IJCTA, 9(30), 2016, pp. 169-174
© International Science Press

Special Aspects of Virtualization Control of Query Service for Very Large Databases Based on Massively Multiprocessing Models

Gennady Vladimirovich Abramov* Victor Leonidovich Burkovskiy Vera Nikolaevna Kostrova** Oleg Jakovlevich Kravets** and Oleg Valerjevich Rodionov****

Abstract : In this work we review special aspects of virtualization control of query service for very large databases based on massively multiprocessing models. The aim of this work is to increase the speed of interaction between the individual computing nodes of the systems based on multiprocessing. The novelty of the described solutions is to provide each computing node of the system with its own copy of the data. This will allow us to implement operations on parallel data processing to minimize interactions between network nodes.

In this work we describe methods of implementing several algorithms for data updating. In one method, data update may occur through the suspension of the system operation for a certain period of time during which we are adding or changing already existing dataset. In this case it is the easiest to perform the data updating typical for the classic data storage, in which there is no need to update previously accumulated data. Also it is possible to implement the updating of information without suspending the system operation.

Consequently, we propose parallel query preprocessor which should generate a set of equivalent queries with respect to a given query tree. These query trees should contain the required number of subtrees, the performance of which may be done in parallel. On the basis of valuation, we should select one of these trees, according to which the query execution will be performed with minimal overhead.

Keywords : database query, service virtualization, parallel processing, preprocessor, equivalent query.

1. INTRODUCTION

As follows from [1], a solution to process queries against a very large databases is a replication of data for its parallel processing. Very large databases are characterized by the fact that the amount of stored data exceeds the amount of RAM available to the system by many orders of magnitude. For effective system computational capacity loading, it is required to provide high-capacity of the disk subsystem. It appears that an effective way to separate the data to be processed is to do it on the basis of the ranges of values to be processed.

Methods of separation used to store the data based on ranges do not allow the separating operation to be performed dynamically. In addition, the possible load imbalance on nodes of the system may occur and it is not amenable to a programmable control. Data separation method based on hashing does not allow working effectively with data separated into ranges. Data placement based on hashing is also performed taking into account a priori specified set of attributes. Methods are intended for use in systems based on symmetric multi-processing (SMP).

Systems based on massive multi-processing (MPP) allow us to achieve greater computing capabilities at lower costs and provide better scalability compared to SMP. The main disadvantage of MPP [2] is a low (compared

* Voronezh state university University sq., 1, Voronezh, 394000, Russian Federation

** Voronezh state technical university Moscow ave., 14, Voronezh, 394026, Russian Federation

to SMP) rate of interaction between the individual computing nodes. It is suggested to solve this problem by providing each MPP computing node with its own copy of the data. Then we can implement operations on parallel data processing by minimizing the interactions between network nodes. It should be noted that under this approach, separate SMPs can act as the MPP nodes. This architecture is called mixed or hybrid architecture and it allows to achieve better productivity [3] due to the possibility of applying methods of optimization of SMP system-oriented query performance.

2. METHODS

We describe a system based on MPP architecture which enables parallel processing of SQL queries. The system consists of N nodes on which SQL servers are installed and they are not required to be uniform. To access the system it is necessary to use a subset of the SQL language that is common to all servers. If our DBMS contains SQL statements which differ only in syntactic form it becomes possible, by applying trivial transformations, to use the syntax of the SQL language which is not common to all SQL servers of the system. However, there are some structures that do not allow trivial conversions. For example, the use of hierarchical queries, if at least one of the SQL servers do not support this possibility. Such restrictions on the use of statements in queries do not arise if we require uniformity of SQL servers of the system. Suppose MPP based database is available. Suppose that the dataset of interest is replicated to N servers of the distributed system.

Under the consistency of the dataset on two nodes we mean the identity of the result of any query to read the data, performed either on the first node or the second node. The considered set is to be consistent at any time of the system operation for all combinations of nodes in order to implement parallel processing. This means that at the time of starting the system operation datasets on its nodes should be synchronized.

