candidate placement. We reorganize only if the performance improvement is worth the work required to reorganize.

6 Experiment And Results

This section describes our performance model, an experiment and its results. We were interested in investigating the following issues:

1) What *CDegDecl* yields highest throughput?
2) What is the effect of communications costs?
3) What is the effect of startup and termination costs?

6.1 The Performance Model

The metric for evaluating our data placement strategy is transaction throughput. To estimate throughput, we implemented a modeling tool called *FIRM*, which is described in detail in [Bou87]. FIRM consists of three programs: the first performs the data-placement algorithm, the second maps workloads to individual IRs and the third solves the resulting analytic queueing model. Inputs to FIRM are:

1) the Order_Entry workloads,
2) the Bubba architecture and configuration description, and
3) data-placement parameters (e.g., *CDegDecl*).

The important predictions output by FIRM are:

1) maximum practical throughput sustainable for the Bubba configuration,
2) average utilizations for every disk and processor in Bubba,
3) message traffic rate at each IR, and
4) average elapsed time for Bubba to execute particular operations.

In a separate analysis, we found that on-the-wire interconnect bandwidth will not be a bottleneck, so FIRM ignores it. However, the to-wire and from-wire protocol is included in the processor work.

The placement of the data across the IRs is computed in FIRM according to the placement heuristics given in Section 5.3. Once the data placement is determined, the costs of operations (such as a join) can be determined from the number of IRs expected to participate in the operation. These costs are

1) the average number of processor instructions,
2) the number of disk IOs, and
3) counts of the total messages received and the total messages sent.

The placement information and the operation cost information are then used by the second FIRM program that maps the operations of the five transactions onto the multiple IRs. A traffic rate for each transaction type at each IR in Bubba is calculated and fed to the FIRM model solver, which implements an open queueing network model. The model solver accepts the relative traffic rates by component type, scales them by the transaction(s) arrival rate(s) and determines the performance measures listed above for a Bubba configuration with processors and disks rated at specific speeds. The solver uses simple operations analysis techniques [Den78], as presented in [Laz84], to determine these measures. This model always yields optimistic results; however, here we only use it for predicting relative throughput of different placements.

6.2 The Experiment

The following configuration parameters were used in our experiment, chosen to be representative of high-performance technology available today:

1) The total number of IRs is 1024.
2) Each IR has 5 MIPS total processing capacity.
3) Each message send or receive requires 2,000 instructions.
4) Startup and termination of a transaction on an IR requires 5,000 instructions.
5) Each IR has a disk storage capacity of 700 megabytes.
6) The average disk access rate is 40 accesses per second.
7) Each IR has 1.5 megabytes of cache space for base relations.

The Order_Entry database contains some relations that are too large to fit onto small numbers of disks; the minimum number of disks required to hold these large relations is 416. Therefore, we varied *CDegDecl* from 416 to 1024. To isolate the effects of communications costs and startup and termination costs, Figures 6.1 through 6.6 show the following three curves for each of the five transactions alone and the Mixed workload:

1) Including both communications costs and startup and termination costs.
2) Not including communications costs but including startup and termination costs.

3) Including communications costs but not including startup and termination costs.

6.3 Experimental Results

For each workload, the three curves differ only after a processor on some IR becomes the system bottleneck. This is because the overhead of declustering is carried only by the processors, not the disks. As processor speeds continue to increase with respect to disk speeds in the coming decade, this crossover point will move to the right and the overheads associated with declustering will diminish. Notice that the New_Order, Order_Shipped and Payment transactions are disk-bottlenecked throughout the entire range, the Suggested_Order and Store_Layout transactions are processor-bottlenecked throughput the entire range, and the Mixed workload goes from disk-bottlenecked to processor-bottlenecked at *CDegDecl*=736.

The negative slope throughout the Store_Layout curve is caused by its O(*DegDecl*^{2}) message cost, because the curve without message costs does not have an overall negative slope. The O(*DegDecl*) message cost in Suggested_Order never causes a downturn, but did decrease throughput by a constant factor. The O(1) message cost in the other three pure workloads has no effect on throughput because they were each disk-bottlenecked. The O(*DegDecl*^{2}) message cost in Store_Layout causes Mixed-workload throughput to begin decreasing even though its *TransFreq*=0.003, but only after the processor bottleneck begins at *CDegDecl*=736. *This phenomena suggests that we should identify the relations involved in this nonlinear operation (N-M join) and reduce their DegDecl.*

The O(*DegDecl*) startup and termination cost in Suggested_Order and Store_Layout never caused a downturn, but does decrease throughput by a constant factor within each. The constant factor for Suggested_Order is significant because the number of startups is about the same as the number of messages because of the 1-N join. The constant factor for Store_Layout is insignificant because the N-M join causes much join work on each IR.

