The Way of Combining Inter-Operational and Intra-Operational Forms of Query Parallelism in Databases

Yulia A. Shichkina[1], Mikhail S. Kupriyanov[2]
Saint Petersburg Electrotechnical University "LETI"
[1]strange.y@mail.ru
[2]mikhail.kupriyanov@gmail.com

Armando de Jesús Plasencia Salgueiro[3]
Instituto de Cibernética, Matemática y Física
La Habana, Cuba
[3]ymauri@gmail.com, armando@icimaf.cu

***Abstract*— The development of multi-core and multiprocessor computing systems would seem to significantly speed up the processing of data from various types of databases. This article describes the problems that arise when trying to parallelize the processing of data in relational databases. To solve one of these problems, the article suggests an approach to combining inter-operational and intra-operational forms of query parallelism, which is based on the theory of oriented graphs. The basis of it is to create a graph on the inter-operational form of query parallelism, and then to transform it, taking into account the fragmentation of relations and the receipt of an intra-operational form.**

Keywords— database; quiry; parallel computing; optimization; interoperational and intra-operational forms of parallelism, fragmentation of relations

I. INTRODUCTION

Modern relational DBMSs are usually highly developed query optimizers. So SQL Server provides parallel processing of queries, optimizing the execution of queries and operations with indexes on computers where several microprocessors (CPUs) are installed. During query optimization, SQL Server attempts to detect operations with indexes that can be accelerated by parallel execution. For such queries, SQL Server inserts the exchange operations into the execution plan to prepare a query for parallel processing. After inserting the exchange operators, is obtained a plan for parallel execution of the query [1].

PostgreSQL can generate query plans that will use multiple CPUs to respond faster to queries. But, the developers argue that for many queries parallel execution does not give any acceleration, either because of the limitations of the current implementation, or because of the fundamental impossibility of constructing a parallel plan that would be faster than a sequential one [2].

In MySql query parallelism is realized even worse. The developers themselves argue that parallelism is one of the most difficult problems for DBMS [3]. The parallelism of queries in MySql is achieved by dividing the query between different clients and connections.

Thus, there is a situation when the increase in the volumes of relational databases and the requirements for the speed of data processing is much faster than the development of methods for accelerating the processing of data, including methods for automatically creating parallel queries.

Studies in this area are conducted continuously. Thus, in [8] is given an analysis of the types of parallelism that exist in queries, although there are no recommendations for their implementation. The authors of [4] consider a class of SQL-queries that include operations «join» and «group by». These studies are based on the theorem on sufficient conditions for the pre-group transformation [9]. At the same time, questions related to the nesting of subqueries are not solved. Similar studies, but with the addition of sorting operators are done by the authors [7].

The problem of minimizing the response time to a query with a multithreaded connection using inter-operational parallelism in parallel and distributed environments is considered in [6]. The task of scheduling queries in this environment is represented as the "thread maximization problem", and its solution gives an algorithm that statically finds the optimal set of plans for queries executed in multithreaded connections. If the query is one, but large, then the problem of its decomposition by threads is not solved by the authors.

The work [5] is devoted to speeding up the processing of requests using the cache. This article presents an algorithm for selecting a cache, estimating the cost of its use under certain conditions, adapting the cache for changing conditions. The approach to cache optimization is based on successive plans in the form of trees.

In general, despite the variety and number of studies in improving the process of obtaining data from various databases, currently the methods and software for optimizing storage schemes and the equivalent conversion of queries to a form that is most effectively implemented on computer systems that allow parallel execution of queries , very little and they are far from perfect. As for optimization of queries to relational databases, in this area of research the model for converting SQL queries to the structure of the query tree prevails [10].

This article presents a number of problems that each researcher must solve when creating a parallel query plan and a general approach to solving these problems.

The paper has been supported by grant of Russian Fund for Basic Research (18-57-34001 Cuba_t).

978-1-5386-5824-6/18/$31.00 ©2018 IEEE

II. SOME PROBLEMS OF CONSTRUCTING THE PARALLEL QUERIES PLAN

Problem 1. Choosing the form of parallelism. As is known [11], there are several forms of parallelism used in parallel DBMSs.

