N94- 33828

A Parallel Data Management System for Large-Scale NASA Datasets

Jaideep Srivastava

Computer Science Department
4-192 EECS Building, 200 Union St. S.E.
University of Minnesota
Minneapolis, MN 55455
srivasta@cs.umn.edu; (612)625-4012

Abstract

The past decade has experienced a phenomenal growth in the amount of data and resultant information generated by NASA's operations and research projects. A key application is the *reprocessing problem* which has been identified to require data management capabilities beyond those available today [PRAT93]. The Intelligent Information Fusion (IIF) system [ROEL91] is an ongoing NASA project which has similar requirements. Deriving our understanding of NASA's future data management needs based on the above, this paper describes an approach to using parallel computer systems (processor and I/O architectures) to develop an efficient parallel database management system to address these needs. Specifically, we propose to investigate issues in low-level record organization and management, complex query processing, and query compilation and scheduling.

1. Problem Understanding: NASA's Future Data Management Needs

The past decade has experienced a phenomenal growth in the amount of raw data and resultant information generated by NASA's operations and research projects [ROEL91]. The need for significant improvement in information technologies to manage, identify, and access this data has been clearly identified [ROEL91, CROM92, CAMP90a, CAMP90b]. This section present's our view of NASA's future data management needs (at least in part). It is based on (i) the description of the *reprocessing problem* given in [PRAT93], (ii) published descriptions of the *Intelligent Information Fusion (IIF)* system [ROEL91], and (iii) miscellaneous NASA publications.

1.1 A View of NASA's Data Management Architecture

Figure 1 shows the schematic of a system architecture where the principal emphasis is on the path data takes, and the transformations it goes through, from sensor collection to the scientific user. This architecture borrows from that of the IIF system [ROEL91]. The aim of this diagram is principally for problem understanding purposes and to establish a context for the subsequent discussion. It is by no means a proposal of what the complete architecture for NASA's data management system should be, and is much wider in scope than that of the present paper.

Sensor data first goes through some very low-level processing to generate 'raw data' [PRAT93] which is stored in a Parallel Raw Data Archive (PRDA). The reprocessing activity creates 'data products' [PRAT93] which are managed by a Parallel Relational Database Management System (PRDBMS). Metadata about both raw data and data products is stored in a Metadata Database (MDB). The three different types of data stores, i.e. the PRDA, PRDB, and MDB, reflect the three basically different types of usage of the data and metadata in such an environment [PRAT93]. The raw data is expected to be used mostly by reprocessing algorithms running on vector supercomputers and massively parallel processors (MPPs), and hence is shown managed by a high-performance file system. Since existing data products can also be inputs to the reprocessing activity [PRAT93], direct access to the Parallel Record Management Layer (PRML) of the PRDBMS by the machines running the reprocessing algorithms is shown. A typical user of the data products is a remote scientist who logs in and *browses* the metadata searching for data relevant to a research project. While most browsing involves interaction with the metadata, the scientist may periodically access data products as well as raw data to identify interesting data. Upon selecting

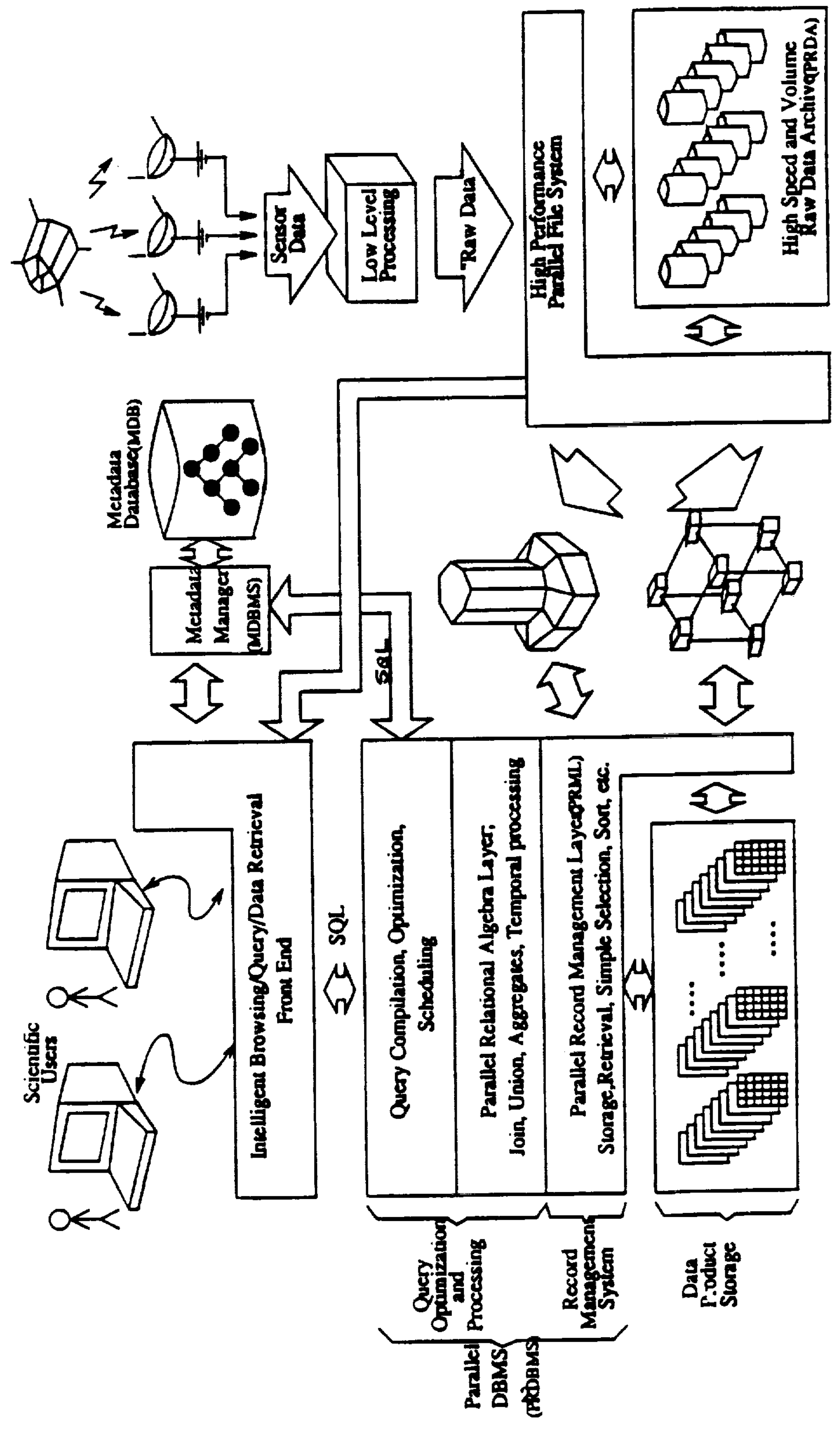

Scientific Users
Intelligent Browsing/Query/Data Retrieval Front End
SQL
Metadata Manager (MDBMS)
Metadata Database(MDB)
SQL
Sensor Data
Low Level Processing
Raw Data
High Performance Parallel File System
High Speed and Volume Raw Data Archive(PRDA)
Query Compilation, Optimization, Scheduling
Parallel Relational Algebra Layer; Join, Union, Aggregates, Temporal processing
Parallel Record Management Layer(PRML) Storage,Retrieval, Simple Selection, Sort, etc.
Query Optimization and Processing
Record Management System
Data Product Storage
Parallel DBMS (PRDBMS)

Figure 1. Our View of NASA Data Management Architecture

the needed data, the appropriate portion is downloaded into the scientists home location. To support such pattern of user behaviour the MDBMS should support large numbers of interactive browsing sessions, each posing mostly small queries against the MDB, interspersed with occasional queries against the PRDA or PRDB. While interactive response time is needed, the bandwidth required is expected to be small during the browsing. Once browsing is complete, the user will issue a series of requests to extract the data to be downloaded to his home location. These requests can be SQL queries to the PRDBMS or file access requests to the PRDA. These requests are expected to have high bandwidth requirements since a large volume of data may be extracted. Since execution times of different plans for a SQL query can differ by a few orders of magnitude, query optimization is critical to ensure both interactive response time and reduced system workload.