The DebitCredit transaction [Ano85] is similar to New_Order, Order_Shipped and Payment transactions in that all four have O(1) communications, startup and termination costs, and is most similar to the Payment transaction in total amount of work required. [Tan87] showed that throughput for DebitCredit increased linearly with the total number of IRs for up to 32 IRs. Note that we varied *CDegDecl* rather than the total number of IRs; however, the effect is similar because the bottlenecked relations are effectively given more IRs as *CDegDecl* is increased. Their results are consistent with our results for New_Order, but are not consistent with our results for Order_Shipped and Payment. The main difference is that our placement was optimized for a Mixed workload, not for any single transaction.

Note that the shape these curves would be roughly the same if we uniformly reduced both the total number of IRs and the total database size. The reduced throughput due to O(*DegDecl*^{2}) communications overhead would have happened at correspondingly lower *CDegDecl*. Thus, these results also apply to systems with fewer IRs.

Throughput achieved with the placement algorithm presented in this paper is more than an order of magnitude better than with earlier more naive algorithms. This emphasizes the importance data placement.

7 Summary And Conclusions

The data placement problem boils down to compromising load balancing with overall load reduction, in the face of various kinds of data locality. We have argued for a particular treatment of locality that we believe effectively accomplishes this compromise for shared-nothing data-intensive architectures.

We have shown how data placement is a critical design problem in highly-parallel systems for data-intensive applications. We also illustrated the importance of data placement experimentally using a particular set of data-placement heuristics and a particular workload having high relation locality. Data placement is both more important and more difficult in highly-parallel systems than in centralized systems. While the disk load-balancing problems are similar in both, parallel systems must also consider load balancing and additional overhead for processors.

We argue that load balancing, in particular declustering and reorganization, must be considered early in the design of such a highly-parallel system as Bubba. Mechanisms, such as Bubba's global directory, must be provided to support variable declustering rather than full declustering. Mechanisms, such as declustering inverted files based on their inverted attribute, dynamic loading and activation, and multiple dataflow control techniques, must be provided to minimize the overheads caused by large-scale declustering. A streamlined communications protocol is required to minimize the cost of each message. Mechanisms to efficiently gather statistics and to minimize the cost of reorganization are needed. Different recovery techniques constrain declustering differently and require different ways to achieve load balancing.

Many open issues remain. One is the effect of better predictors of work. Data accesses alone do not accurately predict disk IOs, processor instructions or any other single component of system load. It is especially important to identify accesses involving operations for which work scales nonlinearly with *DegDecl*, and constrain *DegDecl* for those base relations involved in transactions with nonlinear complexity.

Another open issue is whether to include physical schema design as part of our data placement algorithm. This would require more information in the workload characterization. For example, a separate access frequency for each attribute of each relation, instead of only the overall access frequency of each relation, would allow the decision of whether to have an inverted file to be automated.

Another open issue is the effect on throughput of O(*DegDecl*) operations for workloads with less locality than Order_Entry. We speculate that relation assignment for low-locality workloads is less difficult, so that it might pay to use smaller *DegDecl* for relations accessed by O(*DegDecl*) operations, as well as for relations accessed by O(*DegDecl*^{2}) operations.

The most important open issue is how placement affects transaction response time. We are currently implementing Bubba on a commercially available parallel system with 32 IRs. This implementation will be used as