In the system it is possible to implement several algorithms for data updating. For example, the data update may occur through the suspension of the operation of the system for a certain period of time during which already existing dataset is to be added or changed. At the same time the data updating that is typical for the classic data storage in which there is no need to update previously accumulated data is the easiest to perform. It is possible to perform the data updating without suspending the operation of the system. Then the data update in all nodes must be performed within a single distributed transaction, so that changes in different nodes of the system become available in synchronous mode to users who do reading. This behavior of the system can be achieved by using the locking schemes implemented in the DBMS or on the basis of the timestamp concept when the time of record entering into the database is taken into account during the query processing and records made later than a specified time are ignored.

It is possible to refuse identity of all copies at any time, however it will be possible to work with the database as a whole only in a mode equivalent to dirty read which may be sufficient for operation of the applications. There may be a problem of inconsistent processing – non-repeatable read [4]. A problem of this kind during the performance of parallel queries does not depend on the isolation level of modifying transactions on the local sites. This requirement is necessary to ensure a higher level of isolation of user transactions.

In this system it is possible to implement the mixed concept of data processing – both OLAP (Online Analytical Processing) and OLTP (Online Transaction Processing). To do this, it is advisable to select one “powerful” server to support OLTP part of the database where the information will not be subjected to replication. The rest of the system ensuring the functioning of OLAP components is to be implemented according to the proposed above scheme. Then the complete information for the database will consist of one copy of the data from OLAP part of the system and of OLTP server data. Getting data from both the OLAP part of the system and the OLTP part can be solved by means of DBMS on the basis of the concept of data separation. The data separation point can be selected, for example, as the period closing date on the condition that the data after closing of the relevant period is not subject to editing. In general, consistent changes of random data in the system is à complex (from the point of view of overhead) task in any of the proposed implementations of the system and the implementation of this option seems to be impractical.

The proposed scheme of construction of the system allows to accelerate the execution of SQL queries to read data by reducing the speed of execution of queries to insert/modify/delete data (hereinafter modification). There is a need to update the replicated data within a transaction and it adds the overhead and also makes some limitations. The last of the proposed options of the system implementation represents a compromise between the increase in the overhead costs for data modifying and increase in process rate.

For the described above distributed system, we propose the following parallel algorithm of the execution of the SQL query. The user query initially received on one of the nodes of the system is to be prepared for parallel execution. The possibility of this parallel execution is achieved by transforming the original SQL query to a set of queries which execution result is equivalent to the original query [5, 6] (after combining the new query execution results, additional computing of aggregate functions and grouping, as well as the result reordering).

The set of queries obtained as a result of transformation can be executed on any node of the system independently from each other. This follows from the requirement for identity of data copies and equivalence of the transformed query to an original query. A set of rules for query transformation is pre-fixed, the expedience of use of transformation rules is determined by the type of original query and the statistical data available for the table fields used in the query. This approach allows us to select dynamically the algorithm of parallel execution of a query depending on the database contents.

The key point that can be used effectively in the implementation of the approach is the addition of intermediary element to the sequence of actions responsible for the query execution. Essentially, in practical implementation of the proposed solution it is possible to build a system based on three-tier architecture (Figure 1, Figure 2).

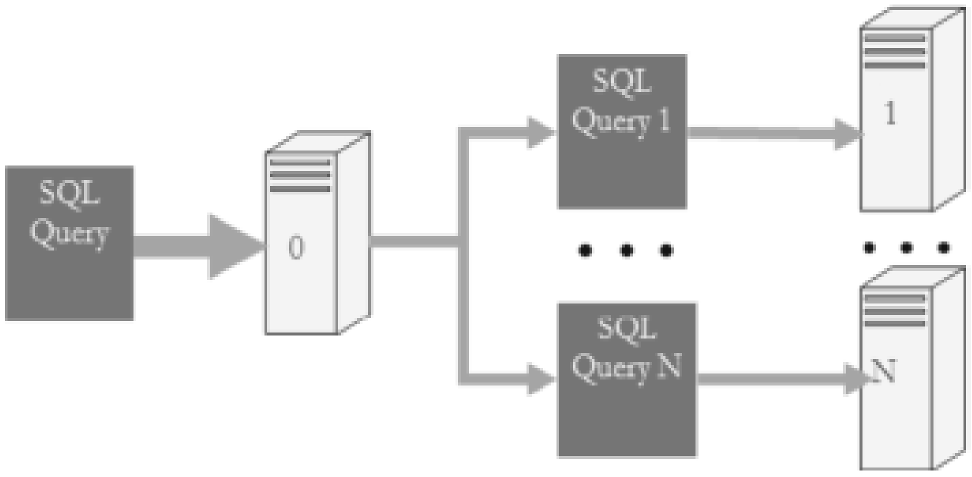

Fig. 1. Query.