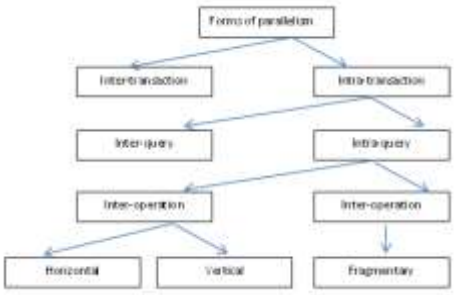

Fig. 1. Classification of forms of parallel data processing

When a user has one, but a complex query and applies to a large amount of data, he can only use intra-query parallelism. In this case, the user has to decide what will be better: apply inter-operational or intra- operational parallelism. Inter- operational parallelism involves the parallel execution of relational operations that belong to the same query plan. Horizontal parallelism involves parallel execution of independent tree subtrees representing the query plan. The main problem with horizontal parallelism is that it is very difficult to make the two subplanes of the same plan begin to generate results at the right time and at the right pace. Vertical (pipeline) parallelism involves the organization of parallel execution of various operations of the query plan on the basis of the pipeline mechanism. The main disadvantage of the synchronous pipeline is the blocking nature of the pipeline processing operations for individual data portions.

Intra-operation parallelism is realized mainly in the form of fragmental parallelism [12], which involves dividing the non-overlapping parts of the relationship, which is the argument of the relational operation [13].

Problem 2. Determination of independent operations in the query. Traditionally, a query tree is used to solve this problem. The actual task in solving this problem is the choice of equal in time execution of subqueries. Especially, this is important if the results of these subqueries are the arguments of the general query.

Problem 3. Choice of the form of fragmentation of the relationship. In relational database systems, fragmentation is divided into vertical and horizontal [14]. The problem, in turn, is divided into two: the choice of the form of fragmentation and the choice of the method of fragmentation. After that, with vertical fragmentation there will be a task of merging the results, and with horizontal fragmentation there will be a problem of dividing into equal fragments.

Problem 4. Combining inter-operational and intra-operational parallelism. Acceleration of the execution of the request depends on the degree and place in which this or that form of parallelism will be applied. And practice shows that acceleration can be negative.

Problem 5. Evaluation of overhead costs. Unlike classical algorithms, the amount of overhead can be very high in queries due to the processed data volumes and the specific operation of the DBMS. For example, the query:

```
Select атрибуты
 From Table1                                        (1)
Where (condition 1) and (attribute <= values);
```

is much slower than the query

```
Select атрибуты
 From Table1                                        (2)
Where (condition 1).
```

Therefore, dividing query (2) into two queries (1) with horizontal fragmentation will be effective only if the data volume is greater than some constant, which can be calculated statistically.

Problem 6. Static evaluation of query acceleration for various forms of parallelism, caching forms, indexing, actions of the embedded DBMS optimizer. For an ideal plan for parallel execution of the query, it is necessary to conduct a full study with an assessment of the degree of influence of various parameters on the speed of query execution and the amount of overhead. Doing this for any one-time query does not make sense, because this process is too time consuming.

Problem 7. Selection of an optimal plan taking into account the solution of the problem 6.

There are no methods to solve these problems completely, unambiguously and universally for any query to any relational database on any computer system. But in the theory of parallel computations there are apparatus and methods [15, 16], which allow to reduce these problems.

III. APPLICATION OF GRAPH THEORY FOR COMBINED INTER- AND INTRA- OPERATIONAL PARALLELISM

To create a parallel query plan, in the first approximation, the researcher can use the method of parallelizing classical algorithms, based on the adjacent lists of the information graph of the algorithm [15]. The query information graph is a directed graph in which the vertices are subqueries and the directed edge connects the vertices *a* and *b* if the subquery corresponding to vertex *b* uses the result of the query corresponding to vertex *a* as input. Because in the majority of queries to relational databases very rarely have a degree of nesting more than three, rarely contain more than 10 tiers, then the creation of an initial information graph is possible manually. Moreover, all methods of optimizing the information graph for various parameters (time, computational nodes, the amount of inter-processor transfers, etc.), developed for classical algorithms, are completely suitable for queries. But, with all the positive properties and capabilities of these methods, queries to relational databases have their own specifics and therefore direct application of the listed methods will at best give some acceleration, and in the worst case will not have any effect. First, this is due to the fact that the input data for queries are whole tables and they contain a very large amount of data. Secondly, the information graph gives information only about task parallelism, but queries that would

contain very many independent subqueries are, in practice, rare. Therefore, often on the informational graph, in the first approximation, the volume of internal parallelism is zero.

Let's consider several variants of the query, consisting of the construct «select», «from», «where».

