5-1990

Dynamic Range Partitioning in Multiprocessor Database Implementations

Ophir Frieder

SU-CIS-90-13

Dynamic Range Partitioning in Mutiprocessor Database Implementations

Ophir Frieder
Bellcore
445 South Street
Morristown, NJ 07960

Kishan G. Mehrotra
Suite 4-116
Center for Science and Technology
Syracuse, NY 13244-4100

May 1990

School of Computer and Information Science
Syracuse University
Suite 4-116
Center for Science and Technology
Syracuse, New York 13244-4100

Dynamic Range Partitioning in Multiprocessor Database Implementations

Ophir Frieder
Bellcore
445 South St.
Morristown, NJ 07960

Kishan G. Mehrotra
School of Comp. Inf. Sci.
Syracuse University
Syracuse, NY 13244

Keywords: Multiprocessing, Join Processing, Data Partitioning

1. Introduction

Multiprocessor implementation of the relational database operators has recently received great attention in the literature [1-4, 8, 11]. As the complexity of implementing the relational operators rests on the inter-node communication patterns involved in an operation, greater research attention has been focussed on *Join* algorithms. The Join traffic patterns subsume those of the remaining relational operators.

To effectively exploit parallelism in *bucket based join implementations*, the domain of the *joining attributes* must be partitioned into equal subranges. That is, the processing of each subrange requires roughly an equal amount of time. A skewed distribution of workload significantly hinders performance. As relations exhibit a non-uniform attribute value distribution, possibly resulting from a previous operation, a priori determination of subrange boundary conditions results in a non-balanced workload across the processors.

Performance degradation in parallel systems employing such static boundary subrange partitioning is demonstrated in Lakshmi and Yu [6]. That study exemplified that even a low degree of attribute skew results in a significant performance penalty. This paper proposes a statistical algorithm for dynamic determination of domain partitioning in bucket based join implementations. This statistics-based approach guarantees a near-uniform processor workload. A parameterization of the *sample size* versus the number of tuples is developed, and a proof of the validity of the approach is discussed. A simple illustrative example is presented.

2. Bucket-Based Join Implementations

Briefly, an attribute is any symbol from a finite set $\mathfrak{L} = \{ A_0, A_1, A_2, ..., A_n \}$. A relation $\Re$ on the set $\mathfrak{L}$ is a subset of the Cartesian product of dom (A_0) x dom (A_1) x dom (A_2) x ... x dom (A_n), where dom (A_i) is the domain of A_i. $R[A_0\ A_1\ A_2\ ...\ A_n]$ represents $\Re$ on the set $\{ A_0, A_1, A_2, ..., A_n \}$. In $R[A_0 A_1\ A_2\ ...\ A_n]$, each column A_i is called an *attribute* of R, and is denoted as $R.A_i$. Each row of R, namely a *tuple*, is designated by $<a_0, a_1, a_2, ..., a_n>$, where $a_i \in$ dom (A_i). Finally, the Join of two relations R[A B] and S[B C], denoted as R[A B] |x| S[B C], is defined by R[A B] |x| S[B C] = { x | x[A B] $\in$ R and x[B C] $\in$ S }, where A, B, and C are a disjoint set of attributes [7].

Bucket based multiprocessor implementations of the join operator, e.g., hash and sort-based algorithms, partition the tuples into subranges (buckets), as follows.

1. Partition the domain of the joining attributes into non-overlapping subranges.
2. Assign each subrange to a unique processor.
3. Route all tuples based on their corresponding joining attribute values to the appropriate processor.
4. Compute the local join.

All tuples residing at a given processor have attribute values within a limited subrange. Thus, as compared to nested loop join implementations [1, 3], few redundant comparisons are performed.