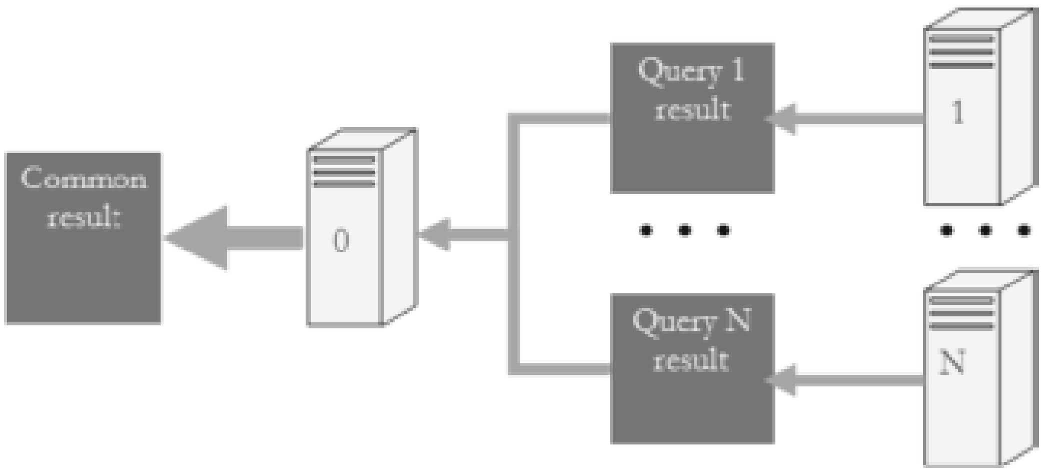

Fig. 2. Result.

Intermediary element between the application and DBMS servers plays the role of an object that allows getting benefits of parallel processing, thus serving as an additional element. From the point of view of the client application, the query processing remains unchanged regardless from the actual number of servers allocated to handle the query. Software of an intermediate level takes on all functions of coordination of operation of separate servers. System nodes specifically allocated for this purpose and nodes that directly enforce the execution of user queries to a DBMS can take the role of an intermediate operation hardware.

3. DISCUSSION AND RESULTS

From [6] it is known that the circuit of initial stage query compilation consists of four steps: query (textual representation) – parser – preprocessor – query logical plan generator – query logical plan rewriter. We add to steps to this circuit – the parallel query parser and the parallel query preprocessor. These two steps will precede the four classic steps of compilation. The parallel query parser performs the same function as the parser in the classical compilation circuit – transforms text of the original SQL query into a parse tree. Implementation details, methods, marks and features of the parsing process are given in [6, 7, 8]. In contrast to the classical circuit, the parallel query preprocessor in the proposed circuit modifies the query tree for the selection of query subtrees suitable for parallel execution. After the parallel preprocessor operation, a set of new queries is generated. The processing of these new queries is based on the classical circuit. Transformations of query parse trees are performed by the preprocessor using pre-fixed set of rules with the aim of obtaining an equivalent query. In some cases, when transformations are done it may require additional operations on relation sets which are returned by the queries [9, 10].

As an example, consider the query that returns the quantity of shipped goods during the shipment period:

SELECT *sum*(QUANTITY)

FROM LINEITEM

WHERE SHIPDATE > = ‘20150710’ *and* SHIPDATE < = ‘20150720’

The parse tree corresponding to this query will look as follows (Figure 3):

