Application of Optimization of Parallel Algorithms to Queries in Relational Databases

Yulia Shichkina[1(✉)], Alexander Degtyarev[2(✉)], Dmitry Gushchanskiy[2], and Oleg Iakushkin[2]

[1] Department of Computer Science and Engineering, Saint Petersburg Electrotechnical University "LETI", St. Petersburg, Russia
strange.y@mail.ru
[2] Saint Petersburg State University, St. Petersburg, Russia
deg@csa.ru, nyapko@gmail.com, oleg.jakushkin@gmail.com

Abstract. All known approaches to parallel data processing in relational client-server database management systems are based only on inter-query parallelism. Nevertheless, it's possible to achieve intra-query parallelism by consideration of a request structure and implementation of mathematical methods of parallel calculations for its equivalent transformation. This article presents an example of complex query parallelization and describes applicability of the graph theory and methods of parallel computing both for query parallelization and optimization.

Keywords: Parallel computing · Optimization methods · Relational database · Query · Information graph

1 Introduction

Nowadays the use of databases and information management systems is an integral part of business processes of enterprises and organizations. The growth of database volumes imposes new and quite strong requirements for the developing hardware infrastructure of data centers. Mark Hurd, co-CEO of Oracle Corporation, believes that the volume of accumulated data will increase fiftyfold by 2020. In 8 years more than 100,000 companies and corporations will be using databases, and the volume of each such database will surpass 1 petabyte [3].

Even today organizations already need powerful tools for transferring, processing and analyzing accumulated information.

The survey performed by Gartner in 2010 showed that the problem of data growth makes the top three most serious problems for 47 % commercial companies. System performance and scalability worry 37 % of organizations, while 36 % of organizations believe that the potential issues are related to network congestion and connectivity. Six in ten questioned organizations announced plans to invest in data retirement efforts. According to April Adams, research director at Gartner, while all the top data center

O. Gervasi et al. (Eds.): ICCSA 2016, Part II, LNCS 9787, pp. 366–378, 2016.
DOI: 10.1007/978-3-319-42108-7_28

hardware infrastructure challenges impact cost to some degree, data growth is particularly associated with increased costs relative to hardware, software, associated maintenance, administration and services [2]. There is one way out: the methods of data processing should be updated in according to available equipment.

During the past twenty years development of databases has been forging ahead. The new branch of database technology called NoSQL emerged, which provides new simple ways of data scaling. And, for processing large amounts of data, for example, those received for IoT with GLONNAS, many companies tend to choose NoSQL solutions. And if yesterday some mechanisms applicable for relational databases were not available for NoSQL, today, for example, fuzzy slices are easily implemented in databases such as MongoDB. However, not every NoSQL technology, especially high scalable ones, has solved the issues related to correspondence of operations to requirements of ACID (atomicity, consistency, isolation, durability) – the standard which guarantees accuracy of operational transactions performance by resources of database management systems in case of a system failure. Today this problem is being handled by a new database systems technology class NewSQL, members of which have combined new approaches of distributed systems from NoSQL with relational data presentation model and data query language SQL. The developers of the NewSQL system VoltDB affirm that it is approximately fiftyfold faster than the traditional OLTP RDBMS.

However, considering NoSQL and NewSQL, it must be noted that:

1. For the absolute majority of big projects related to implementation of database management systems relational DBMS are still preferred, and the market continues maintaining traditional approaches to solving such issues;
2. In spite of the long history of relational databases development, due to the development of new supercomputing approaches it turns out that not all mechanisms of relational databases are actively used and new opportunities open up.
3. Data in relational databases is created, updated, deleted, and queried through the API method calls. API is open to users and allows them to perform maintenance and scaling of its database instances. The most common clients API are JDBC, ODBC, OLE-DB, etc. API-functions “know” how to send a request to the database and how to process the returned cursor. They provide conversion of user queries transmitted over the network as packets, which are processed by a dedicated server. They do not contain embedded parallelization mechanisms though.
4. All modern relational databases use CBO (Cost Based Optimization). It suggests that each operation is determined by its “cost”, and the total cost of the query is reduced through the use of the most “cheap” chain operations. Although this mechanism significantly speeds up query processing, it does not take into the possibility of parallel execution of parts of a query.