3. Dynamic Bucket Partitioning Algorithm

The Dynamic Bucket Partitioning Algorithm (DBPA) corresponds to first 2 steps of the bucket-based multiprocessor join implementations described in Section 2 where step 1 consists of 4 steps. Let T[A B C] = R[A B] |x| S[B C], with relations R and S horizontally partitioned over M processors, $R_1, R_2, R_3, ..., R_M$ and $S_1, S_2, S_3, ..., S_M$. Any R_i or S_i, $1 \le i \le M$ may be null.

To achieve a perfectly balanced processor workload distribution, an exact global histogram of the data element values is required. As collecting such information is likely to require more time than the savings resulting from an even processor workload, a statistical algorithm that requires limited global knowledge is proposed. The algorithm proceeds as follows:

1. Determine the common range of R.B and S.B.
   Let $\min_{R_i}$, $\max_{R_i}$, $\min_{S_i}$, and $\max_{S_i}$ be the minimum and maximum local R_i and S_i values, $1 \le i \le M$, respectively.
   Let MIN = maximum (minimum ($\min_{R_i}$), minimum ($\min_{S_i}$)), $1 \le i \le M$.
   Let MAX = minimum (maximum ($\max_{R_i}$), maximum ($\max_{S_i}$)), $1 \le i \le M$.
   The common range of R.B and S.B is [MIN, MAX].

2. Remove items not in the common range of R.B and S.B. Determine global count of the remaining R and S tuples.
   Define R'_i and S'_i, $1 \le i \le M$, as the local R_i and S_i values within the common range of R.B and S.B, respectively.
   Let n_{iR} = number of R'_i, $1 \le i \le M$.
   Let n_{iS} = number of S'_i, $1 \le i \le M$.
   The total number of R' and S' respectively is:

$$n_R = \sum_{i=1}^{M} n_{iR} \quad \text{and} \quad n_S = \sum_{i=1}^{M} n_{iS}$$

3. Choose a sample size, N'. Clearly, the larger N' is, the more representative is the randomly selected sample of the global tuple distribution. However, as shown in Section 5, a relatively small N' suffices.
   Determine, the number of randomly selected data points belonging to R'_i, and S'_i, $1 \le i \le M$, to represent N'.
   Let $N_R = \lceil N'\left(\frac{n_R}{n_R + n_S}\right) \rceil$ and $N_S = N' - N_R$.
   Then, at each node i, $1 \le i \le M$, the number of R'_i and S'_i to be drawn is:

$$N_{iR} = \lceil N_R \left(\frac{n_{iR}}{n_R}\right) \rceil \qquad N_{iS} = \lceil N_S \left(\frac{n_{iS}}{n_S}\right) \rceil.$$

$$\text{Let } N = \sum_{i=1}^{M} N_{iR} + \sum_{i=1}^{M} N_{iS}$$

4. Forming the bucket boundaries.
   Draw the **random samples** of sizes N_{iR} and N_{iS} from all processors i, $1 \le i \le M$. A possible random sampling algorithm is described in [9].

   Let Z be the union of all the samples drawn from all the processors. Recalling that N = cardinality(Z), sort the Z values such that $Z[0] \le Z[1] \le \ldots Z[N]$.

   The range of values for processor i, $1 \le i \le M$, are [BOUN[i-1], BOUN[i]], where BOUN[k] is defined as:

$$BOUN[k] = \begin{cases} Z[0], & \text{if } k = 0, \\ Z\left[\frac{kN}{M}\right], & \text{if } 1 \le k \le M-1, \\ Z[N], & \text{if } k = M. \end{cases}$$