Option 1. The query S is performed only on one table T. The query S is a subquery to other queries $S_1..S_k$, k ∈N. The information graph of such query is shown in Fig. 2a.

First, consider the case where the query S does not contain the aggregation function avg.

Parallelization of the query will be done by dividing the table T by some attribute into two tables T_1 and T_2: $T = T_1 \cup T_2$, where the symbol "∪" denotes the union operation in SQL. Then the information graph of the query will change and take the form of the graph from Fig. 2b.

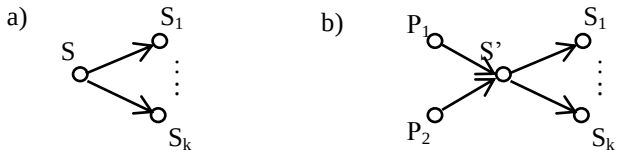

Fig. 2. Graphs with single and divided query tables

Variants of the query S', depending on the expression in the construct «Select», are presented in Table 1 below.

TABLE I. ACCORDANCE OF INITIAL AND FINAL FORMS OF REQUEST

№	Предложение Select	S'
1.	Select max(d1)…	Select max(@m1,@m2);
2.	Select min(d1)…	Select min(@m1,@m2);
3.	Select sum(d1)…	Select sum(@m1,@m2);
4.	Select count(d1)…	Select sum(@m1,@m2);

Now, suppose that there is no aggregate function in the query S. Then the query S in accordance with the graph in Fig. 1b) will correspond to the query S ':

Select @m1 union @m2;

Let's consider the last variant of a simple query - the case when the query S contains the aggregation function avg. Such queries can also be represented by the scheme from Fig. 2b. They will correspond to S ':

Select (@m1+@m2)/(@c1+@c2);

Thus, independently of the type of aggregation function and its presence in the construct «Select», all queries of the form "select ... from T where ..." can be represented with horizontal fragmentation by the Fig. 2b.

Option 2. The query S is made only to several tables T_1 and T_2:

Select attributes
From T_1, T_2
Where (condition) and (T_1.key=T_2.key);

The query S is a subquery to other queries $S_1..S_k$, k ∈N. The information graph of such a query will also correspond to Fig. 1a.

Let each of the tables T_1 and T_2 be divided in some way (it does not matter in the current context exactly) into two parts: $T_1 = T_{11} \cup T_{12}$, $T_2 = T_{21} \cup T_{22}$. Then, by the property of a natural compound, get:

$$T_1 \times T_2 = (T_{11} \cup T_{12}) \times (T_{21} \cup T_{22}) = T_{11} \times T_{21} \cup T_{12} \times T_{21} \cup T_{11} \times T_{22} \cup T_{12} \times T_{22}.$$

In accordance with this rule, the query S will be transformed to the form S':

Select @m_{11} union Select @m_{12} union Select @m_{21} union Select @m_{22};

Accordingly, the query graph then takes the form:

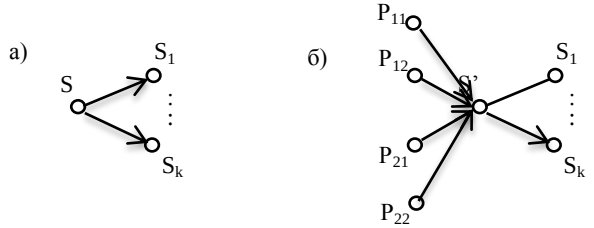

Fig. 3. Graphs with single and divided query tables

Let: k is the number of divisible tables, t is the number of tiers in the information graph of the query, r is the current tier of the graph, n_r is the number of requests on tier r, m_r is the number of queries for option 1, k_i is the number of parts in the i-th table, k_g is the number of vertices added to the initial graph. Then:

$$k_g = \sum_{r=1}^{t} \left(\sum_{i=1}^{m_r} k_{ri} + \sum_{i=m_r+1}^{n_r} \sum_{j=m_r+1}^{n_r} k_{ri} k_{rj} \right)$$

Fig. 4 shows the graphs showing the growth of the total number of vertices, depending on the complexity of the initial graph and the initial number of vertices in the graph when dividing each table into only two parts. Complexity is the product of values: the number of tiers, the average number of vertices on the tier, and the average number of tables in queries.