1.2 Parallel I/O: Key to NASA's Data Management

Given the volume of data/information in NASA's applications, the use of multiple disks for storage is well accepted. In a database processing environment, the fact that disk I/O is the main bottleneck has been a consensus among researchers. Recent years have seen phenomenal increase in processor speeds, while the 'disk access time' has not shown much improvement, exacerbating the 'access gap' problem. The advent of multiple processor machines has added to this problem. Fortunately, the computer architecture community has started addressing the needs of data intensive applications by developing parallel I/O architectures, e.g. Redundant Array of Inexpensive Disks (RAID) [PATT88] and Disk Arrays [GORD91]. This promises future parallel I/O systems which can feed data to the multiprocessor at a high sustained bandwidth.

Along with the development of parallel I/O hardware, there is a need to develop efficient parallel I/O algorithms to exploit their full potential. The main focus of research in parallel algorithms has been on main memory resident data, where processor parallelism has been of primary concern [LEWI92]. With I/O bandwidth being a principal concern, high performance parallel databases require parallel algorithms for disk resident data. Parallel processing of database operations was first addressed by the database machine community, where the focus was on designing special-purpose hardware [SU86]. No single architecture was found suitable for all database applications, and the cost of building special purpose hardware for specific applications led to only limited success in this direction [DeWI92]. In the past few years there has been renewed interest in looking at database issues for general purpose parallel machines. The availability of a variety of commercial parallel machines, which has eliminated the expense of building special purpose hardware, is in large measure responsible for this [DeWI92].

A crucial factor in our choice of the relational model for the PRDBMS component of the architecture in Figure 1 is that the set-oriented, non-procedural nature of the relational model provides opportunities for massive parallelization [DeWIT92]. This choice is further supported by the fact that the IIF system has already proposed using a relational DBMS for its low-level record management system (LLRMS) [ROEL91].

1.3 Scope of Our Project

Realization of the architecture shown in Figure 1 is a major task and requires research and development in many areas. The scope of our project is limited to addressing problems in the PRDBMS component of the system. Specifically, we address the following problems:

- Data organization, loading, sorting, and retrieval, and index creation and maintenance, in the Parallel Record Management Layer. The proposed solutions must consider that access requests to this layer will be a mix of (i) very high rate of large size access requests from the reprocessing algorithms, and (ii) low to medium to sometimes large size requests from the upper layers of PRDBMS.

- Parallel algorithms to support expensive operations, e.g. join, union, etc., in the Parallel Relational Algebra Layer.

- Compilation and optimization of SQL queries, and resource allocation and scheduling of operators in the resultant plan. Minimizing response time and maximizing throughput will be considered as the optimization criteria.

The rest of the paper is organized as follows: Section 2 presents the technical details of our approach. Section 3 presents a list of goals that must be met, including specific technical problems that must be solved, to make such a system a reality. Section 4 provides the conclusions and section 5 contains the list of references.

2. Technical Details of the Proposed Approach

Our overall goal is to investigate techniques for building a parallel database engine which could fulfill the needs of the PRDBMS component of Figure 1. Following are the key ideas behind our approach:

- Tuples in a relation (or records in a file) are modeled as points in a multi-dimensional space, with each attribute representing an axis.
- This multi-dimensional space can be divided into (overlapping or non-overlapping, nested or non-nested) subdivisions.
- The subdivisions are allocated to different I/O units (e.g. disks) of a parallel computer, with usually many subdivisions going to a single unit, and possibly a single subdivision replicated on multiple units for reliability. This has been termed *declustering* [DEWI90]. The aim is to provide good (close to optimal) load-balancing for query processing.
- New *declustering-aware* parallel algorithms for basic data retrieval operations, e.g. relation/file scan, as well as complex operations, e.g. join and sort, are built to take advantage of the underlying declustering.
- The query compiler/parallelizer/scheduler takes considers architectural parameters and declustering information, in addition to the traditional query and database parameters, in minimizing the execution plan cost. In addition, it generates an initial resource allocation schedule for plan execution.

The remainder of this section is organized as follows: Section 2.1 presents an architecture for the PRDBMS. Sections 2.2 through 2.4 describe our approach to solving specific problems in the *record management, relational algebra*, and *query compilation* layers of the PRDBMS.

2.1 Parallel RDBMS Architecture

As shown in Figure 1, the PRDBMS has a layered architecture. The *parallel record management layer* provides the abstraction of relations/tables which can be created, deleted, populated, sorted, and on which simple selections (predicates involving single relations only) can be performed. This abstraction is used both by the higher layers of the PRDBMS and by the reprocessing algorithms. The *parallel relational algebra layer* contains algorithms for complex operations such as join, union, difference, aggregation, etc. It uses the abstractions provided by the record management layer. The *query compilation layer* provides a declarative interface (SQL) to PRDBMS users (the intelligent front-end and metadata manager in Figure 1), and does the necessary translation and optimization of declarative queries into a sequence of relational algebra operations.

2.2 Parallel Record Management Layer

The parallel record management layer uses the services offered by the operating system to provide an abstraction of relations/tables containing records.

2.2.1 Requirements

We first identify the characteristics of data stored in the record management system as well as of the retrieval requests on it. Datasets for many large-scale scientific applications, including those of NASA, exhibit the following characteristics [ROEL91, CAMP90a]:

- The basic data unit is an observation, e.g. from a satellite, with various attributes such as latitude, longitude, temperature, time, etc.
- The data is multi-dimensional, e.g. the three spatial dimensions, the temporal dimension, and various other attributes.
- The database is fairly stationary, i.e. new data can be appended or results of analyses can be added. However, the basic data once added is rarely, if ever, updated.
- High speed and volume of reprocessing requires support for efficient creation and population of relations, both in terms of bandwidth and response time.
- A very high rate of large size retrieval requests is expected from reprocessing algorithms. Large size requests are also expected from the intelligent front-end working on the users' behalf, albeit not at quite the same rate as reprocessing algorithms (though it really depends on user load).

2.2.2 Approach

In the following we describe our approach to the specific problems listed below. Comparisons with related work are included where appropriate.