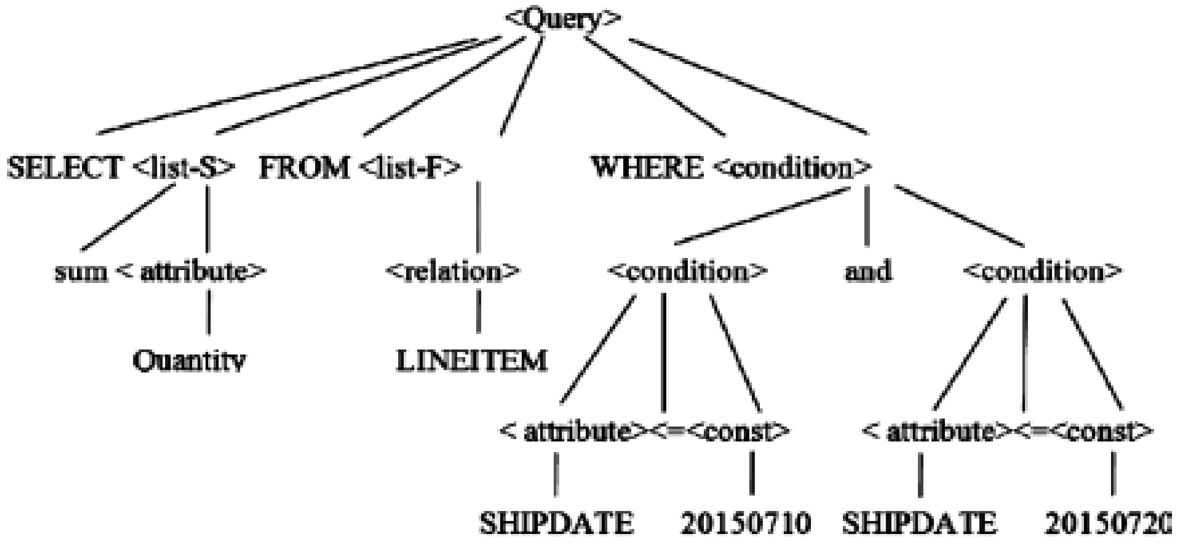

Fig. 3. The parse tree.

We can transform the request equivalently to the form shown in Figure 4, 5.

The equivalence of two queries should be understood as queries which execution results in formation of relations which are the same in all attributes of tuples and accurate to the tuple order, if the sorting instruction is not specified and taking into account the sequence order otherwise.

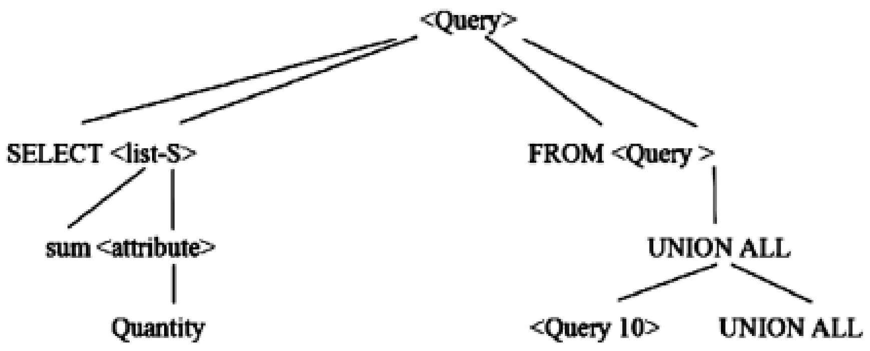

Fig. 4. The request transformation (1)

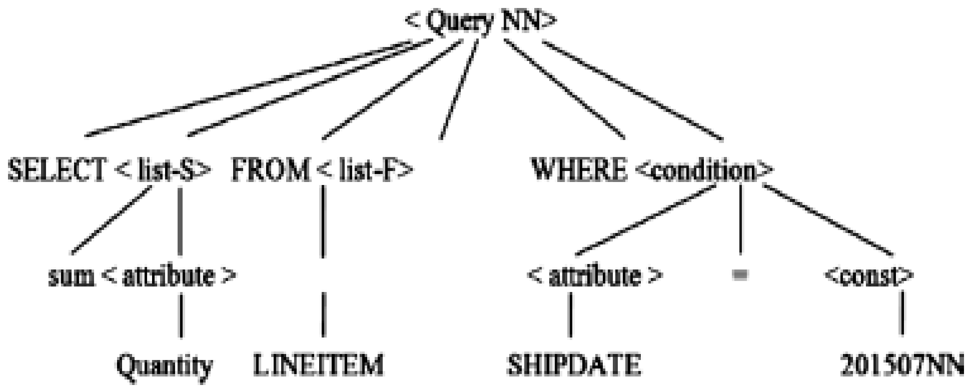

Fig. 5. The request transformation (2)