Considering the above and the fact that data growth in the existing databases continues, as well as the need of fast data processing increases, the task of creation of an interactive mechanism for parallel query processing becomes more and more relevant. If one manages to create a device which will provide effective parallelizing of queries in relational databases, it will not just extend their lifecycle and expand their sphere of application, but also

will reduce expenses of organizations with informational systems, which are successfully created long ago and based exactly on relational databases.

It should be noted, that, in spite of intensive development of computer systems, new methods, and programming languages, parallel data processing is still the area, where the problem of dividing computation into processes falls on end users (programmers, database managers, etc.). Quality and speed of derivable solutions on data processing most often depend on qualification of the user of a database management system. It is possible to reduce the impact of human factor on the parallel data processing by means of transfer the function of computing the number of CPU needed for handling applied problems and modeling the future parallel query for ECM.

The software for DBMS can provide the following types of parallelism (Fig. 1) [9]:

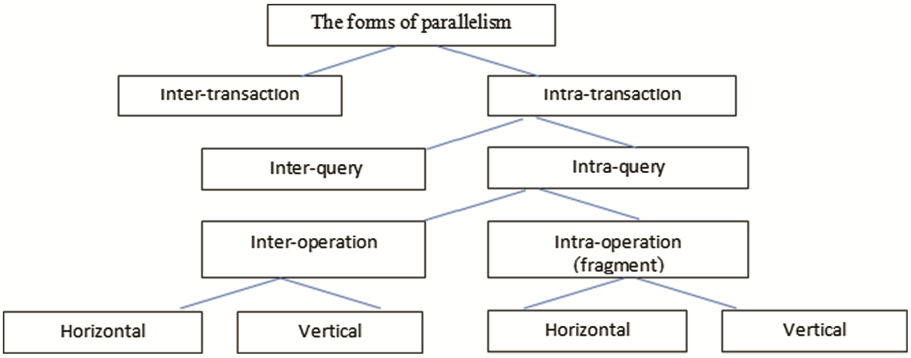

Fig. 1. Types of parallelism

Inter-transaction parallelism is parallel execution of many separate transactions for the same database.

Intra-transaction parallelism is parallel execution of a separate transaction [1].

Inter-query parallelism is a parallel processing of separate SQL queries in the same transaction [6].

Intra-query parallelism is parallel execution of a separate SQL query. This type of parallelism is common in relational databases. This is due to the fact that relational operations on series of tuples are well-adapted for efficient parallelization [4].

Inter-operation parallelism is parallel processing of relational operations on the same query. Inter-operation parallelism can be implemented either as horizontal parallelism or vertical parallelism [5].

Horizontal parallelism is parallel processing of separate operations in a query [7].

Vertical parallelism is a parallel processing of different operations in a query based on pipeline mechanism.

Fragment parallelism is splitting of the relation, which is an argument of a relational operation, into disjoint parts. A single relational operation is performed as several parallel processes (agents), either of which handles an independent fragment of the relation. The obtained resulting fragments merge into combined resulting relation [8].

Fragmentation in relational database systems can be vertical or horizontal. Vertical fragmentation provides splitting of the relation into fragments of columns (attributes). Horizontal fragmentation provides splitting of the relation into fragments of rows (tuples). Almost all parallel database management systems, which sustain fragment parallelism, use only horizontal fragmentation.

This article deals with inter-transaction parallelism based on the mixture of inter-query, intra-query inter-operation horizontal parallelism, and inter-operation parallelism with horizontal fragmentation.

The degree of inter-query parallelism is restricted both by the number of SQL queries, which compose this transaction, and by precedence constraints between separate SQL queries. Therefore the inter-query approach to query parallelization can be successfully combined with the intra-query one for increase of the degree of complex query parallelism in general.

Theoretically intra-query fragment parallelism can provide the arbitrarily high degree of relational operations parallelization. In practice, however, the degree of fragment parallelism can be significantly restricted by two factors. In the first place, fragmentation of the relation always depends on operation semantics. In the second place, a failed fragmentation can lead to significant imbalance in CPU load.