- Data declustering, i.e. partitioning a file of records across multiple disks of a parallel I/O system.
- Parallel algorithms for range query processing on a single relation/table.
- Parallel algorithms for loading large data files into relations/tables.

Unit datum is modeled as a tuple/record whose attributes/fields represent various facets of the datum such as latitude, longitude, temperature, time, etc. Relations/Files, i.e. a collection of records of the same type, model sets of observations of the same type. A general request on a collection of observations of the same type is modeled as a multi-attribute range query, with predicates defined on one or more attributes.

Let D_i ($1 \leq i \leq d$) be an ordered set. A *record* is an ordered d-tuple $(r_1, r_2, ..., r_d) \in D_1 \times D_2 \times ... \times D_d$. D_i is defined to be the domain of the i^{th} attribute, and r_i is the value of the i^{th} attribute of the record. A d-dimensional file, F, is a non-empty set of records, stored on a parallel disk system with M disks.

The most general retrieval operation, the range query, is denoted by $Q = ([L_1, U_1), ..., [L_d, U_d))$, with $[L_i, U_i)$ being the desired range on the i^{th} attribute. The answer to the range query Q is $A(Q) = \{(r_1, ..., r_d)$ $F \mid L_i \leq r_i < U_i,\ 1 \leq i \leq d\}$. Note that the exact-match query and the partial-match query can be treated as special cases of the range query. For a query Q, let $Work_i(Q)$ be the number of blocks required from disk i to answer the query, $1 \leq i \leq M$, and let $Work(Q) = \sum_{1 \leq i \leq M} Work_i(Q)$ be its total work. Assuming parallel operation of individual disk units, and the performance of the I/O subsystem being the critical factor in system performance - which is a reasonable assumption given trends in parallel machines, the response time of the query is $Rsp(Q) = MAX_{1 \leq i \leq M} \{Work_i(Q)\}$. The optimal (minimal) response time for the query Q by distributing data over M disks is then $\lceil Work(Q)/M \rceil$.

Now, the data declustering problem for a parallel record management system is to develop a strategy such that it provides (i) optimal parallelization of individual queries (***speed-up***) as well as (ii) good parallelization of all possible queries (***robustness***). In the last few years a number of declustering strategies have been proposed [DeWI90, GHAN91, GHAN92, HUA91 LI92, FALO93, ABDE93]. A survey of a some of these is given in [DeWI92]. The focus of [DeWI90, GHAN91] is to decluster based on a single attribute, thus improving performance only of queries containing a predicate on that attribute. [GHAN92] improves upon their previous proposal by selecting a *typical query* and using information about it to improve declustering. [HUA91] considers multiple attributes but optimality is not addressed. [ABDE93, FALO93] identify specific subsets of queries for which their schemes have optimal performance, but the issue of robustness is not addressed. Our work [LI92] has developed the *Co-ordinate Modulo Declustering (CMD)* techniques (i) which is optimal for a very large percentage of all possible single relation SQL queries, (ii) has a small deviation from optimality for the rest, and (iii) whose deviation from optimality decreases as the size of the query result grows. Complete details of CMD and its comparison with other schemes is given in [LI92]. Here we provide a brief overview.

For illustration assume that all files are subsets of the unit space $S = [0, 1)^d$, $d \geq 1$. Divide each dimension of S into nM equal sized intervals for some integer n:

$$[0,\ 1/nM),\ [1/nM,\ 2/nM),\ ...,\ [(nM-1)/nM,\ 1).$$

Let the i^{th} interval of the k^{th} dimension be denoted by $I_{ki} = [l_{ki}, h_{ki}) = [i/nM\ ,\ (i+1)/nM)$, for $0 \leq i \leq nM - 1$, with its *interval coordinate*, ic_k, being i. Given a region $[l_{1i_1}, h_{1i_1}) \times [l_{2i_2}, h_{2i_2}) \times ... \times [l_{di_d}, h_{di_d})$ of S, its *region coordinate*, rc, is defined to as an ordered set of its interval coordinates, i.e. $rc = (ic_1, ic_2, ..., ic_d)$. Now, a region, i.e. partition of the multi-dimensional data space, with region coordinate rc is assigned to disk $CMD(rc, M)$, where the allocation function CMD is defined as:

$$CMD(rc, M) = (ic_1 + ic_2 + ... + ic_d)\ mod\ M.$$

Example 1: Let $S = [0,1)^2$, $M = 4$ and $n = 2$, i.e. each dimension is divided into 8 intervals with length 0.125 each. The partitions of S and their allocation to disk units is shown in Fig. 2.

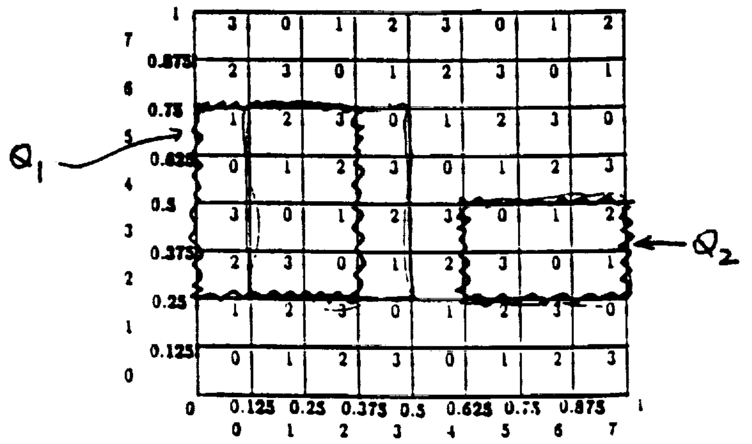


Figure 2. The partition and allocation of S = [0, 1) x [0,1) among 4 disks with M = 4 and n - 2

We have developed parallel algorithms for multi-dimensional range queries on data with CMD partitioning. The following theorems describe the key properties of the algorithms. Proofs are given in [LI92].

Theorem 1 (*Speedup*): The CMD method is optimal for all range queries whose length, in terms of the number of regions covered, on some dimension is equal to kM where k is an integer.

Corollary 2.1. The CMD method is optimal for all range queries in which at least one attribute is unspecified (since the query length on that attribute is the complete range, automatically an integral multiple of M).

Example 2: Consider query $Q_1 = ([0.000, 0.375), [0.250, 0.750))$ in Figure 2. Assuming each region can be fetched in a single disk access, $Work(Q_1) = 12$ disk accesses. Since exactly 3 accesses need to be made to each of the disks 0, 1, 2, 3, the response time for Q_1 is optimal. The condition in Theorem 1 is sufficient but not necessary since optimal response time is also achieved for query $Q_2 = ([0.625, 1.000), [0.250, 0.500))$.

Theorem 2 (*Robustness*): For any arbitrary range query Q the response time, $Rsp(Q)$, is bounded by $\lceil Work(Q)/M \rceil + (M-1)^{d-1} - 1$.

Theorem 2 gives an approximate upper bound, and the actual performance of CMD is much better. For example, for 2 and 3 dimensions the worst case upper bounds are $M/4$ and $M^2/16$, respectively. Note that range queries usually examine a very large subspace of S, i.e. $Work(Q)$ is usually large. Thus $\lceil Work(Q)/M \rceil$, the fraction that is optimal, is much more significant than $(M-1)^{d-1} - 1$.

