A Note on the Strategy Space of Multiway Join Query Optimization Problem in Parallel Systems

Kian-Lee Tan *Hongjun Lu*
Department of Information Systems and Computer Science
National University of Singapore
Singapore 0511

ABSTRACT

In this short note, we estimate the search space of optimizing multiway join queries in multiprocessor computer systems, i.e. the number of possible query execution plans that need to be considered.

1. Introduction

Consider the problem of evaluating the expression $R_1 \bowtie R_2 \bowtie \cdots \bowtie R_n$ where there are n relations $R_1, R_2, \ldots, R_n$ and $n-1$ joins. While it is widely accepted that the number of possible execution plans is large, there is, to the best of the authors' knowledge, no work that indicates how large the search space is. An upper bound on the number of processing trees (PT) that corresponds to the expression is given in [Swam88] as $\binom{2n}{n} \cdot n!$. However, this number merely considers the join orders without join methods. In a multiprocessor environment, the allocation of resources such as processors and memory increases the search space drastically. We first provide an exact number of PTs that corresponds to a join expression, followed by deriving a lower and an upper bounds on the number of evaluation plans for such an expression in a parallel environment.

2. Number of Processing Trees

The processing tree identifies the order of evaluating the joins in a multiway join query. Each PT can be represented as a triple (T_k, r, T_{n-k}) where r is a distinguished node called the *join result* of T_n, T_k is a PT of k relations for some $k = 1, \cdots, n-1$, called the left subtree of T_n, and T_{n-k} is a PT of $n-k$ relations, called the right subtree of T_n. From this definition, we can deduce that x_n, the number of possible PTs T_n of n relations, is given by the recurrence relation:

$$x_n = \sum_{k=1}^{n-1} \binom{n}{k} \cdot x_k \cdot x_{n-k} \quad \text{where } x_1 = 1$$

Since for every triple (T_k, r, T_{n-k}), there is a corresponding triple (T_{n-k}, r, T_k) and the two are indistinguishable, we have y_n, the ***number of distinct processing trees*** as:

$$y_n = \frac{1}{2} \cdot \sum_{k=1}^{n-1} \binom{n}{k} \cdot y_k \cdot y_{n-k} \quad \text{where } y_1 = 1$$

$$= \binom{2(n-1)}{n-1} \cdot \frac{(n-1)!}{2^{n-1}} \qquad (1)$$

Proof of Equation (1):

Let $T_k = \binom{2(k-1)}{k-1} \cdot \frac{(k-1)!}{2^{k-1}} = \frac{1}{2^{k-1}} \cdot \frac{(2(k-1))!}{(k-1)!}$

Now $\frac{1}{2} \sum_{k=1}^{n-1} \binom{n}{k} T_k T_{n-k}$

$$= \frac{1}{2} \sum_{k=1}^{n-1} \frac{n!}{(n-k)!k!} \frac{(2k-2)!}{(k-1)!2^{k-1}} \frac{(2(n-k)-2)!}{(n-k-1)!2^{n-k-1}}$$

$$= \frac{n!}{2^{n-1}} \sum_{k=1}^{n-1} \frac{(2k-2)!}{k!(k-1)!} \frac{(2(n-k)-2)!}{(n-k)!(n-k-1)!}$$

$$= \frac{n!}{2^{n-1}} \sum_{k=1}^{n-1} \frac{(2(k-1))!}{(k-1)!^2} \cdot \frac{1}{k} \cdot \frac{(2((n-1)-k))!}{(n-k-1)!^2(n-k)}$$

$$= \frac{n!}{2^{n-1}} \sum_{k=0}^{n-2} \frac{2k!}{k!^2(k+1)} \cdot \frac{(2((n-2)-k))!}{(n-2-k)!^2(n-2-k+1)}$$

Let $a_k = \frac{(2k)!}{k!^2(k+1)}$. Since $\sum_{k=0}^{n} a_k a_{n-k} = a_{n+1}$ (Property of Catalan's number), we have

$$\frac{1}{2} \sum_{k=1}^{n-1} \binom{n}{k} T_k T_{n-k}$$

$$= \frac{n!}{2^{n-1}} \frac{2(n-1))!}{(n-1)!^2 n} = \frac{(2(n-1))!}{2^{n-1}(n-1)!} = T_n \quad \square$$

According to Eq. (1), for an example, the number of distinct PTs for 2, 3, 4, 5, 6 and 7 relations are 1, 3, 15, 105, 945, 10385 respectively.