4. Analytical Justification of Partitioning Algorithm

To *optimize* the exploitation of parallelism, an equated processor workload is required. Thus, initially a workload function must be defined. In the case of a bucket join, the workload function consists sorting both relation fragments and then linearly comparing the sorted fragments. Assume that each node i, $1 \le i \le M$, contains $|R_i|$ local R_i and $|S_i|$ local S_i tuples after the data are redistributed according to attribute values. Then, the workload at node i, W_i, in comparison time units is:

$$\begin{aligned} W_i &= \text{TIME}(\text{sort}(R_i)) + \text{TIME}(\text{sort}(S_i)) + \text{TIME}(\text{merge}(R_i + S_i)) \\ &= (|R_i| \log_2 |R_i| + |R_i| - 1) + (|S_i| \log_2 |S_i| + |S_i| - 1) + (|R_i| + |S_i| - 1) \\ &= |R_i| \log_2 |R_i| + |S_i| \log_2 |S_i| + 2|R_i| + 2|S_i| - 3. \end{aligned}$$

The proposed solution attempts to minimize $(|R_i| + |S_i|) \log_2 (|R_i| + |S_i|)$ at each node i, $1 \le i \le M$.

To equally partition the total workload across all processors, for each processor i, $1 \le i \le M$,

$$|R_i + S_i| = \frac{|R + S|}{M}.$$

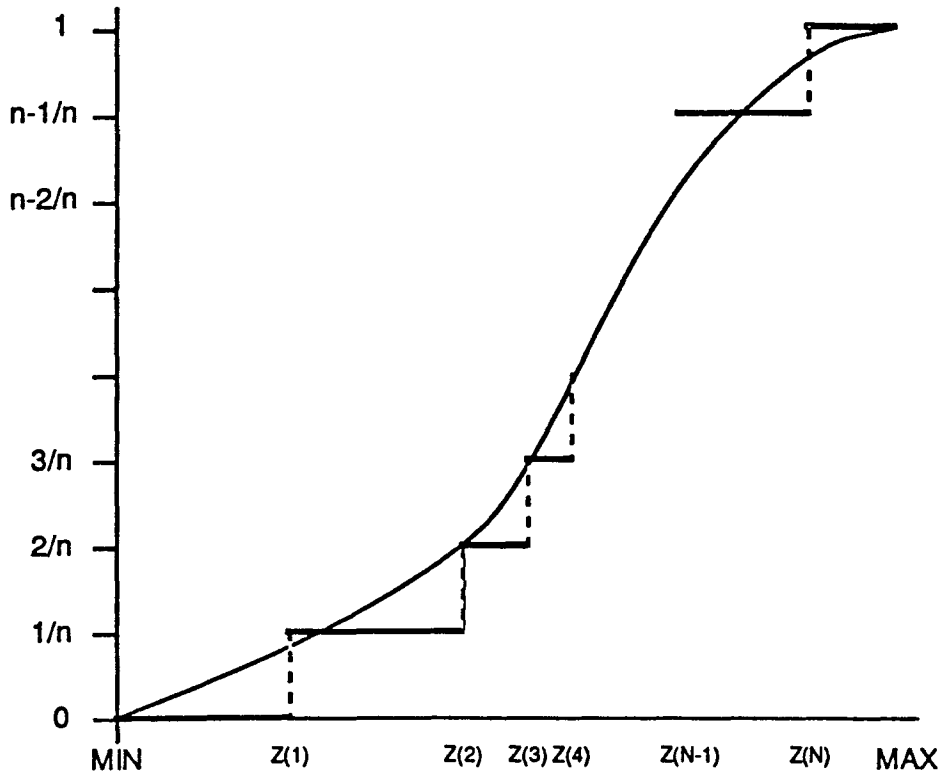

Figure 1. F_N as an Estimator of F.

When formally specified the bucket partitioning problem is:

> Given a large population, R ∪ S, of cardinality N*, with an unknown distribution function, F, partition the population into M equal, non-overlapping regions. That is, determine the boundary conditions BOUN[0], BOUN[1], ... BOUN[M], where the range of values for processor i, $1 \le i \le M$, are [BOUN[i-1], BOUN[i]].