Parallel Data Loading Algorithms: Our recent work [LI93] is developing efficient parallel algorithms for loading files of records into a CMD format. Initial results show that almost linear speedup of the process, in terms of the number of disk units, is achievable. Detailed algorithms and their properties are discussed in [LI93].

2.3 Parallel Relational Algebra Layer

The *parallel relational algebra layer* contains algorithms for complex operations such as join, union, difference, and aggregation. It uses the abstractions provided by the parallel record management layer.

2.3.1 Requirements

Descriptions of various NASA projects, including the *Intelligent Information Fusion (IIF)* system [ROEL91, CAMP90a], the *Intelligent User Interface for Catalog Browsing* system [CROM89], etc., have identified the need for performing complex comparisons across different types of data sets. Thus, the requirements for this layer are:

- Efficient algorithms for complex operations such as join, union, set difference, etc.
- Efficient algorithms for various kinds of aggregate operations.
- Since space and time are special types of attributes, correlations on them can be potentially treated in a more specialized and efficient manner, e.g. by supporting temporal joins [McKE92].

2.3.2 Approach

In the previous section we presented results about the efficacy of the CMD approach in processing queries accessing a single relation. A vast body of work [DeWI92, WOLF91, FRIE90, SCHN89, DeWI92, CHEN92, NICC92] has shown that join continues to be one of the most expensive relational operations in the parallel environment. Our recent work [NICC92] has shown that an approach to achieving efficiency for complex database operations in a parallel environment is to make them *declustering aware*, i.e. an algorithm implementing a complex operation (e.g. join) will perform better if it is aware of the underlying declustering strategy. [NICC92] describes and analyzes in detail the benefits of making hybrid-hash join algorithm [DeWI84] aware of CMD declustering. We outline the approach here.

For a relation stored using CMD declustering, we define the following:

Definition (*Join Axis, b*): The axis of the multi-dimensional space representing the join attribute 'b'.

Each interval (l_i, h_i) along the join axis denotes a subrange of the join attribute domain.

Definition (*Joining Region, JR(R,B,i)*): The $d-1$ dimensional subspace, of the d dimensional space, created by fixing the subrange of the join axis, b, to have values in the interval (l_i, h_i) and allowing the other axes to be free.

$JR_{R_{a_i}}$ is the i^th joining region of relation R along attribute axis a.

As shown in Figure 3(a), consider R and S as relations to be joined on attribute b. $JR(R, b, 2)$ and $JR(S, b, 1)$ are example joining regions of relations R and S, respectively. A joining region of R must join with every joining region of S with which it overlaps on the join axis. Thus, $JR(R, b, i)$ and $JR(S, b, j)$ must be joined *iff*:

$$(l_i \leq l_j \leq h_i) \text{or} (l_i \leq h_j \leq h_i)$$

The following results describe the properties of our declustering aware approach, details of which are presented in [NICC92]:

Theorem 3: If there is enough aggregate buffer memory, i.e. among all processors together, to hold the largest joining region of the smaller relation, plus one disk block per processor, then no data need be read from the I/O system more than once.

Corollary: There exist cases where declustering aware algorithm will read a disk block exactly once while a non declustering aware algorithm will read it more than once.

In addition to reducing disk accesses, a declustering aware algorithm may entirely eliminate part of the computation, by skipping over entire joining regions of either relation, if there is no intersecting joining region of the other relation, as shown in Figure 3(b). Essentially, a declustering-aware approach to query processing has the following advantages:

- A large problem is broken down into a set of subproblems, such that the sum of the work for the set of subproblems is usually lesser than that for the original. For example, the work for an equi-join between relations R and S, with sizes $|R|$ and $|S|$ respectively, is roughly proportional to $|R||S|$, say with a nested-loops join. If, however, the join axis has k partitions, a declustering-aware nested-loops algorithm is required to do only $k(|R||S|)/k^2$ total work for the k subproblems.
- The performance of most database algorithms, e.g. join, sort, etc., is highly sensitive to the amount of main memory buffer available, with performance often increasing dramatically as the ratio $BufferSize/ProblemSize$ increases [CHOU86, YU93]. For a given amount of aggregate main memory buffer (of the parallel machine), breaking a problem into smaller subproblems has the net effect of increasing this ratio.
- Skewed data distribution causes serious performance problems for most database algorithms [DeWI92a DeWI92b], mainly due to improper load balancing. Declustering aware algorithms provide one way to handle this [NICC92].

2.4 Parallel Query Compilation and Scheduling Layer

Database query compilation for sequential machines provides the functionality of translating a high-level (declarative) query into an optimized sequence of relational algebra and record management level operations. For a parallel machine, the additional decisions of (i) determining the type and degree of parallelization, (ii) an estimation of resource requirements, and (iii) an initial assignment of resources, must be made [GANG92, SRIV93].

2.4.1 Requirements

Descriptions of various NASA projects, including the ***Intelligent Information Fusion (IIF)*** system [ROEL91, CAMP90a], the ***Intelligent User Interface for Catalog Browsing*** system [CROM89], etc., have identified that the interface between the applications, e.g. intelligent front-end of Figure 1, and the database of data products be a high-level one, e.g. SQL. Query compilation and scheduling for parallel databases is currently an active research area [DeWI92a, WILS91, GANG92, SCHN90, HUA93, SRIV93, NICC93]. While detailed survey and comparisons are provided in [SRIV93, NICC93], the basic requirements for this layer are:

- Translation from SQL to an internal form (not a research issue).
- Optimizations performed on the internal form based on the desired objective, e.g. minimize work, minimize response time, etc., to generate a 'good' query execution plan.

- Determining the type(s) and degree of parallelization of the query plan.
- Estimation of resource needs for a query plan to help resource managers during query execution.
- Determining an initial resource allocation for the plan, which may potentially be modified during execution.

2.4.2 Approach

Our overall approach to query compilation is shown in Figure 4. It is a 2-phase approach, where in Phase 1 a compiler that optimizes SQL for sequential machines is used, which (heuristically) minimizes work. *This is not a research issue since good sequential optimizers exist.* The output is fed to Phase 2 which (i) parallelizes the sequential plan, (ii) estimates its resource needs, and (iii) generates an initial resource allocation schedule. The output of Phase 2 is a set of tasks schedulable on a parallel machine. An example input query, represented as a query graph, and its corresponding set of tasks, t_1 through t_{11}, is shown in Figure 5. In each of the seven time slices, numbered 0 through 6, the total resources allocated for this query's execution are shared between the tasks allocated to the slice. Further details are in [NICC93]. While in general it is not true that the parallelization of a 'good' (or even the optimal) sequential plan will yield the best parallel plan, a 2-phase approach such as ours has the advantages of (i) drastically reducing the search space size, and (ii) leveraging off the existing technology in sequential optimization. We share the belief with [STON88, HONG91, HONG92] that a 2-phase approach is a viable heuristic and worth a detailed investigation.