Both inter-query and intra-query types of parallelism require good parallelization skills, understanding of inter-query dependences, and significant time and labor costs for their implementation from a programmer. Particularly for this reason inter-query parallelism is not sustained by majority of modern database management systems. However, research shows that the information graph of a query, through simple modifications in accordance with the principles of inter-query, horizontal, and fragment horizontal parallelism, by its structure is identical to the information graph of an algorithm, and therefore for its analysis and optimization one can implement mathematical methods of parallel computing based on graph theory and adjacency lists.

2 Query Parallelization in Client-Server Databases

For all mass computer models and different operating systems the PC software market offers many commercial database management systems, which vary in their functionality and capabilities. The most popular client-server database management systems are: Microsoft SQL Server, Oracle, Firebird, PostgreSQL, and MySQL. Despite of some of these DBMS have more powerful functional set and others are built with less various data processing functions, the operation principle of all database management systems listed above is the same.

One way to achieve higher efficiency is to use task parallelization algorithms. There are three application areas for such algorithms in DBMS:

- Parallel input/output,
- Parallel administration tools and utilities,
- Parallel processing of database queries.

Parallelization of input/output in conjunction with optimal task planning allows accomplishing quite efficient simultaneous access to fragmented tables and indexes located on several physical disks, thus boosting the operations with comparatively slow external devices manyfold.

In contrast with parallel input/output and administration, parallelism implementation in request processing is considered more difficult. A theoretical foundation of the possibility of query parallelization in relational database management systems is the property of relational closeness. The result of each relational operator: SELECT is selection of subsets of relation (table) rows; PROJECT is selection of subsets of fields (columns); JOIN is combination of two tables – is a new relation, and, as far as any query can be divided into hierarchy of elementary operators, it is rational to try to execute them in parallel. Undoubtedly, parallelism is inherent in SQL internally. Query processing consists of a set of atomic operations, and their structure and sequence are determined by the performance enhancer after the examination of several options.

In client-server DBMS data processing is performed on a server where data is stored. Client applications send requests for processing and receiving data from database management system and receive the answers. Client applications do not have immediate access to data files. Database server is a multi-user version of DBMS with parallel processing of queries coming from all workstations. Its task is to implement the transaction manipulation logic using necessary synchronization methods – maintaining locking protocols for the resources and providing prevention and/or elimination of deadlocks. In response to a user query, a workstation will receive not "raw material" for future processing, but complete results. By such architecture the workstation software represents only the front-end of central database management system. This allows reduction of network traffic, shortening the time of waiting locked data resources in multi-user mode, unloading workstations, and, provided that the central machine is powerful enough, utilization of cheaper equipment for them. However, this does not allow distributing the parts of a query between hardware cores for their parallel execution.

For example, in MySQL queries are made in a parallel way only if they are from different clients. It means that MySQL cannot parallelize execution of a query on several processing nodes. Therefore, for increase of query efficiency it is necessary to optimize complex queries by means of their decomposing into smaller ones and executing from different clients followed by merge of the results.

For instance, a query for output of user id, email, post id, post message, topic name, topic id, and full name of the user takes 32,77 ms:

```
SELECT u.id_user, u.email, p.id_post, p.message,
t.topic_name, t.id_topic, u.name FROM users u, topics t,
posts p WHERE u.id_user BETWEEN 100 and 900 and
t.id_topic = p.id_topic GROUP BY u.name.
```

After dividing this query into 2 separate ones, the time shortens to 20.6 ms – 16.17 ms faster than the first query:

```
SELECT u1.id_user, u1.email, p.id_post, p.message,
t.topic_name, t.id_topic, u1.name FROM users u1, topics t,
```

```
posts p WHERE u1.id_user BETWEEN 100 and 900 and
t.id_topic = p.id_topic GROUP BY u1.name LIMIT 0,10000;
SELECT u2.id_user, u2.email, p.id_post, p.message,
t.topic_name, t.id_topic, u2.name FROM users u2, topics t,
posts p WHERE u2.id_user BETWEEN 100 and 900 and
t.id_topic = p.id_topic GROUP BY u2.name LIMIT 10001, 20000;
```

Firstly, there are different ways of implementation for the majority of queries with embedded SELECT constructions, and among them there could be ones with faster as well as slower execution speed. Secondly, the example above shows that decomposition of a complex query into several interconnected simple ones allows running the part of selected mutually independent subsequent queries in a parallel way as if they are from different clients. This leads to data processing speed boost.

