The Choice of the Solution Space for Optimisation of Parallel Queries with Large Joins

Khalid Abdulaziz Al Nafjan
Department of Electronic Technology
Riyadh College of Technology – Saudi Arabia
Knafjan@rct.edu.sa

Jon kerridge
Department of Computing Studies
Napier University
Edinburgh, UK
J.kerridge@dcs.napier.ac.uk

Abstract

The choice of the correct solution space to be searched by a query optimiser is one of the important factors that govern the success of any query optimiser. The importance of the solution space becomes more important for parallel queries that consist of large number of joins due to the increase of the size of the solution space. In this paper we study the effect of varying different parameters on the optimisation of parallel queries that consist of large number of joins. We perceive the effect of these parameters on the choice of the solution space.

Key words: Parallel Query Optimisation, Solution Space, Optimisation parameters

1. Introduction

A key factor that contributes to the success of a database system is the effectiveness of its query optimiser in producing optimal query execution plans. Parallelism is a very good solution for decreasing the response time of queries execution. However it adds to the complexity of query optimisation due to the introduction of different factors. An important element that influences the performance of the query optimiser -which has been largely ignored- is the choice of the proper solution space to be search by the query optimiser. The solution space of a query is the set of possible plans that can execute the query.

The choice of the correct solution space is very important for the optimisation of complex queries such as those that consist of large number of joins. If the incorrect solution space has been searched, then not only a worse query execution plan might be obtained, but also more overheads maybe incurred on the optimiser searching the large solution space.

In this paper we emphasise on queries with large number of joins. That type of complex queries is important for several emerging database applications.

Different research has been conducted in the area of optimising queries that consist of large number of joins in centralised [Swam88,Ioan90] and parallel databases [Chen95, schn90]. However to the best of our knowledge, non of these has investigated the appropriate solution space to be searched. Due to its superior performance especially for ad-hoc queries, the join algorithm used is the hash-based join algorithm.

2. Problem Formulation

The solution space of a query defines in an abstract way the set of all possible query execution plans that can execute the query where each point in the space represents a possible plan. For queries with large number of joins, the solution space becomes much larger and it is not possible for the query optimiser to spend a substantial amount of time for the selection of the suitable plan.

Different types of solution spaces can be searched. This include left deep, right deep, bushy, segmented right deep, and zigzag. Amongst these solution spaces, in this paper we considers the use of right deep and segmented right deep because of their ability to achieve high levels of query parallelism [Schn90,Chen95].

Because the solution spaces are of very large sizes, traditional search strategies such as exhaustive search [Seli79] are no longer applicable. To tackle this problem we have used Genetic Algorithms (GAs) which have proven to be an efficient search strategy for large solution spaces in centralised [Stei93] and parallel databases [Nafj97a,Nafj97b]. We do not provide any details about the concepts of GAs or their use for query optimisation. The

interested reader can refer to different publications about GAs (e.g. [Mitch94]) and to [Stei93,Nafj97a] for their use in query optimisation.
It is hard to specify rigorous rules using which the suitable solution space can be identified, especially for ad-hoc queries where each query represents an individual case. However, our objective in this paper is to try to understand the circumstances under which either of the two solution spaces is better employed. Basically we take a closer look at the effect of varying some parameters of the optimisation process. In particular the impact of varying three parameters on the two solution spaces is considered. These parameters are:

- Optimisation time
- Join selectivity factors
- Processors memory size

3. Experimental Setup

All experiments in this paper were designed to measure the performance of right deep solution space compared with that of segmented right deep solution space while varying one of the parameters mentioned in Section 2. For the comparison of the two solution spaces, arbitrary queries were chosen and the optimal join processing trees of both solution spaces are compared for each query. Each query consists of 30 relations to be joined. Relations that participate in each single query are classified under four categories as specified in Table 1. As in real applications, the Cardinality of these relations varies considerably. Sizes have been chosen so that we have relations with small, medium, large and very large sizes.

Relation Type	Size (Rows)	Percentage of Total Relations
Small	10,000- 200,000	30 %
Medium	200,000 - 1,000,000	25 %
Large	1,000,000 - 5,000,000	35 %
Very Large	5,000,000 - 10,000,000	10 %

Table 1: Relations categories

According to these percentages, the accumulative size of base relations for each query execution plan would be in the range of 50 to 60 million rows. Row size for each relation is fixed at 100 bytes. Hence, the required memory to accommodate these relations at processing is equivalent to 5000 to 6000 Megabyte of the system memory. Of course this amount of memory is not allocated at the same time since relations are processed in segments and do not need to be loaded in the system memory at once. Further, different selectivities were used in our experiments which would produce intermediate relations with sizes ranging from 10% up to 500% of the sum of the two relations to be joined. They have been chosen so that resulting intermediate relations are neither very small nor too large. Although these figures are somewhat arbitrary, they have shown to be suitable for our experiments.