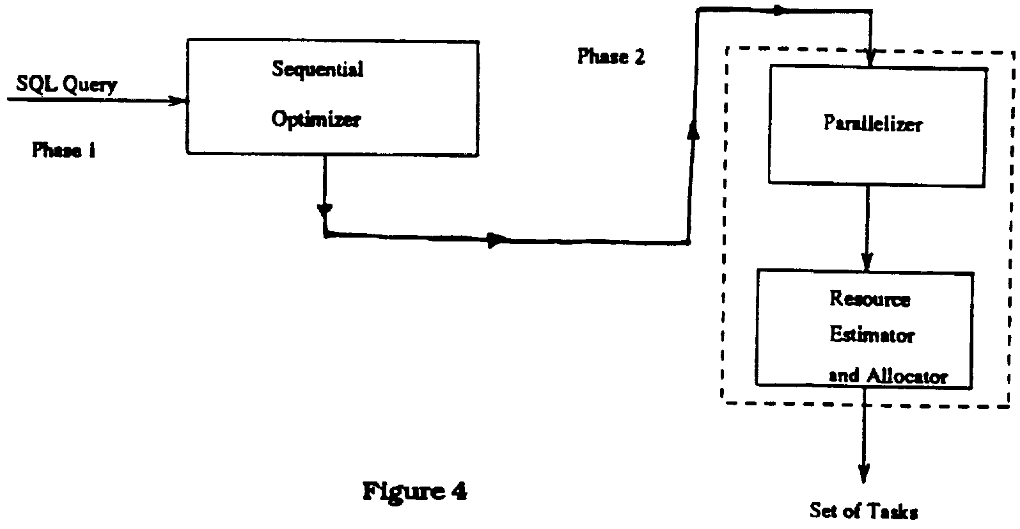


Figure 4

We now briefly describe the key elements of our approach to query compilation and scheduling. Details are provided mainly in [NICC93] and some in [SRIV93]. Specifically, we propose (i) a parallel query plan representation, (ii) a new cost model to incorporate parallel execution, and (iii) heuristic search algorithms.

Query Plan Representation: A parallel query plan can exploit the following kinds of parallelism:

- *Intra-operator parallelism*: A relational operator, such as select, project or join, can be performed by multiple processors simultaneously.
- *Inter-operator parallelism*: Different relational operators of a query, eg. different joins, can be performed in parallel by different (sets of) processors.
- *Pipelining*: Different relational operators can be performed in a pipelined manner using separate (groups of) processors. The result of one is pipelined to the other.

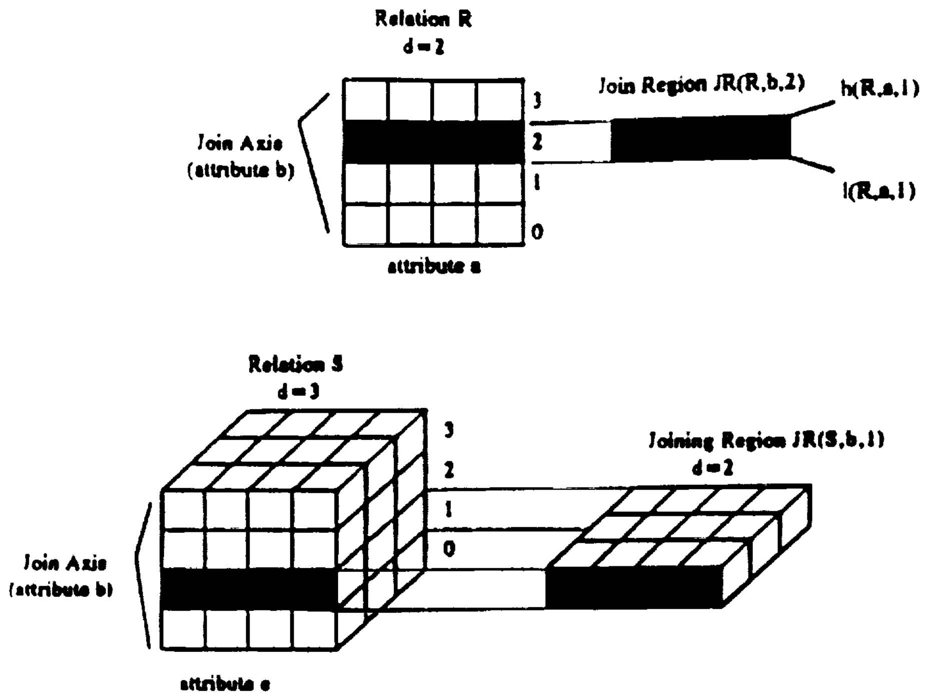


Figure 3 a - 3-Dimensional Relation and a Single Join Region

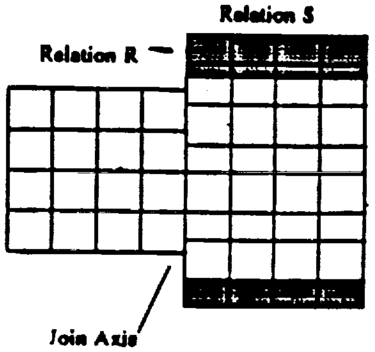
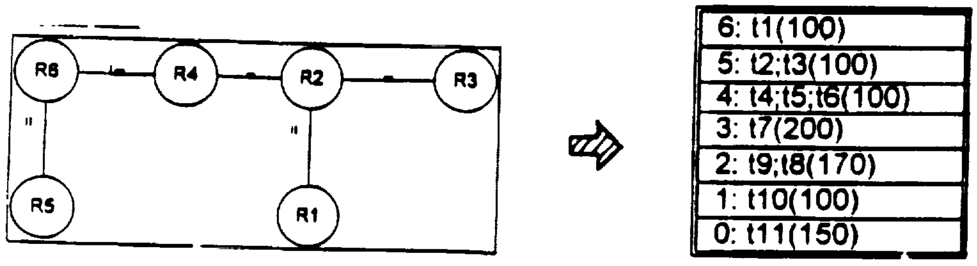


Figure 3 b - A join of situation where some partitions can be ignored



6: t1(100)
5: t2;t3(100)
4: t4;t5;t6(100)
3: t7(200)
2: t9;t8(170)
1: t10(100)
0: t11(150)

Figure 5

In our model a parallel query plan is represented as a *capacitated labeled ordered binary tree*. The shape represents inter-operator parallelism, the orientation represents operand ordering, the node labeling represents intra-operator parallelism, the M (P) branch labeling represents materialization (pipelining) of results between operators, and the branch capacity represents the size of the main-memory (producer-consumer) buffer when materialization (pipelining) of intermediate results is being done.