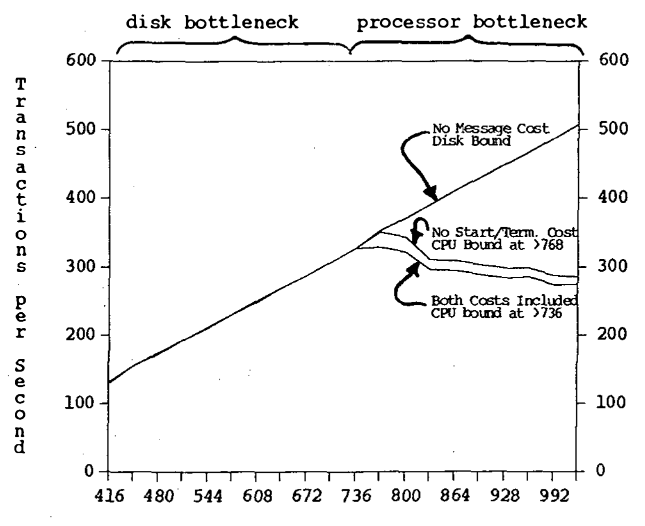

Figure 6.1: Mixed Workload

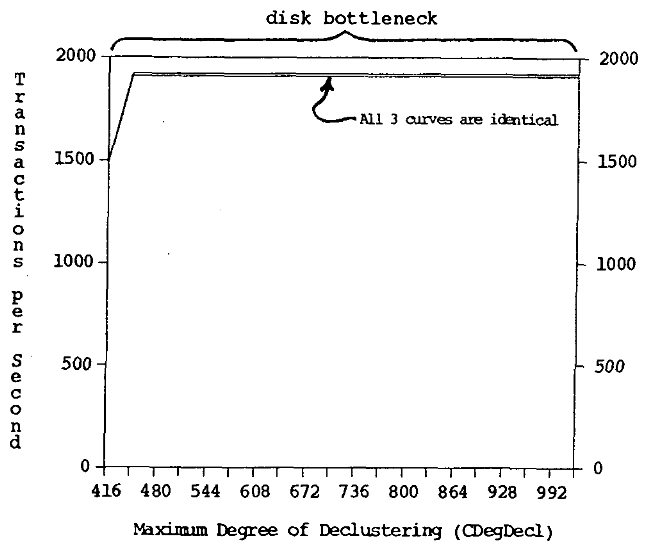

Figure 6.4: Payment Workload

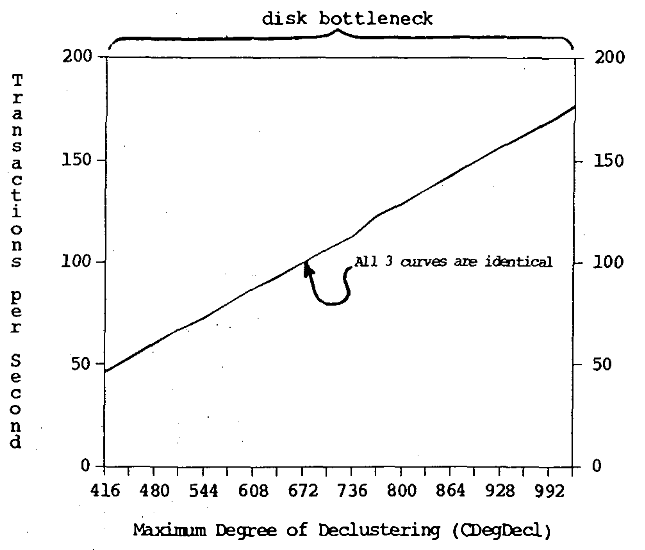

Figure 6.2: New_Order Workload

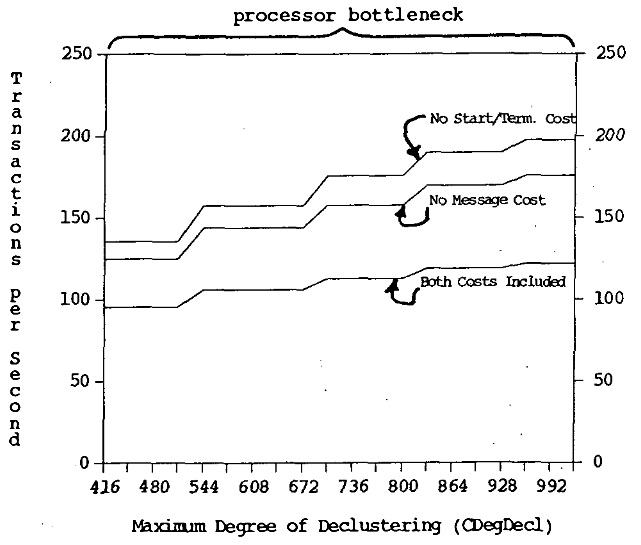

Figure 6.5: Suggested_Order Workload

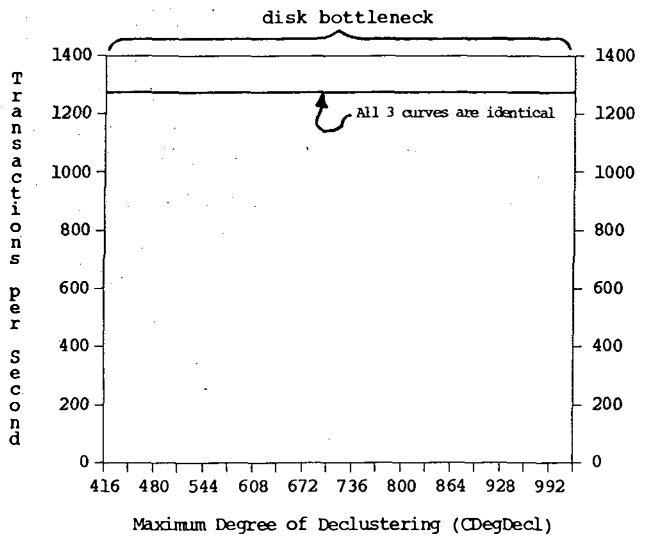

Figure 6.3: Order_Shipped Workload

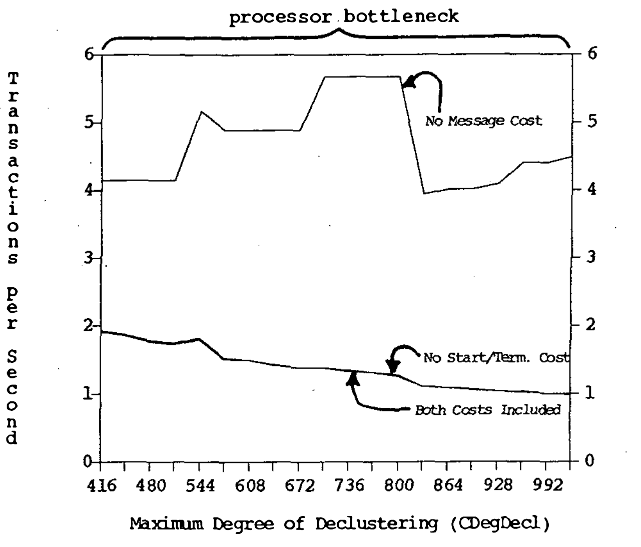

Figure 6.6: Store_Layout Workload

an experimental vehicle to better understand performance characteristics. Of major concern is how response time scales with increasing parallelism via both more IRs and higher *DegDecl*. Based on our own preliminary results and the results in [DeW87], we believe that consideration of response time will make data placement much harder than the version of the problem we have presented in this paper.

Acknowledgements

Thanks to Chris Buckalew who helped code, maintain and run FIRM, and to Jim Gray for his helpful comments.

References

[Ale87] W. Alexander, T. Keller and E. Boughter, "A Workload Characterization Pipeline for Models of Parallel Systems," *ACM SIGMETRICS Conference*, Alberta, Canada (May 1987).

[Ale88] W. Alexander and G. Copeland, "Comparison Of Dataflow Control Techniques In Distributed Data-Intensive Systems," *ACM SIGMETRICS Conference*, Santa Fe, New Mexico (May 1988).

[AlC88] W. Alexander and G. Copeland, "Process And Dataflow Control In Distributed data-Intensive Systems," *ACM SIGMOD Conference*, Chicago (June 1988).