4. Results

4.1 Effect of Increasing Number of Generations

In this experiment we have measured the performance of the optimiser by changing the allowed running time. For the genetic algorithm, the allowed time given to the query optimiser is determined by the number of generations. The larger the number of generations, the more time given to the query optimiser.

Each query has been run using a number of generations ranging from 100 to 600. The best join processing trees for all queries in both solution spaces have then been compared. Figure 2 shows the average performance of the query optimiser.

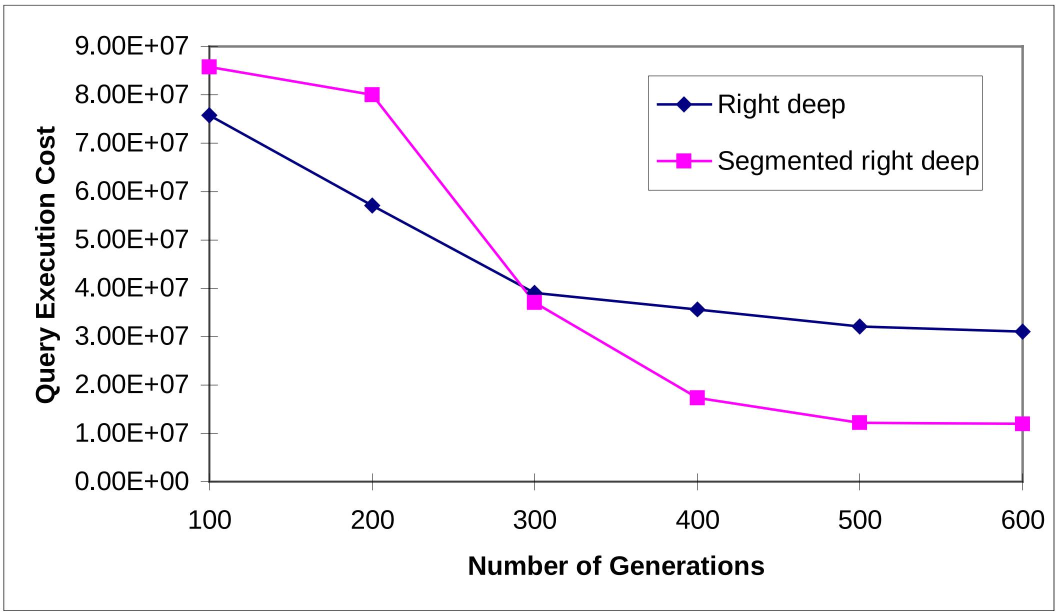

Figure 2: Effect of increasing number of generations on the query optimiser

From the Figure, it can be seen clearly that at small number of generations (i.e., 100 and 200), the query optimiser was able to find join processing trees in the right deep solution space which are notably better than those found in the segmented right deep solution space. When increasing the number of generations to 300, the quality of join processing trees for both solution spaces is improved considerably. However, the effect of further increase of generations is advantageous only for the quality of join processing trees using the segmented right deep solution space, whereas it is slight for the right deep solution space.

These results were expected, due to the size of both solution spaces. Because of the flexibility of its structure, the segmented right deep solution space usually has very good join processing trees. However because its size is large, it needs more time to be searched than the right deep solution space.

4.2 Effect of Varying Join Selectivity Factors

Join selectivity factors are the means by which the size of the resulting relation of a join is determined. In this Section we observe the effect of changing these selectivity factors on the performance of the query optimiser for the two solution spaces. Figure 3 shows the average cost of the best obtained join processing trees for both solution spaces.