In PostgreSQL it is easy to parallelize queries with intra-query with horizontal fragmentation parallelism by using a simple and evident trick: creating an index with the function "the remainder on dividing the id by a number of processing nodes".

For example, the table with the data from some transducers has the following format:

```
CREATE TABLE device (id BIGINT, time TIMESTAMP,
device_id INTEGER, indication INTERVAL);
```

The data from devices comes into this table at a high rate, and must be processed in reasonable time by the function:

```
FUNCTION calculate(IN device_id INTEGER, IN indication
INTERVAL, OUT status_code text) RETURNS void;
```

The processing is executed by the query:

```
SELECT calculate(device_id, indication) FROM device;
```

With continuous data reading from the devices, server processing node will soon fail to perform its work properly while the amount of data will be growing.

PostgreSQL, as well as MySQL, cannot parallelize queries on its own. This situation can be handled by creating the following index:

```
CREATE INDEX idx ON device USING btree ((device_id %
4));
```

and executing a query in four threads:

```
SELECT calculate(device_id, indication) FROM device
WHERE device_id % 4 = @rank;
```

where @rank is equal to 0, 1, 2 or 3 (its own for each thread). As a result this action solves problems with locks, which can appear if two different threads get a signal from a single device. Moreover these processes will work faster in the context of parallel execution than in case of parallelizing the database itself, for instance, as in Oracle. This method is applicable for any database with support of function-based index (Oracle, PostgreSQL). In MS SQL one can create calculated column and base index on it. There

is no support of functional indexes in MySQL, but as an alternative it is possible to create a new column with its index and renew it with a trigger.

Therefore, implementation of intra-query parallelism with horizontal fragmentation can be very effective, but it requires high qualification of programmers in two spheres: applied software (good knowledge of functions and capabilities of the used SQL server) and good understanding of the query structure including detecting the ways of query decomposition into a number of subqueries or table decomposition followed by merging the results into the final output buffer. It is possible to simplify work of programmers in improving both parallel performance and quality of work, as well as to reduce the cost of the work, if the function of query optimization is put in a simple and light program wrapper above the SQL server. The wrapper implements equivalent query transformation by mathematical methods in the form that allows parallel execution on a set of processing nodes.

3 Query Decomposition and Visualization with Graphs

Query decomposition can be made in vertically or horizontally, as well as combining these two approaches.

Horizontal query decomposition is a decomposition of relations, to which the query is applied (horizontal fragmentation). The query itself stays unchanged (Fig. 2b). The Fig. 2a shows the original query.