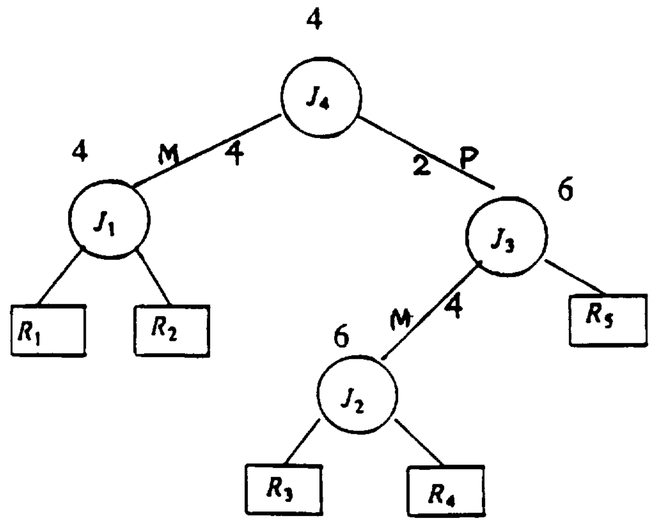


Figure 6

Figure 6 shows a plan for a query with four joins, i.e. J_1, J_2, J_3 and J_4, between five relations, i.e. R_1, R_2, R_3, R_4 and R_5. J_1 has inter-operator parallelism with J_2 (and with J_3). Operations J_1 and J_4 are on a root-leaf path and thus do not have inter-operator parallelism. The same holds between J_2, J_3 and J_4. Since the branch between J_1 and J_4 is labeled with M, J_1 must complete *before* J_4 can begin. The same holds for J_2 and J_3. The branch between J_3 and J_4 is labeled P, and thus the two are pipelined, with J_4 beginning as soon as J_3 has produced the first result tuple. The labels 4, 4, 6 and 6 on J_1, J_4, J_2 and J_3, respectively, represent the number of processors assigned to each. Note that the processors assigned to operators at the opposite ends of a branch labeled M are the same set, i.e. they first perform the child task and then proceed to the parent task. The processors on the opposite ends of a branch labeled P are distinct sets since the operations are pipelined. The 4 processors will first perform the join J_1 and then J_4. The 6 processors will first perform the join J_2 and then J_3. The two processor sets will be working independently while performing the joins J_1 and J_2. While performing J_3 and J_4, the 6 processor set will be the producer while the 4 processor set will be the consumer. The capacity of 2 on the branch (J_4, J_3) means that the intermediate buffer is assigned 2 units of memory. A capacity of 4 on the other branches indicates that each materialized intermediate result has been assigned 4 units of buffer space. Upon overflow the results must go to disk. Total system memory is 10 units.

Cost Model for Parallel Query Plans: A cost model for parallel query plans requires (i) developing analytical cost expressions for individual operators such as select, project, join, etc., and (ii) combining the expressions for individual operators to obtain costs for entire plans. Special care has to be paid in combining costs for operators executing in a parallel or pipelined manner. The two key components are:

- *Cost of Individual Operators*: A number of simulation and experimental evaluations of parallel algorithms for relational operators exist [DeWI90, BARU88, SCHN89, FRIE90]. For query optimization, however, an analytical parameterized cost model is needed. In addition to conventional parameters such as database size, query selectivity, indexes, algorithm used, etc., the cost of an operator depends on (i) its degree of parallelization, (ii) its resource allocation, (iii) parameters of the machine architecture, e.g. costs for unit processing, I/O, and communication operations, and (iv) data declustering.

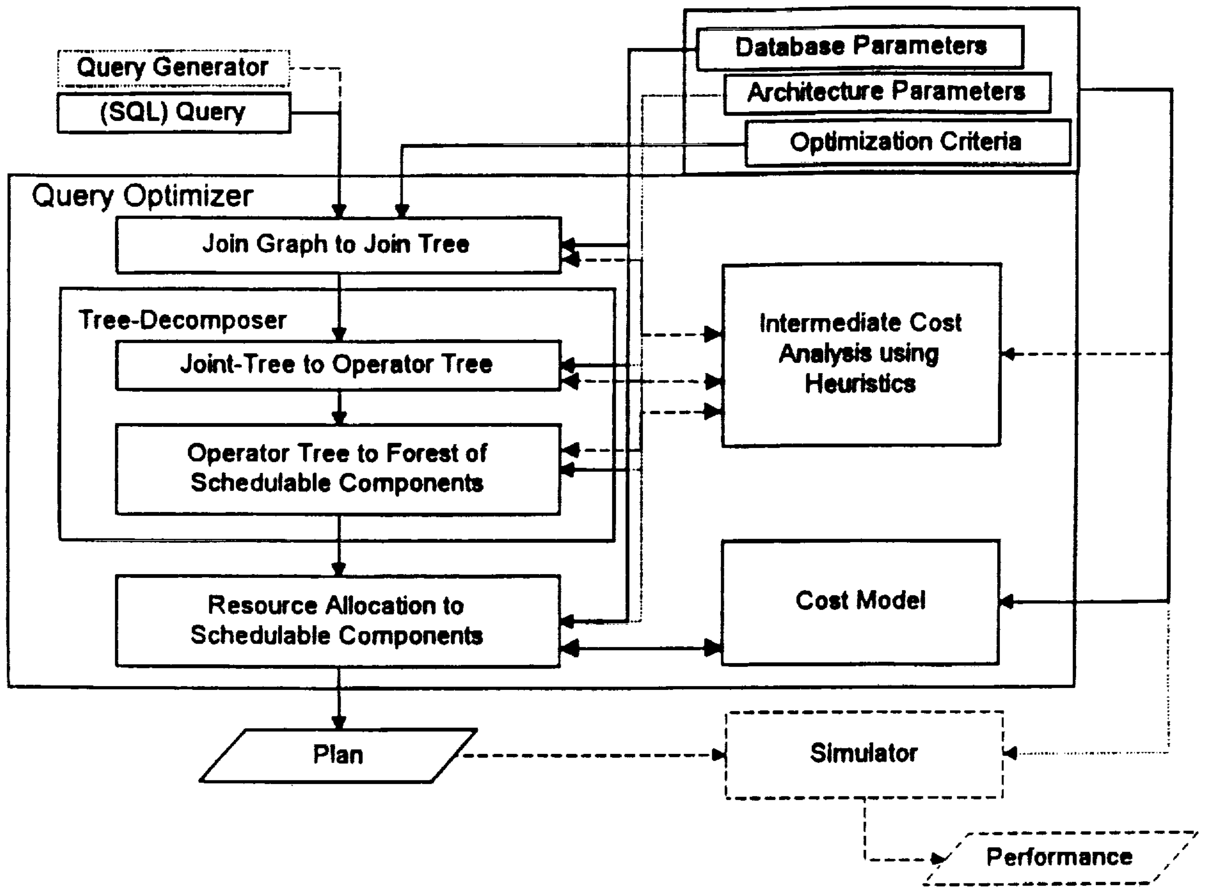


Figure 7. Architecture of an Optimizer

- *Combining Operator Costs*: For parallel query processing the plan with total minimum work and the one with the shortest critical path may not be identical [GUST89]. Maximizing overall throughput in a multiprogrammed environment requires minimizing a query's total work, while minimizing individual response time requires reducing the critical path. Calculating the critical path in a plan can be quite tricky as it needs to consider data flow dependencies and resource allocation [GANG92, SRIV93, NICC93].

In [SRIV93, NICC93] we describe the details of a cost model that addresses the above issues. It provides means of labeling nodes of the query plan tree with various cost metrics such as work, response time, etc., and lends itself to efficient bottom-up evaluation.