The SQL query that corresponds to this parse tree will be :

SELECT *sum* (R. QUANTITY)
FROM (
SELECT *sum* (R. QUANTITY)
FROM LINEITEM
WHERE SHIPDATE = ‘20150710’
UNION ALL
…
UNION ALL
SELECT *sum* (R. QUANTITY)
FROM LINEITEM
WHERE SHIPDATE = ‘20150720’
) *as* R

Obviously, uncorrelated queries allow parallel execution, so all the resulting subqueries in the query parse tree can be computed independently. In the general case, further computing of the query according to the parse tree can be carried out only upon receipt of the results of all lower level subqueries and statements in the hierarchy of subqueries and statements.

4. CONCLUSION

We may now state the goals that must be achieved by means of equivalent transformations of queries :

1. The transformation rule should generate from the original query a new query that contains a pre-defined number of uncorrelated subqueries.

2. Received queries should have approximately equal exercise value, as further computing of the query is possible only after computing of the relevant subqueries and in case of significant excess of one subquery execution time over the others, other nodes (not occupied with computing of subquery) of the system can be idle. Query transformation should control the load balance between nodes of the system by subqueries generation.
3. In the upper levels of the query parse tree, transformation should leave the "cheapest" operations, the implementation of which does not require processing of a large number of entries.
4. If possible, transformation should not increase the volume of relations obtained by computing of subqueries in order to eliminate the transmission of large amounts of data between the nodes of the system. Large amounts of such transfers may seriously slow down the query execution and reduce the advantage of parallel execution of the query [11].

In the general case there exist more than one rule for the equivalent transformation of the query which can prepare this query for parallel execution [9], we can achieve different final speed of query execution by choosing different rules. Implementation details of the algorithm for selecting the query transformation will be described in future work.

The parallel query preprocessor should generate a set of equivalent queries with respect to a given query tree. These query trees should contain the required number of subtrees, the performance of which may be done in parallel. On the basis of valuation [1, 3], we should select one of these trees, according to which the query execution will be performed with minimal overhead.

5. REFERENCE LIST

1. Brobst S. and Robertson O. Taming Data Giants. DBMS, 2 (1997) 38.
2. Brobst S. and Robertson O.. Database Systems: A Practical Approach to Design, Implementation and Management. Pearson, 2015.
3. Shekhar, S., Srivastava, J. and Dutta, A Formal Model of Trade-off between Optimization and Execution Costs in Semantic Query Optimization. Proceedin s of the 14th VLDB Conference Los Angeles, Califomia, pp. 457-467, 1988.
4. Watson, R. T. Data Management: Databases & Organizations, JDBSLC, 2005.
5. Sokolinsky L. B. Organization of parallel query execution in multiprocessor database machines with hierarchical architecture. Programming 6 (2001), 13-29.
6. Aho, A. V., Sethi, R. and Ullman, J. D. Compilers: Principles, Techniques, and Tools. Pearson Education, 1986.
7. Lokshin, M. V. and Kravets O. Ya. Building of systems for parallel processing of queries to the DBMS. Telematika'2004: Publications of the XI all-Russian scientific-methodical conference (June, 7-10, 2004). St. Petersburg: ITMO, pp. 94-95.
8. Garcia-Molina H., Ullman J. D., Widom J. Database Systems. Full course. M. Williams, pp. 1088, 2003.
9. Oleinikova S.A., Kravets O.Ya., Zolotukhina E.B., Shkurkin D.V., Kobersy I.S., Shadrina V.V. Mathematical and Software of the Distributed Computing System Work Planning on the Multiagent Approach Basis. International Journal of Applied Engineering Research, 11(4) (2016), 2872-2878,
10. Kravets O.Ya., Makarov O.Yu., Oleinikova S.A., Pitolin V.M., Choporov O.N. Switching subsystems within the framework of distributed operational annunciator and monitoring systems: program design features. Automation and Remote Control, 74 (11) (2013), 1919-1925.
11. Kravets O.Ja. Mathematical Modeling of Parametrized TCP Protocol. Automation and Remote Control, 74 (7) (2013), 1218-1224.