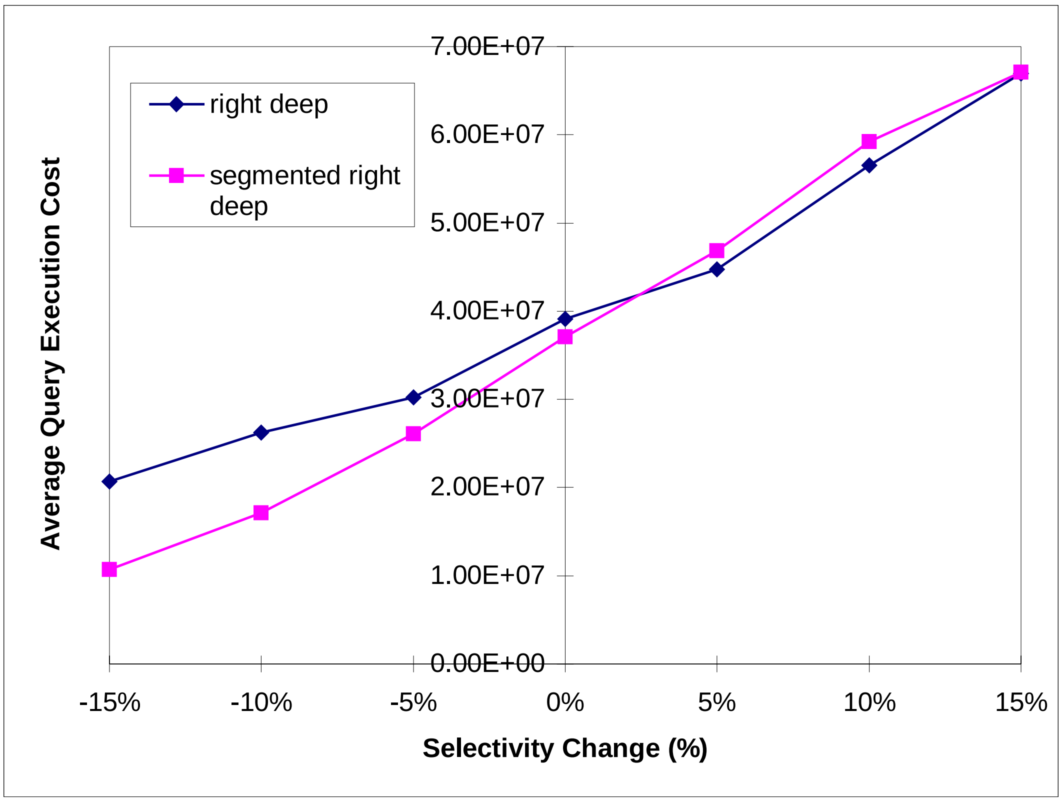

Figure 3: Effect of varying join selectivity factors

From the Figure, it is very obvious that when the join selectivity factors are reduced, the query optimiser shows a better performance using the segmented right deep solution space. On the other hand, when the join selectivity factors are increased, the performance of the optimiser is slightly better using the right deep solution space. To explain these results we have to understand the effect placed on the optimised query when varying the join selectivity factors.

When increasing the join selectivity factors, the sizes of intermediate relations resulting from successive joins are increased. Conversely, decreasing the join selectivity factors will lead to decreasing the sizes of these intermediate relations.

4.3 Effect of Memory Size

In this Section we study the effect of varying the memory size of each processor in the system. We observe the performance of the query optimiser using both solution spaces while changing the amount of memory that the system can accommodate. Basically, for different runs of the same queries, the total memory of each processor is changed. Figure 4 shows the performance results for different runs using memory sizes ranging from 20 to 70 megabytes per processor.

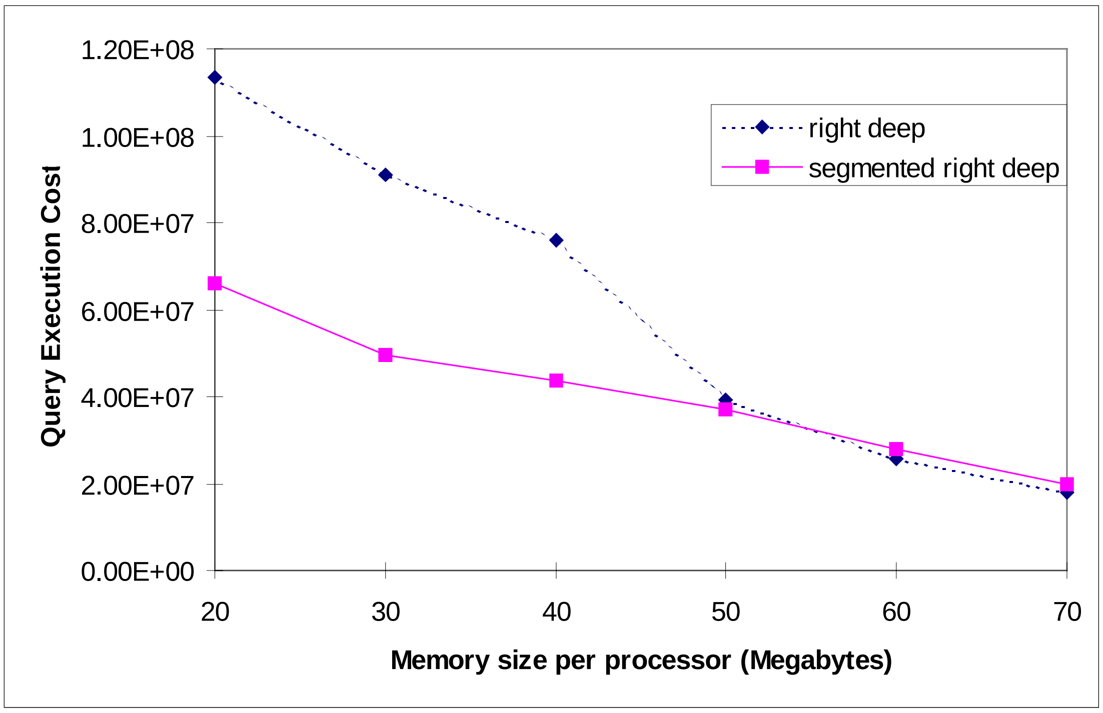

Figure 4: Effect of changing memory size on the performance of the query optimiser.