Search Algorithm: It has been argued by [SWAM88,SWAM89,IOAN90] that exhaustive enumeration techniques such as dynamic programming [SELI79] are not likely to be successful for queries with large number of joins, i.e. 100 or so, and have proposed heuristic combinatorial optimization techniques such as Simulated Annealing, Iterative Improvement, and Successive Augmentation. The size of the search space for parallel query plans will be much larger than that for sequential ones [SRIV93]. This makes the need for efficient search algorithms of paramount importance. In [SRIV93] and [NICC93] we present two search heuristics to reduce the search space. The key elements of our approach are the following:

- The join-tree output from the sequential optimizer is converted into an operator tree.
- Decisions is made about which branches, i.e. intermediate results, will be pipelines and which will be materialized.
- Resource estimation for various tasks is done.
- Resource allocation for various tasks is carried out.
- At each step some heuristic choices are made to reduce the search space size.

We have built a prototype query optimizer and performed its initial evaluation [SRIV93, NICC93]. Figure 7 shows a schematic of our prototype optimizer. It is a *customizable* optimizer in the sense that it is table-driven and takes architectural parameters from a file as an input to its cost model. Thus, it is customizable to various architectures.

3. Goals & Specific Research Issues

3.1 Research Issues in the Record Management Layer

For the record management layer, the following specific research problems must be addressed:

- Evaluate the CMD approach with NASA data sets.
- Based on above evaluation tune/modify CMD, and if need be create new declustering strategies for NASA's data sets.
- Enhance our approach to provide better declustering by including information about a *core set* of NASA application queries. Many applications often have such a set, and we would like to identify such a set for the reprocessing algorithms.
- Since the relations are partially sorted on each dimension, its benefit on parallel external sorting algorithms needs to be examined.
- CMD provides an implicit indexing because of partial ordering of various domains. How this affects and is complemented by explicit indices, e.g. tree or hash based, needs exploration.
- Development of specialized indices for the parallel I/O system to speed-up the evaluation of aggregates [SRIV89], temporal selections [KOLO89], etc.
- Develop efficient parallel algorithms for loading large data files into relations in the PRDBMS, since this expected to be a frequent operation [PRAT93].
- Develop algorithms to perform operations along the temporal and spatial dimensions efficiently.

3.2 Research Issues in the Parallel Relational Algebra Layer

In this layer the following research issues must be addressed:

- Evaluate our *declustering aware* join algorithm on NASA's data sets.
- Based on above evaluation tune/modify the join algorithm, and if need be create new ones, for NASA's data sets.
- Apply the declustering aware approach to other algorithms in the relational algebra layer, e.g. union, difference, aggregation, etc.

3.3 Research Issues in the Query Compilation Scheduling Layer

Query compilation and scheduling is a wide open area of research today, and a number of issues remain open. Given the fact that it took almost a decade to get satisfactory sequential database query compilers, this is likely to be an area of active research for a few years. Specifically, the following research issues must be addressed:

- Evaluate the effectiveness of our optimizer on some typical queries found in NASA applications.
- Customize our prototype optimizer for a parallel architecture that NASA may be considering for building/acquiring a parallel DBMS on.
- Evaluate and validate the optimizer cost model, which is one of the keys to building a successful optimizer [DeWI92a]

4. Conclusions

In the past decade there has been a tremendous growth in the amount of data and resultant information generated by NASA's operations and research projects. This growth is expected to continue in the future. Use of parallel computers, both processing and input-output, will be a key to solving the resultant data management problem. In this paper we have described the architecture of a parallel data management system which is based on visualizing data as points in space and query processing as geometric operations. The architecture is highly parallel and is quite generic, i.e. can be realized on a wide variety of parallel machines. We provided an overview of our results and pointed out a number of open research issues.

5. References

[BARU88] C. K. Baru, O. Frieder, D. Kanflur, and M. Segal, "Join on a cube: Analysis, Simulation and Implementation", in *Database Machines and Knowledge Base Machines*, M. Kitsuregawa and H. Tanaka, Eds. Boston: Kluwer, 1988, pp 61-74.

[BORA83] H. Boral, and D. DeWitt, "Database Machines: An Idea Whose Time has Passed? A Critique of the Future of Database Machines", in *Proceedings of the 1983 Workshop on Database Machines*, Springer-Verlag, 1983.

[CAMP90a] William J. Campbell, and Robert F. Cromp, "Intelligent Information Fusion for Spatial Data Management", in *Proceedings of the 4th International Symposium on Spatial Data Handling*, 1990, Zurich, Switzerland.

[CAMP90b] William J. Campbell, and Robert F. Cromp, "Evolution of an Intelligent Information Fusion System", in *Photogrammetric Engineering and Remote Sensing*, Vol. 56, No. 6, June 1990, pp. 867-870.

[CHEN 92] M. S. Chen, M. L. Lo, P. S. Yu, and H. C. Young, "Using Segmented Right-Deep Tress for the Execution of Pipelined Hash Joins", in *Proceedings of the 18th VLDB Conference*, August 1992, Vancouver, B.C., Canada.

[CHOU86] Hong-Tai Chou and David J. DeWitt, "An Evaluation of Buffer Management Strategies for Relational Database Systems", in *Proceedings of* 12th *International Conference on Very Large Data Bases*, 1986.

[CROM89] R.F. Cromp, Sharon Crook, "An Intelligent User Interface for Browsing Satellite Data", in *Proceedings of 1989 Goddard Conference on Space Applications of AI*, Greenbelt, MD.

[CROM92] Robert F. Cromp, "An Intelligent Information Fusion System for Handing the Archiving and Querying of Terabyte-sized Spatial Databases", in *Proceedings of International Space Year Conference on Earth and Space Science Information Systems*, 1992, Pasadena, CA.

[DeWI84] D.J. DeWitt, et al, "Implementation Techniques for Main memory Database Systems", in *Proceedings of ACM SIGMOD Conference*, Boston, MA, June 1984.

[DeWI90] D.J. DeWitt, et al, "The Gamma Database Machine Project", in *IEEE Transactions on Knowledge and Data Engineering*, Vol 2, No 1, March 1990.

[DeWI92a] D. J. DeWitt and J. Gray, "Parallel Database Systems: The Future of High Performance Systems," in *Communications of the ACM*, Vol. 35, No. 6, June 1992.

[DeWI92b] D. J. DeWitt, J. F. Naughton, D. A. Schneider, and S. Seshadri, "Practical Skew Handling in Parallel Joins", in *Proceedings of the 18th VLDB Conference*, August 1992, Vancouver, B.C., Canada.

[ABDE93] K.A.S. Abdel-Ghaffar, A. El-Abbadi, "Optimal Disk Allocation for Partial Match Queries", in *ACM Transactions on Database Systems*, Vol 18, No 1, March 1993.

[FALO93] C. Faloutsos, P. Bhagwat, "Declustering Using Fractals", in *Proceedings of 2nd International Conference on PDIS*, San Diego, CA, January 1993.

[FRIE90] O. Frieder, "Multiprocessor Algorithms for Relational Database Operations on Hypercube Systems", in *IEEE Computer*, Vol 19, No 4, December 1990.

[GANG92] S. Ganguly, W. Hasan, R. Krishnamurthy, "Query Optimization for Parallel Execution", in *Proceedings of ACM SIGMOD Conference*, San Diego, CA June 1992.

[GHAN91] S. Ghandeharizadeh, L. Ramos, Z. Asad, and W. Qureshi, "Object placement in Parallel Hypermedia Systems", in *Proceedings of the* 17th *International Conference on Very Large Data Bases*, Barcelona, Spain, 1991.