In the uniprocessor environment, besides choosing the join order, a query execution plan needs to specify the join method for each of the join as well. ***Given m join methods supported by the DBMS, the total number of query evaluation plans becomes***

$$z_n = m^{n-1} \cdot y_n \qquad (2)$$

3. Bounds on the Number of Plans

The search space increases drastically in a multiprocessor environment. This is so since different resource allocation will generate QEPs with different performance. The critical resources to be considered in query optimization include the number of processors and the available buffer space. To simplify the problem, we consider here only the allocation of processors. Thus, a QEP is a PT with information of join methods and number of processors allocated to each join. We also assume that all processors are assigned to a set of joins at the same time. Next assignment will not start until all processors complete their current assignment.

Consider a PT of the y_n of them. Given p processors, there are several ways of allocating the processors to the joins. All processors may be assigned to process each join serially. p_1 processors may be assigned to process a join while at the same time p_2 processors may be assigned to process another join where $p = p_1 + p_2$. This problem of allocating processors to joins is similar to the problem of *putting indistinguishable balls into distinguishable cells such that each cell cannot be empty* [Robe84]. In our case, the processors are the balls and the joins are the cells. The number of ways to distribute p indistinguishable processors into j joins is $\binom{p-1}{j-1}$. ***The number of plans (without considering join methods) that corresponds to one particular processing tree is***

$$Total_n = \sum_{j=1}^{z} k_z \binom{p-1}{j-1} \cdot Total_{n-j} \qquad (3)$$

where $Total_1 = 1$. In the above equation, $z = \begin{cases} w & w \le p \\ p & otherwise \end{cases}$ and k_z is the number of *feasible* joins to be selected from the w joins. w represents the exact number of joins that the processors can be allocated for concurrent processing. Note that w is a variable and may have different values for different PTs and for a PT, at each iteration of the recursion w takes on different values. When $z = w$, $k_z = \binom{w}{j}$ but when $z = p$, $\binom{p}{j} \le k_z \le \binom{w}{j}$.

An upper bound to Eq. (3) is then to let $w = n-1$. This, however, will imply that some of the plans are meaningless. For example, let $n = 10$ and $p = 3$. For a linear PT $(R_1 \bowtie R_2) \bowtie R_3) \cdots \bowtie R_{10})$, it is meaningless to consider the plan that assign all the processors to the last join first. Therefore

$$UPPER\_Total_n = \sum_{j=1}^{z} \binom{n-1}{j} \binom{p-1}{j-1} \cdot UPPER\_Total_{n-j} \qquad (4)$$

where $UPPER\_Total_1 = 1$ and $z = \begin{cases} n-1 & n-1 \le p \\ p & \text{otherwise} \end{cases}$.

Therefore, for all the y_n PTs, the maximum number of plans is:

$$UPPER\_QEP_n = \sum_{i=1}^{y_n} UPPER\_Total_n \qquad (5)$$

$$= y_n \cdot UPPER\_Total_n$$

and the maximum number of QEPs, inclusive of join method is:

$$m^{n-1} \cdot UPPER\_QEP_n \qquad (6)$$

A lower bound corresponding to the Eq. (3) is then to avoid choosing the joins to be performed concurrently, that is there is only a specific set of joins that can be done concurrently. Therefore,

$$LOWER\_Total_n = \sum_{j=1}^{z} \binom{p-1}{j-1} \cdot LOWER\_Total_{n-j} \qquad (7)$$

where $LOWER\_Total_1 = 1$ and $z = \begin{cases} n-1 & n-1 \le p \\ p & otherwise \end{cases}$

Therefore, for all the y_n PTs, the least number of plans is:

$$LOWER\_QEP_n = \sum_{i=1}^{y_n} LOWER\_Total_n \qquad (8)$$

$$= y_n \cdot LOWER\_Total_n$$

and the minimum number of QEPs, inclusive of join method is:

$$m^{n-1} \cdot LOWER\_QEP_n \qquad (9)$$

One could the further expand the search space by considering buffer space allocation. With such a huge search space, it is computationally expensive to perform an exhaustive search to obtain the optimal execution plan. The challenge to the database community remains to develop efficient heuristics that not only able to prune the search space effectively but also to ensure that the plan obtained from the restricted space is near-optimal, if not optimal.

References

[Robe84] Roberts, F. S., Applied Combinatorics, Prentice-Hall, 1984.

[Swam88] Swami, A. and Gupta, A., "Optimization of Large Join Queries," *Proc. SIGMOD 1988*, pp. 8-17.