To determine the boundary values, a sample of size N is drawn from the total population, and a sample distribution function , F_N, an estimator of F, is constructed. The relation between F_N and F, shown in figure 1, is a graphical argument justifying the approach. For notational convenience, denote BOUN[k] of the previous section by $\hat{\xi}_k$. Then,

$$F_N(\hat{\xi}_i) = \frac{i}{M}, \qquad 0 \le i \le M,$$

where $\hat{\xi}_0$ = MIN and $\hat{\xi}_M$ = MAX, i.e., precisely $\frac{1}{M}$ of the sample values fall between $(\hat{\xi}_{i-1}, \hat{\xi}_i]$, for $1 \le i \le M$. As F_N is a closed approximation to F, it is expected that $\frac{1}{M}$ of the population values will fall between $(\hat{\xi}_{i-1}, \hat{\xi}_i]$, for $1 \le i \le M$. This last statement is made precise in the following discussion.

Let Q_i = the proportion of population Z values in the interval $(\hat{\xi}_{i-1}, \hat{\xi}_i]$, and
let P_i = $\frac{1}{M}$, i.e., the desired proportion of population Z values in interval $(\hat{\xi}_{i-1}, \hat{\xi}_i]$, for $1 \le i \le M$.

Finally, denote by ξ_i, the unknown i^{th} boundary point for the population Z values, i.e.,

$$F(\xi_i) = \frac{i}{M}, \qquad 1 \le i \le M,$$

where ξ_0 = MIN and ξ_M = MAX.

The proposed algorithm is unlikely to result in an optimal partitioning of the workload. Hence a characterization of the resulting partitioning from the optimal, namely, $Q_i - P_i$, is necessary. First, we note that

$$\begin{aligned} Q_i - P_i &= Pr[\, \hat{\xi}_{i-1} < Z \le \hat{\xi}_i \,] - \frac{1}{M} \\ &= (Pr[\, Z \le \hat{\xi}_i \,] - Pr[\, Z \le \hat{\xi}_{i-1} \,]) - (\frac{i}{M} - \frac{i-1}{M}) \\ &= (Pr[\, Z \le \hat{\xi}_i \,] - \frac{i}{M}) - (Pr[\, Z \le \hat{\xi}_{i-1} \,] - \frac{i-1}{M}) \\ &= U_i - U_{i-1} \end{aligned}$$

where

$$\begin{aligned} U_i &= Pr[\, Z \le \hat{\xi}_i \,] - \frac{i}{M} \\ &= F(\hat{\xi}_i) - F(\hat{\xi}_{i-1}), \quad \text{for } 1 \le i \le M. \end{aligned}$$

Second, we consider the distribution of U_i.

Theorem:

For large values of N, the distribution function of the random variable $\sqrt{N}\, U_i$ is approximated by a NORMAL distribution with a mean = 0 and a variance = $\frac{i}{M}(1 - \frac{i}{M})$.

Proof:

[The proof of this theorem is well known and is presented here for completeness only]

From Rao [10, pg 154], we observe that

$$\hat{\xi}_i = \xi_i + \frac{F_N(\xi_i) - F(\xi_i)}{f(\xi_i)} + R_N,$$

where

$f(x) = \frac{\partial}{\partial x} F(x)$ and the remainder term, R_N, is on the order of:

$$R_N = O\left(\left(\frac{\log \log N}{N} \right)^{0.75} \right).$$

Hence,

$$U_i = F\left(\xi_i + \frac{F_N(\xi_i) - F(\xi_i)}{f(\xi_i)} + R_N \right) - F(\xi_i).$$

By Taylor expansion of the first term on the right side of the above equation,

$$= F(\xi_i) + \left(\frac{F_N(\xi_i) - F(\xi_i)}{f(\xi_i)} + R_N \right) f(\xi_i) + R_N^* - F(\xi_i)$$

$$= F_N(\xi_i) - F(\xi_i) + f(\xi_i) R_N + R_N^*.$$

Here R_N^* is the remainder term in the Taylor expansion and is on the order of:

$$R_N^* = O\left((F_N(\xi_i) - F(\xi_i))^2 \right).$$

But, note that $NF_N(\xi_i)$ is a Binomial random variable with mean $NF_N(\xi_i)$ and variance $NF_N(\xi_i)(1 - F(\xi_i))$. Then, by the Central Limit Theorem, Rao [10, pg 21], the distribution of $\sqrt{N}\left(F_N(\xi_i) - F(\xi_i)\right)$ is approximated by a NORMAL distribution with mean 0 and variance $F_N(\xi_i)(1 - F(\xi_i)) = \frac{i}{M}\left(1 - \frac{i}{M}\right)$.