[GHAN92] S. Ghandeharizadeh, David J. DeWitt, and Waheed Qureshi, "A Performance Analysis of Alternative Multi-Attribute Declustering Strategies", in *Proceedings of ACM SIGMOD Conference*, San Diego, CA June 1992.

[GORD91] David Gordon, etc., "Disk Arrays: Are They of Use for Database Processing?" *Panel in Proceedings of First International Conference on Parallel and Distributed Information Systems*, Dec.1991, Miami Beach, Florida, pp. 117-118.

[GUST89] J.L. Gustafson, "Challenges to Parallel Processing", talk given at University of Minnesota, Minneapolis, September 1989.

[HONG91] Wei Hong and M. Stonebraker, "Optimization of Parallel Query Execution Plans for XPRS", 1st *International Conference on Parallel and Distributed Information Systems*, Miami, Florida, 1991.

[HONG92] Wei Hong, "Exploiting Inter-Operation Parallelism in XPRS", in *Proceedings of ACM SIGMOD Conference*, San Diego, CA June 1992.

[HUA91] K. A. Hua and C. Lee, "Handling Data Skew in Multiprocessor Database Computers Using Partition Tuning", in *Proceedings of the* 17th *International Conference on Very Large Data Bases*, Barcelona, Spain, 1991.

[HUA93] K.A. Hua, Y. Lo, and H.C. Young, "Including the Lad Balancing Issue in the Optimization of Multi-Way Join Querys for Shared-Nothing Database Computers", in *Proceedings of the 2nd International Conference on Parallel and Distributed Information Systems*, January 1993, San Diego, CA.

[IOAN90] Y. E. Ioannidis, and Y. Kang, "Randomized Algorithms for Optimizing Large Join Queries," in *Proceedings of Intl. Conf. on the Mgmt. of Data*, Atlantic City, NJ, May 1990.

[KOLO89] C. Kolovson and M. Stonebraker, "Indexing Techniques for Historical Databases", in *Proceedings of the* 5th *International Conference on Data Engineering*, Los Angeles, LA, 1989.

[LEWI92] Ted G. Lewis, and Hesham El-Rewini, "Introduction to Parallel Computing", *Prentice Hall, Englewood Cliffs, New Jersey*, 1992, ISBN 0-13-498924-4.

[LI92] Jianzhong Li, Jaideep Srivastava, and Doron Rotem, "CMD: A Multidimensional Declustering Method for Parallel Databases Systems", in *Proceedings of the 18th VLDB Conference*, August 1992, Vancouver, B.C., Canada.

[McKE92] Edwin L. McKenzie Jr. and Richard T. Snodgrass, "Evaluation of Relational Algebras incorporating the Time Dimension in Databases", in *ACM Computing Surveys*, Vol. 24 No. 4, December 1991.

[MERC93] A. Merchant and P. S. Yu, "Issues in the Design of Multi-Server File Systems to Cope with Load Skew", in *Proceedings of the 2nd International Conference on Parallel and Distributed Information SYstems*, January 1993, San Diego, CA.

[NICC92] Thomas Niccum, Jaideep Srivastava, and Jianzhong Li, "DA-Joins : Declustering Aware Parallel Join Algorithms," Computer Science Dept. Univ. Of Minnesota, TR92-71.

[NICC93] Thomas Niccum, Jaideep Srivastava, Bhaskar Himatsingka and Jianzhong Li, "A Tree Decomposition Approach to the Parallel Execution of Relational Query Plans", AHPCRC Univ. of Minnesota, TR93-019.

[PATT88] David A. Patterson, Garth Gibson, and Randy H. Katz, "A Case for Redundant Arrays of Inexpensive Disks (RAID)', in *Proceedings of ACM International Conference on Management of Data (SIGMOD)*, June 1988, pp. 109-116.

[PRAT93] Terrence W. Pratt, "CESDIS Proposal Guidelines", February, 1993.

[ROEL91] Larry H. Roelofs, and William J. Campbell, "Applying Semantic Data Modeling Techniques to Large Mass Storage System Designs", Presented at *the Tenth IEEE Symposium on Mass Storage Systems*, 1991.

[SCHN89] D.A. Schneider, D.J.DeWitt, "A Performance Evaluation of Four Parallel Join Algorithms in a Shared-Nothing Multiprocessor", in *Proceedings of ACM SIGMOD Conference*, Portland, OR, June

1989.

[SCHN90] D. A. Schneider and D. J. DeWitt, "Trade-offs in Processing Complex Queries via Hashing in Multiprocessor Database Machines", in *Proceedings of* 16th *VLDB Conference*, Brisbane, Australia, 1990.

[SELI79] P. P. Selinger, et al, "Access Path Selection in a Relational Database Management System", in *Proceedings of ACM SIGMOD International Conf. on Mgmt. of Data*, 1979.

[SRIV89] J. Srivastava, J.S.E. Tan, V.Y. Lum, "TBSAM: A Tree-Based Access Method for Processing Aggregate Queries", in *IEEE Transactions on Knowledge and Data Engineering*, Vol 1, No 4, December 1989.

[SRIV93] Jaideep Srivastava, and Gary Elsesser, "Optimizing Multi-Join Queries for Shared-Memory Multiprocessor", in *Proceedings of the 2nd International Conference on Parallel and Distributed Information SYstems*, January 1993, San Diego, CA.

[STON86] M. R. Stonebraker, "The Case for Shared Nothing," in *Database Engineering*, Vol. 9, No. 1, 1986.

[STON88] M.R. Stonebraker, et al, "The Design of XPRS", in *Proceedings of 14th VLDB Conference*, Los Angeles, CA, August 1988.

[SWAM88] A. Swami, and A. Gupta, "Optimization of Large Join Queries," in *Proceedings of ACM SIGMOD Intl . Conf. on Mgmt. of Data*, Chicago, IL, June 1988.

[SWAM89] A. Swami, "Optimization of Large Join Queries: Combining Heuristics and Combinatorial Techniques," in *Proceedings of Intl. Conf. on Mgmt. of Data*, Portland, OR, June 1989.

[SU86] S.Y.W. Su, "Database Computers: Principles, Architecture and Techniques", *New York: McGraw-Hill*, 1986.

[TDMS90] "Third Generation Database Manifesto," In *ACM SIGMOD Record*, Vol. 19, No. 3, September 1990.

[TMC91] "Data Vault", *talk given by TMC*, 1991.

[WILS91] A.L. Wilschut and P.M.G. Apers, "Dataflow Query Execution in a Parallel Main-Memory Environment", in 1st *International Conference on Parallel and Distributed Information Systems*, 1991, Miami, Florida.

[WOLF91] J.L. Wolf, D.M. Dias, P.S. Yu, and J. Turek, "Comparative Performance of Parallel Join Algorithms",in 1st *International Conference on Parallel and Distributed Information Systems*, 1991, Miami, Florida.

[YU93] P. S. Yu and Douglas W. Cornell, "Buffer Management Based on Return on Consumption in a Multi-Query Environment", in *VLDB Journal*, Vol. 2, No. 1, January, 1993.