3G−2

An Efficient Query Execution Plan for Multi-Way Joins in Shared-Nothing Database Environment

Lilian Harada and Naoki Akaboshi
Fujitsu Laboratories

1. Introduction

In [1], we have introduced an analytical model for the parallel processing of multi-way joins in shared-nothing database environment, with analysis of the hash-join algorithm applied to the linear trees represented in the left and right-deep tree structures. We have found that the length of the pipeline segment that depends on the structure of the query execution tree, and the utilization of the system resources (amount of main memory, disk and interconnection network bandwidth) are important factors in the efficient parallel processing of multi-way joins.

In this paper we develop a scheme to determine the query execution plan based on the system resources balance so that the query can be efficiently executed in a shared-nothing database environment. As shown by our simulation, the proposed approach leads to query plans of significantly better performance than those achievable by the previous schemes using left-deep and right-deep trees.

2. Tradeoffs between Left-Deep and Right-Deep Query Trees

In [1] we have introduced a performance modeling of a multi-way join of N relations on different N-1 join attributes, i.e., $R_1 \bowtie_{a_1} R_2 \bowtie_{a_2} \ldots \bowtie_{a_{N-1}} R_N$, denoted by left-deep and right-deep structures of query execution trees, as shown in Fig. 1(a) and (b). Using hash-join algorithms, multiple joins can be pipelined so that the early resulting tuples from a join operator, before the whole processing of the join operator is completed, can be sent to the next join operator for processing. In a left-deep tree, the result of a join operator is used to build the hash table for the next join operator, and several join operators thus need to be executed sequentially. In contrast, in a right-deep tree all the hash tables are built from the original input relations, and the result of a join operator is input into the next join operator as a probing tuple. The tuples of the right-descendant relation can thus go through the whole right-deep tree in a pipelined manner. When the amount of memory is not enough to accommodate all the N-1 hash tables, the right-deep tree can be decomposed into disjoint segments such that the hash tables of the active operators in each segment fit into memory. The segments are executed one by one bottom up. The last join operator in each segment have to spool its tuple production to a temporary relation in the disk, which will be the probing relation of the next segment.

Given the constraint of limited space, here we will only concern ourselves with an overview of the results of our performance modeling, and focus on the comparison of the left- and right-deep trees. Complete details on the performance modeling and evaluation are available in [1].

When the network bandwidth is much larger than the disk bandwidth, all the segments of both left- and right-deep trees are I/O bound and thus, the performance is proportional to the data transferred to/from the disks. We could see that the left-deep tree, which determines short pipeline segments and utilizes the available memory to build the hash table for the next pipeline segment, showed better performance than the right-deep tree, which fully utilizes the available memory to enlarge the segment pipeline length, at the cost of spooling the intermediate data into the disk.

By decreasing the network bandwidth to the disk bandwidth, the performance for both trees decreases. However, we could see that the degradation was greater for the left-deep tree. From the evaluation results, we could see that the right-deep tree could be processed with longer lengths of pipeline segments, such that the maximum overlap among the disk transfer and the network transfer could be achieved, leading to the best performance.

3. Execution Plan Balancing

Summarizing the results of our previous evaluation of the left-deep and right-deep trees, we conclude that, in order to achieve good performance, for each pipeline segment processing, it is necessary to have (a) good balance between network and I/O transfer, as much as possible; however, for I/O bound systems, this balance can not be achieved and so, the best performance is achieved when (b) there is no I/O of intermediate data.

As an effort to improve the execution of pipelined hash joins, one would naturally like to develop efficient schemes to generate effective query plans that fully exploit these results. Consequently, we propose in this paper the approach based on the balance of the system resources for the execution of pipelined hash-joins. We generate a linear tree which is composed of a set of right-deep subtrees. But it differs from a right-deep tree in that the determination of the pipeline segment is done by considering the system resources balance, and that the resulting relation of a pipeline segment can be either written back to disk or maintained in memory and immediately used as a building relation for the next pipeline segment.

Because of lack of space, we only illustrate the idea of our strategy, with no details of the costs used for the determination of the pipeline segments, which are based on the performance modeling introduced in [1].

The idea is to divide the query execution plan into segments. By considering the available memory size, the disk and network bandwidths, each segment is determined as:

(1) The shortest pipeline segment from all the possible segments like in a right-deep tree, such that disk I/O and network transfers are well balanced.

(2) If a segment can not be determined satisfying (1), choose the segment with the best balance of disk I/O and

network transfers, by considering the resulting relation of the pipeline segment is immediately used as a building relation for the next segment, as in a left-deep tree.

The pipeline segments are determined by (1) or (2) in a top-down manner.

Fig. 1 shows an example of the query execution plans based on the left- , right-deep tree and our approach, for a 10-way join, whose input and intermediate relations cardinalities are shown in Table 1, when the available memory is 300 Ktuples. As can be observed from Fig. 1, the three approaches differ in the determination of the pipeline segments. In the left-deep tree, each segment contain only one hash table. In the right-deep tree, each segment length is maximized, by filling the available memory with the maximum number of hash tables. On the other hand, in our proposed tree, each segment length is determined so that the corresponding cost of the network and disk transfers are as much overlapped as possible.