[Ano85] Anon el al, "A Measure Of Transaction Processing Power," *Datamation*, Vol. 31.7 (April 1985).

[Att84] R. Attar, P. Bernstein and N. Goodman, "Site Initialization, Recovery And Backup In A Distributed Database System," *IEEE Transactions on Software Engineering*, Vol. SE-10, No. 6 (November 1984).

[Bat82] D.S. Batory, "Optimal File Designs And Reorganization Points," ACM *TODS*, Vol. 7, No. 1 (March 1982).

[Bou87] E. Boughter, W. Alexander and T. Keller, "A Tool for Performance-Driven Design of Parallel Systems", MCC Tech. Report ACA-ST-312-87 (1987).

[Bun84] R. Bunt, J. Murphy, and S. Majumdar, "A Measure of Program Locality and its Applications," *ACM SIGMETRICS Conference*, Cambridge, Mass. *(May 1984)*.

[Chu69] W. W. Chu, "Multiple File Allocations in a Multiple Computer System," *IEEE Trans. on Computers*, Vol. C-18, No. 10 (October 1969).

[Cve87] Z. Cvetanovic, "The Effects Of Problem Partitioning, Allocation, and Granularity On The Performance Of Multiple-Processor Systems," *IEEE Trans. on Computers*, Vol. C-36, No. 4 (April 1987).

[Den78] Denning, P., Buzen, J., "The Operational Analysis of Queuing Network Models", *ACM Computing Surveys Vol. 10, No. 3 (*September 1978).

[DeW86] D.J. DeWitt, R.H. Gerber, G. Graefe, M.H. Heytens, K.B Kumar and M. Muralikrishna, "GAMMA--A High Performance Dataflow Database Machine," *VLDB Conference*, Japan (August 1986).