In addition, as N -> N*, $\sqrt{N}\,R_N$ and $\sqrt{N}\,R^*_N$ converge to very small numbers and do not make any contribution to the main term $\sqrt{N}\,(F_N(\xi_i) - F(\xi_i))$. Therefore,

$$U_i\sqrt{N} \approx \sqrt{N}\,(F_N(\xi_i) - F(\xi_i))$$

and its distribution are approximated by a NORMAL distribution with mean 0 and variance $\frac{i}{M}(1 - \frac{i}{M})$. Q. E. D.

Similarly, it can be shown that the distribution of $\sqrt{N}\,(Q_i - P_i) = \sqrt{N}\,(U_i - U_{i-1})$ is approximated by a NORMAL distribution with mean 0 and variance $\frac{1}{M}(1 - \frac{1}{M})$.

Consequently, we can make the following statement regarding the error in approximating P_i by Q_i.

$$\Pr\left[\frac{-2}{\sqrt{N}}\sqrt{\frac{M-1}{M^2}} \le Q_i - P_i \le \frac{2}{\sqrt{N}}\sqrt{\frac{M-1}{M^2}}\right] \approx .95$$

In general, this states that, by varying N, it is possible to get arbitrarily close to P_i, i.e., a perfectly balanced workload distribution. In the limiting case of N = N*, it is obvious that $P_i \equiv Q_i$. As shown in Section 5, even a vary small N achieves nearly an optimal distribution.

5. Experimental Analysis

To illustrate the approach, a simulation of a 4 node system was developed. The R values were randomly generated following UNIFORM distributions of varying range. The S values were also randomly generated but using NORMAL distributions with varying mean and variance. All values are randomly partitioned across the 4 nodes.

Three sample runs are presented. In all three runs, a sample size N, of 10000 data elements was drawn at random, as described in Section 3. The population size consisted of

N* = 1000000		Z = 912344	N=10000	e_i = 228086
Proc. No.	R	S	Total	Absolute Error
0	130684	98823	229507	0.0062
1	64802	161805	226607	0.0065
2	73671	151284	224955	0.0137
3	143233	88042	231275	0.0140

N* = 5000000		Z = 4597114	N=10000	e_i = 1149278
Proc. No.	R	S	Total	Absolute Error
0	653430	492285	1145715	0.0031
1	320007	797976	1117983	0.0272
2	384751	783975	1168726	0.0169
3	739206	425484	1164690	0.0134

N* = 10000000		Z = 9172246	N=10000	e_i = 2293061
Proc. No.	R	S	Total	Absolute Error
0	1285454	947809	2233263	0.0261
1	655757	1619610	2275367	0.0077
2	748183	1543630	2291813	0.0005
3	1483378	888425	2371803	0.0343

Figure 2. Experimental Results

one million, five million, and ten million data points, respectively. The results are provided in figure 2.

Figure 2 illustrates the results of the three sample runs. N* is the size of the total population. Z is the number of data values in common range of relations S and R. The optimal partitioning of Z, if it was possible is represented by e_i. As the optimal partitioning may require that some of the data points whose value is v_t be routed to processor i and the rest to processor i+1, an optimal partitioning may not always be possible. Finally, the actual resulting partitioning, o_i, and the absolute error are presented. The absolute error is:

$$\text{error} = \left| \frac{e_i - o_i}{e_i} \right|.$$

in Figure 2, we note that the maximal error is 0.0140, 0.0272, and 0.0343, for the one million, five million, and ten million data points, respectively. Even with examining only 0.1 % of the total data points, a near optimal partitioning is obtained.