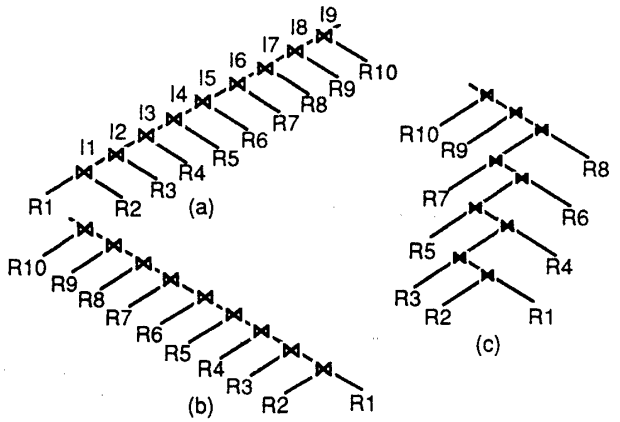

Fig. 1 (a) Left-Deep Tree, (b) Right-Deep Tree and (c) Our Proposed Tree

	R1	R2	R3	R4	R5	R6	R7	R8	R9	R10
K Tuple	91.7	96.2	97.5	102.9	97.8	109.4	105.6	94.1	99.3	106.9

	I1	I2	I3	I4	I5	I6	I7	I8	I9
K Tuple	91.9	94.9	107.6	101.7	104.8	92.4	95.3	108.7	103.9

Table 1 Input and Intermediate Relations Cardinalities

4. Evaluation Results

Extensive simulations were performed to evaluate the proposed query execution plan. In the following, we present only some representative results. We consider a 10-way join whose relations and results sizes are shown in Table 1, when the tuples length is 208 B. The disk sequential and random bandwidth are maintained at 0.54 and 0.32 MB/s, and the network bandwidth are 0.54, 0.70 and 1.72 MB/s for Fig. 2, 3 and 4, respectively, which show the execution times of the left-deep, right-deep and our approach when varying the total memory size of a system composed of 16 processors. As can be observed from all three curves, our approach shows the best performance by flexibly determining the pipeline segments that best fits for the given memory size, disk and network bandwidths.

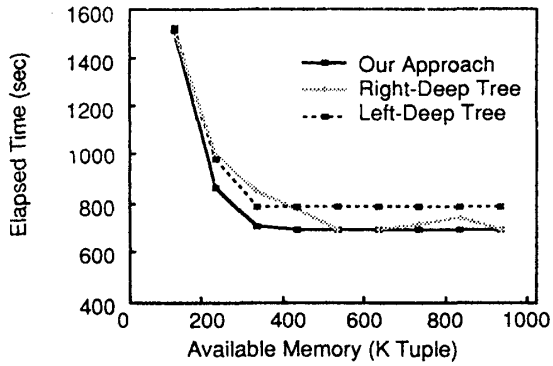

Fig. 2 Results for net. bandwidth = 0.54 MB/s

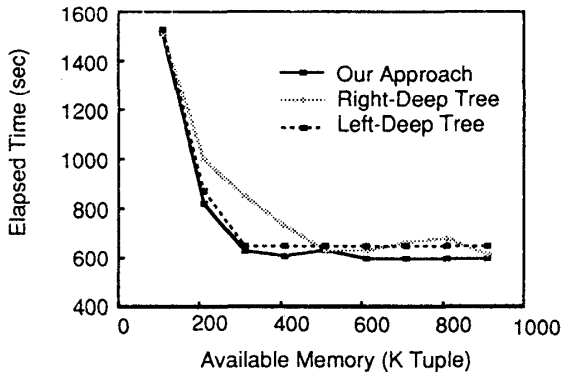

Fig. 3 Results for net. bandwidth = 0.70 MB/s

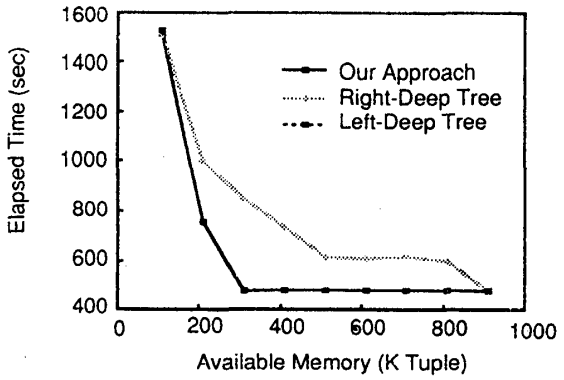

Fig. 4 Results for net. bandwidth = 1.72 MB/s

5. Conclusion

In this paper we have presented a query execution plan which considers the load balance for determining the pipeline segments for a multi-way join in a shared-nothing database environment. Preliminary experiments indicate that the consideration of the system resource balance are effective in producing parallel execution plans, with better performance than those achieved by the traditional schemes using left-deep and right-deep trees.

References

[1] L.Harada and N.Akaboshi, "Evaluation Results of Multi-Way Joins in Shared-Nothing Database Environment", 情報処理学会第 45 回全国大会, 5R-2, 1992

Acknowledgment

We wish to thank Mrs. Miyuki Nakano from Univ. of Tokyo for many helpful suggestions on this subject.