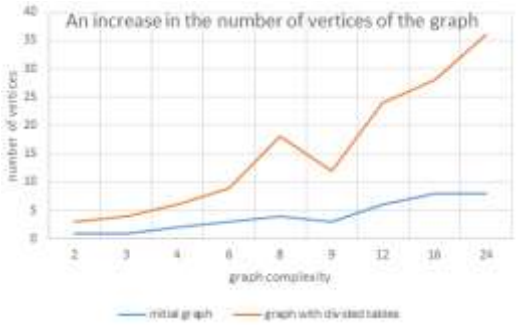

Fig. 4. Graph of increasing complexity of the query graph

According to Fig. 4. It is seen that with increasing complexity of the graph, the number of vertices in the modified graph, taking into account the tables divided into parts, increases very strongly. Manually constructing such a graph is

very laborious, and it is even more difficult to analyze it for constructing an optimal parallel query execution plan. In this case, the parallel computation methods help [15, 16].

IV. CONCLUSION

The analysis of existing solutions in the field of parallelization of queries has shown that today there are many private solutions, both obtained by individual researchers and organizations and implemented in various software products. These private solutions do not allow most users to easily solve their tasks for speeding up queries to relational databases. The article identifies the most significant problems for which a universal apparatus is required. One of such apparatus can be the theory of oriented graphs. In this case, the methods developed for parallelizing classical algorithms, and designed to optimize them according to the number of nodes, the execution time, the number of inter-processor nodes, etc., are also suitable for optimizing queries. But, for their application, the graph of the initial query must be modified. As a result, the query graph allows to combine inter-operational and intra-operational parallelism. And adding to the graph of weights, allows to optimize the query, taking into account such parameters as indexes, overhead, data volume and so on.

REFERENCES

[1] https://technet.microsoft.com/ ru-ru/ library/ ms178065 (v=sql. 105).aspx

[2] https://postgrespro.ru/docs/postgresql/9.6/parallel-query

[3] https://www.seagate.com/ru/ru/our-story/data-age-2025/

[4] M.Al.Hajj Hassian and M.Bamba, Parallel processing of Group by join queries on nothing shared machines, ICSOFT 2006, Software and Data Technologies, pp 230-241.

[5] S. Babu, K. Munagala, J. Widom, and R. Motwani. Adaptive caching for continuous queries. In ICDE, 2005, pp.118-129.

[6] A. Dasphande, L.Hellester, Flow algorithm for parallel query optimization. In:24th International Conference on Data Ingineering , pp.754-763. IEEE,Cancun, 2008.

[7] R. Acker, C.Roth, R.Bayer, Parallel query processing in databases on multicore architecture ICA3PP 2008: Algorithms and Architectures for Parallel Processing pp 2-13.

[8] P. Mohankumar, P. Kumaresan and Dr. J. Vaideeswaran, Optimism analysis of parallel queries in databases through multicores International Journal of Database Management Systems (IJDMS), Vol.3, No.1, February 2011, pp.156-164.

[9] Tsois, A., Sellis, T.K.: The generalized pre-grouping transformation: Aggregate-query optimization in the presence of dependencies. In: VLDB, pp. 644–655 (2003).

[10] Spiliopoulou M., Hatzopoulos M., Ttanslation of SQL queries into a graph structure: query transformations and pre-optimization issues in a pipeline multiprocessor environment, Informafion Sysfems 1992, Vol. 17, No. 2, pp. 161-170.

[11] L.B. Sokolinskij. Parallel'nye sistemy baz dannyh. Izdatel'stvo MGU, 2013, 184 s. (*In Russian*)

[12] DeWitt D.J., Gray J. Parallel Database Systems: The Future of HighPerformance Database Systems // Communications of the ACM. - 1992. -Vol. 35, No. 6. -P. 85-98.

[13] Graefe G. Query evaluation techniques for large databases // ACM Computing Surveys. 1993. Vol. 25, No. 2. -P. 73-169.

[14] Williams M.H., Zhou S. Data Placement in Parallel Database Systems // Parallel database techniques. IEEE Computer society, 1998. P. 203-218.

[15] Shichkina, Y.A., Kupriyanov, M.S. Applying the list method to the transformation of parallel algorithms into account temporal characteristics of operations, Proceedings of the 19th International Conference on Soft Computing and Measurements, SCM 2016, 7519759, c. 292-295.

[16] Shichkina, J., Degtyarev, A., Kulik, B., Fridman, A. Optimization of relational databases schemas by means of n-tuple algebra, AIP Conference Proceedings, 1863,110008.