[DeW87] D.J. DeWitt, S. Ghandeharizadeh, D. Schneider, R. Jauhari, M. Muralikrishna and A. Sharma, "A Single User Evaluation Of The Gamma Database Machine," Proceedings of the Fifth International Workshop on Database Machines, Japan (October 1987).

[Eas74] K. Eswaran, "Placement of Records in a File and File Allocation in a Computer Network," *Information Processing 74, IFIPS* (1974).

[Flo78] A. Flory, J. Gunther and J. Kouloumdjian, "Database Reorganization By Clustering Methods," *Information Systems*, Vol. 3, No. 1 (1978).

[Gra78] J. Gray, "Notes on Database Operating Systems," IBM Research Laboratory, San Jose, Report RJ2188 (1978).

[Gra87] J.N. Gray and F. Putzolu, "The 5 Minute Rule for Trading Memory for Disc Accesses and the 10 Byte Rule for Trading Memory for CPU Time," *ACM SIGMOD Conference*, San Francisco (May 1987).

[Hwa84] K. Hwang and F. Briggs, *Computer Architecture And Parallel Processing*, McGraw-Hill Pub. Co. (1984).

[Jak80] M. Jakobsson, "Reducing Block Accesses In Inverted Files By Partial Clustering," *Information Systems*, Vol. 5, No. 1 (1980).

[Kat78] J.A. Katzman, "A Fault-Tolerant Computing System," *Eleventh Conference on System Sciences*, Hawaii (January 1978).

[Laz84] E. Lazowska, J. Zahorjan, G. Graham, K. Sevcik, *Quantitative System Performance*, Prentice-Hall (1984).

[Liv87] M. Livny, S. Khoshafian and H. Boral, "Multi-Disk Management," *ACM SIGMETRICS Conference*, Alberta, Canada (1987).

[Mah76] S. Mahmoud and J.S. Riordon, "Optimal Allocation of Resources in Distributed Information Networks", *ACM TODS*, Vol 1, No. 1 (March 1976).

[Mar76] K. Maruyama and S.E. Smith, "Optimal Reorganization Of Distributed Space Disk Files," *Commun. of the ACM*, Vol. 19, No. 11 (November 1976).

[Muk87] R. Mukkamala, "Design of Partially Replicated Distributed Database Systems: An Integrated Methodology," Tech. Report 87-04, Department of Computer Science, University of Iowa (July 1987).

[Omi83] E. Omiecinski and P. Scheuermann, "A Global Approach To Record Clustering and File Reorganization," Technical Report, Department Of EECS, Northwestern University (December 1983).

[Sam87] H.W. Sammer, "Online Stock Trading Systems: Study Of An Application," *IEEE COMPCON*, San Francisco (February 1987).

[Shn73] B. Shneiderman, "Optimum Data Base Reorganization Points," *Commun. of the ACM*, Vol. 16, No. 6 (June 1973).

[Soc79] G.H. Sockut and R.P. Goldberg, "Database Reorganization---Principles And Practices," *ACM Computing Surveys*, Vol. 11, No. 4 (December 1979).

[Sto86] M. Stonebraker, "The Case For Shared Nothing," *Database Engineering Conf.*, Vol. 9, No. 1 (March 1986).

[Tan87] The Tandem Database Group, "NonStop SQL, A Distributed, High-Performance, High-Availability Implementation of SQL," *Workshop on High Performance Transaction Systems*, Asilomar, CA (September 1987).

[Ter85] "DBC/1012 Data Base Computer System Manual, Release 1.3," C10-0001-01, Teradata Corp., Los Angeles (February 1985).

[Tue78] W.G. Tuel, "Optimal Reorganization Points For Linearly Growing Files," *ACM TODS*, Vol. 3, No. 1 (March 1978).

[Vrs85] D. Vrsalovic, E.F. Gehringer, Z.Z. Segal and D.P. Siewiorek, "The Influence Of Parallel Decomposition Strategies On The Performance Of Multiprocessor Systems," *IEEE/ACM Symposium on Computer Architecture*, Boston (June 1985).

[Yao76] S.B. Yao, K.S. Das and T.J. Teorey, "A Dynamic Database Reorganization Algorithm," *ACM TODS*, Vol. 1, No. 2 (June 1976).

[Yu85] C.T. Yu, C.M. Suen, K. Lam and M.K. Siu, "Adaptive Record Clustering," *ACM TODS*, Vol. 10, No. 2 (June 1985).