Several observations are noted from this Figure. Decreasing the amount of memory per processor implies that a join processing tree will consist of more segments because each segment will accommodate a smaller number of relations. Since the resulting intermediate relation of each segment is written back to storage devices, the number of disk I/Os will increase by the increase in the number of segments. This reason explains the rapid increase in cost for both solution spaces when memory is decreased.

However, the cost increase for the segmented right deep solution space is not as significant as that of right deep solution space. This is expected to be due to the fact that increasing the number of segments has given the segmented right deep solution space more opportunity to demonstrate its structure flexibility by using more intermediate relations as building tables in the join processing tree. As in the last experiment, using intermediate relations as building relations can lead to more efficient use of the system memory. In contrast, when the amount of memory is increased, this has led to a decreasing the number of segments per join processing tree since each segment can accommodate more relations. Hence, the flexibility of segmented right deep solution space is not as fully utilised as in the case of smaller memory.

In fact, in the case of processors with large memory, right deep solution spaces show a slightly better performance. This is because for this amount of available memory, the number of segments becomes very small, whereas right deep solution space has a smaller solution space to be investigated, which made it a better choice in this particular case. Again, the conclusion found in the previous Section can be further used to filter our understanding of the present results in this Section. In the previous Section we found that the query optimiser can find better join processing trees in the segmented right deep solution spaces as long as the intermediate relations resulting from each segment are smaller than the remaining original base relations. This conclusion can be used here as a condition for the segmented right deep solution spaces to perform well under small sizes of processors memory.

5 Conclusions and Future Work

The factors that can determine the appropriate solution space to be used by the query optimiser are interrelated. No one of these factors can be used in isolation with the others. The interrelation of these factors are as follow:

In general, if enough execution time is given to the query optimiser, then it will find better or at least similar join processing trees in the segmented right deep solution space when compared to those that can be found in the right deep solution space. This observation is especially true when the join selectivity factors are small so that small intermediate relations result from each segment in the join processing tree. If the join selectivity factors are large then large intermediate relations will result by executing each segment in the join processing tree. In this case, segmented right deep solution spaces are not very efficient since they will not be able to decently employ the resulting intermediates.

Further, the amount of memory that is available to the query optimiser can also affect the choice of the solution space. If the allocated memory to each processor is small then this will lead to larger number of segments per join processing tree. For segmented right deep solution spaces, this case is beneficial since more intermediate relations can be utilised as building relations and hence the system memory is consumed efficiently. However this case can be advantageous only if small join selectivity factors are used and hence the resulting intermediate relations are small enough to be used as building relations. On this basis, we also expect that increasing the size of base relations that participate in the query will have similar effect to increasing join selectivity factors. This is because increasing base relations will increase the number of segments in a join processing tree.

An important direction for the future work is the utilisation of the bushy solution space and hence independent parallelism. Efficient scheduling strategies are essential to overcome the barriers that have prevented the use of this kind of solution spaces.

References

[Chen95] M.S. Chen, M. L. Lo, P.S Yu, and H. C. Young. Applying segmented right deep trees to pipelining multiple hash joins, *IEEE Transactions on Knowledge and Data Engineering.* Vol. 7, No. 4, 1995.

[Ioan90] Y. Ioannidis. Randomized Algorithms for Optimizing Large Join Queries**,** *Proceedings of the ACM-SIGMOD Conference on Management of Data 91,* May 1990.

[Nafj97a] K. Nafjan and J. Kerridge. Large Join Optimisation on Parallel Shared-Nothing Database Machines Using Genetic Algorithms, *European Conference on Parallel Processing (Euro-par-97),* Germany, 1997.

[Nafj97b] K. Nafjan, Optimisation of Parallel Queries, *PhD Thesis,* Department of Computer Science, University of Sheffield, 1997.

[Schn90] D. Schneider and D. DeWitt. Tradeoffs In Processing Complex Join Queries via Hashing in Multiprocessor Database Machines", *Proceedings of the 16th VLDB Conference,* Australia,1990.

[Stei93] M. Steinbrunn, G. Moerkotte, and A. Kemper. Optimizing Join Orders, *Technical Report MIP-9307,* Passau University,1993.

[Swam88] A. Swami and A. Gupta. Optimization of Large Join Queries**.** Proceedings of the ACM SIGMOD conf. On Management of Data, IL, USA, May 1988.