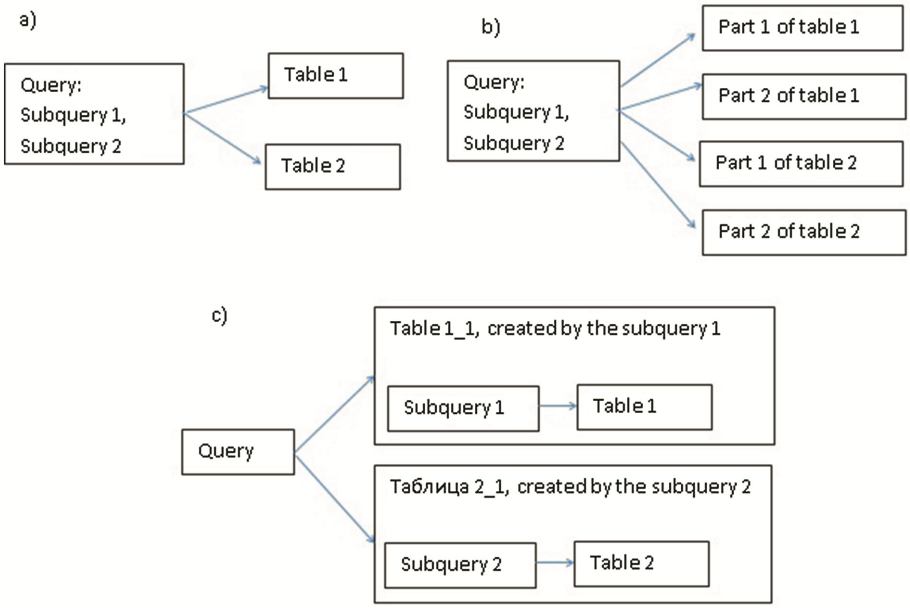

Fig. 2. Queries modification

Vertical request decomposition is a decomposition of the query itself into separate subqueries, through relations, to which the original query was applied, stay unchanged (Fig. 2c). Depending on the structure of the primary query, it is possible to execute either horizontal or vertical decomposition, or, in the best case, their combined variant of simultaneous implementation both vertical and horizontal decomposition (Fig. 3).

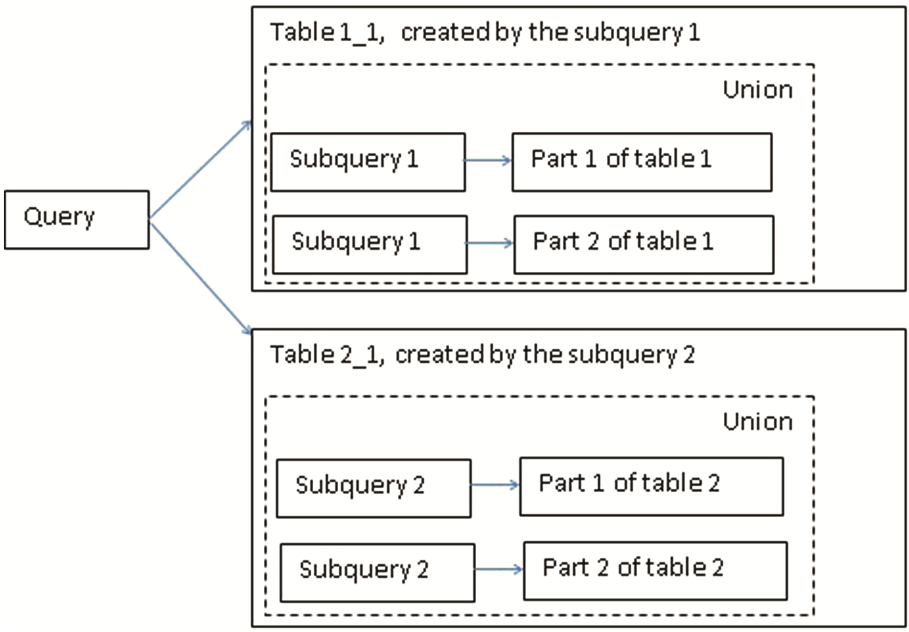

Fig. 3. Combined query modification

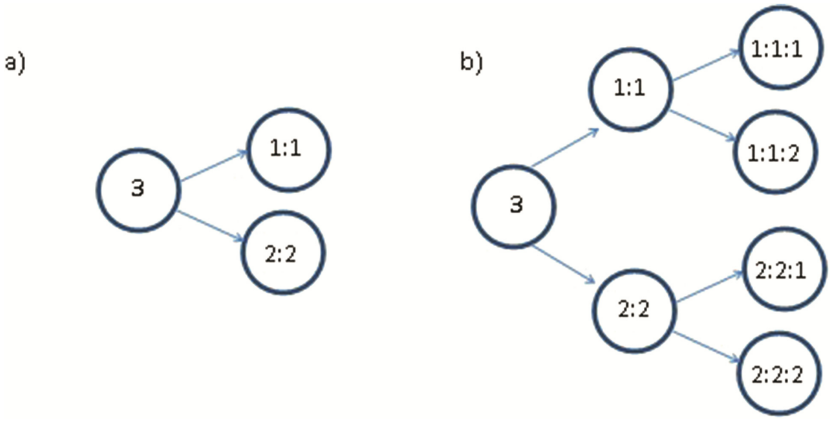

Fig. 4. Visualization of queries with graphs

Graph theory gives more vivid visualization options for presenting interconnections between parts of a query and corresponding relations. On the Fig. 4a one can see a graph that corresponds to the vertical query transformation from the Fig. 2c. The Fig. 4b corresponds to the combined decomposition (Fig. 3). It is possible to decompose the relations even deeper (Fig. 5). Graphs allow, besides query visualization, defining the degree of the internal parallelism by means of matrix methods of optimization of parallel algorithms, evaluating the necessary amount of computing resources, and creating new models of parallel queries.