6. Conclusion and Future Work

To nullify the effects of data skew in multiprocessor bucket based join algorithms, a statistical algorithm that dynamically determines the bucket ranges was developed. The theoretical underpinnings of the algorithm were provided. Using a simulation of a 4 node multiprocessor, an experimental evaluation of the algorithm was performed. The results demonstrate that with examining only 0.1% of the total data points, a near optimal partitioning of the total join workload across the processors is obtained. Thus, the proposed algorithm nullifies the performance degradation resulting from data skew without requiring that the total data values be examined.

Utilization of dynamic bucket partitioning introduces additional overhead in join processing. The incurred overhead results both from additional local computation and inter-processor communication. From an initial study [5], if only a small number (roughly 3000 or less) of data items are processed at each node, the communication portion of the overhead dominates the computational portion (sorting a large set of items is computationally time-consuming). As the number of items sampled per node is small, the incurred overhead is dependent on the communication network employed in terms of absolute total time (seconds), and likely to depend on the communication network in terms of the relative percentage of the total execution time.

Thus, in brief, we remark that by off setting data skew we expect to improve the performance of the join algorithm at the cost of some additional computation and data communication. A detailed experimental study of the net reduction in execution time is currently under way.

To evaluate the overhead, parallel join algorithms for various architectures were and are being developed. Some of the distributed-memory architectures under investigation include ring, broadcast bus, mesh, and hypercube-based multiprocessors. An investigation of the observed performance benefits of this approach on Local Area Networks is also planned. Current experimentation consists of varying the skewness of the data while keeping the

number of processors and interconnection scheme fixed and studying the issue of scalability. That is, the degree of effective parallelism in terms of the number of nodes that can be exploited with a satisfactory improvement in performance.

Acknowledgement

We wish to thank Dr. Gideon Frieder for his technical insights in the research effort. We also wish to thank Nick Karonis and Paul Jackson for their contributions.

References

1. Baru, C. K. and O. Frieder, "Database Operations in a Cube-Connected Multicomputer System," *IEEE Transactions on Computers*, 38(6), 1989.

2. De Witt, D. and R. Gerber, "Multiprocessor Hash-Based Join Algorithms," *Proceedings of 1985 Very Large Database Conference*, 1985.

3. Hillyer, B. and D. E. Shaw, "NON-VON's Performance on Certain Database Benchmarks," *IEEE Transactions on Software Engineering*, 12(4), 1986.

4. Kitsuregawa, M., Tanaka, H., and Moto-Oka, T.,"Architecture and Performance of Relational Algebra Machine GRACE," *Int'l Conf. on Parallel Processing Proceedings*, 1984.

5. Karonis, N., *personal communication*, 1989.

6. Lakshmi, M. S. and P. S. Yu, "Effect of Skew on Join Performance in Parallel Architectures," *IEEE Int'l Symp. on Databases in Parallel and Distributed Systems*, 1988.

7. Maier, D., *The Theory of Relational Databases*, Computer Science Press, Rockville, Maryland, 1983.

8. Omiecinski, E. and E. Tien, "A Hash-Based Join Algorithm for a Cube-Connected Parallel Computer," *Information Processing Letters*, 30(5), 1989.

9. Rajan, V., R. K. Ghosh, and P. Gupta, "An Efficient Parallel Algorithm for Random Sampling," *Information Processing Letters*, 30(5), 1989.

10. Rao, P., *Asymptotic Theory of Statistical Inference*, Wiley, 1987.

11. Valduriez, P. and G. Gardarin, "Join and Semijoin Algorithms for a Multiprocessor Database Machine," *ACM Transactions on Database Systems*, 9(1), 1984.