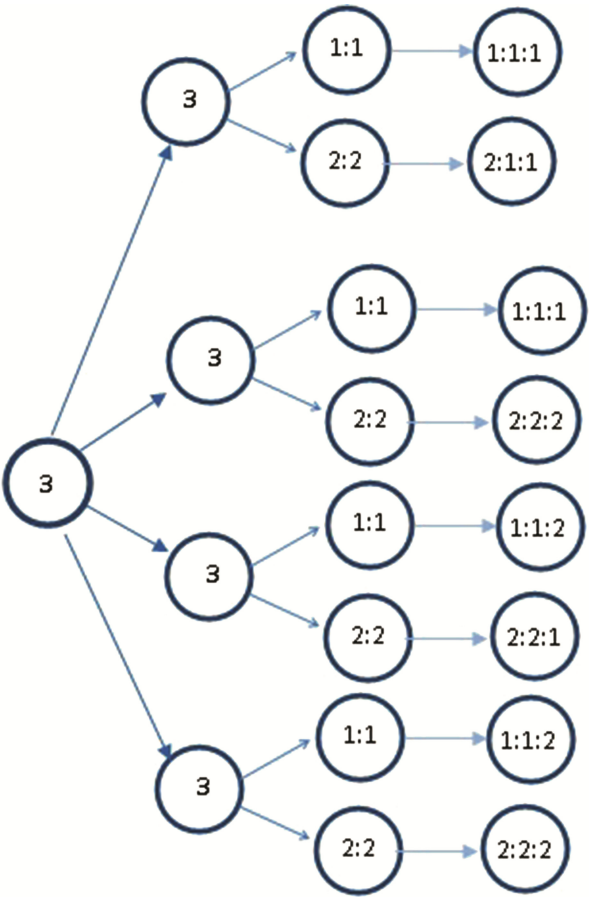

Fig. 5. Representation of a combined query with graphs

The Fig. 6 shows a query, which consists of several subqueries. The same query can be executed in some other ways, for example, by the code shown in the Fig. 7.

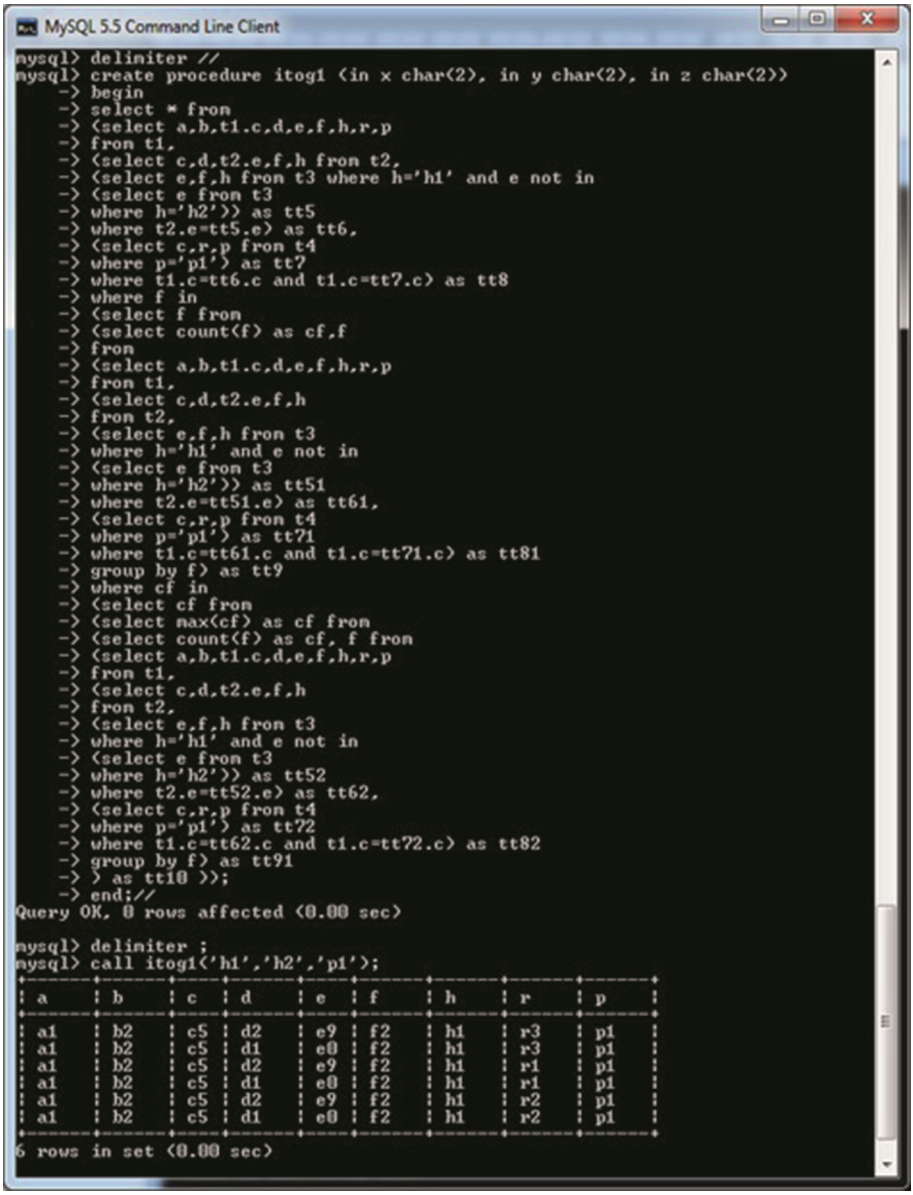

Fig. 6. An example of a "heavy" query

The degree of query parallelism depends on how successfully its structure will be set up. However, it is quite hard to analyze such queries manually. Moreover, even if some faster and more efficient solution will be found, this won't guarantee that the obtained solution will represent the global extremum of all solutions.

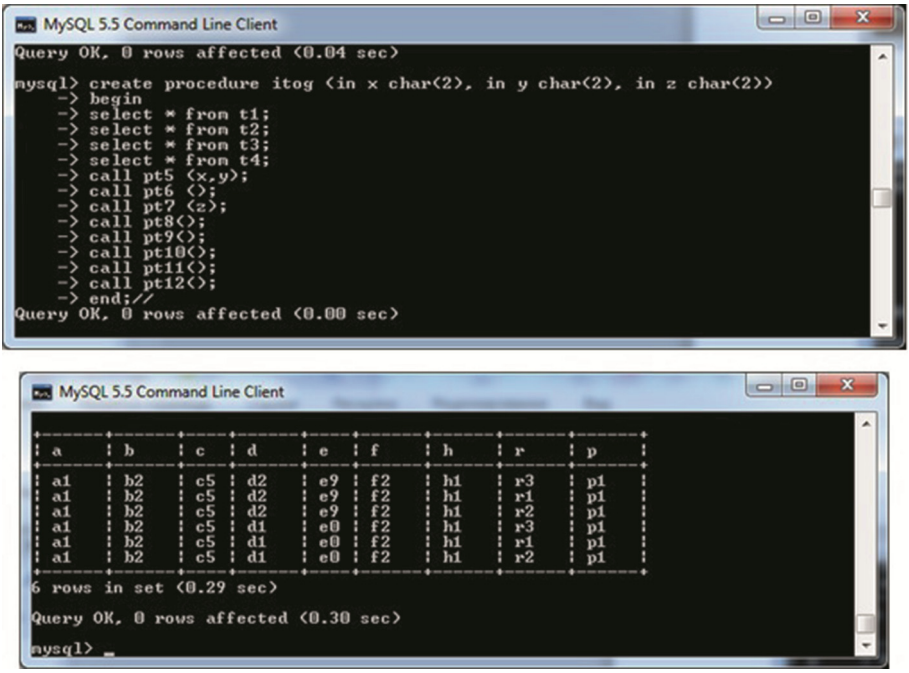

Fig. 7. The modified query

Figure 8 shows query information graph. The searching methods of parallel branches, elaborated for classic algorithms, also could be applied to such graph, for example:

- Methods of finding early and late terms;
- Methods based on scheduling theory;
- Matrix methods based on information graph;
- List-based methods.

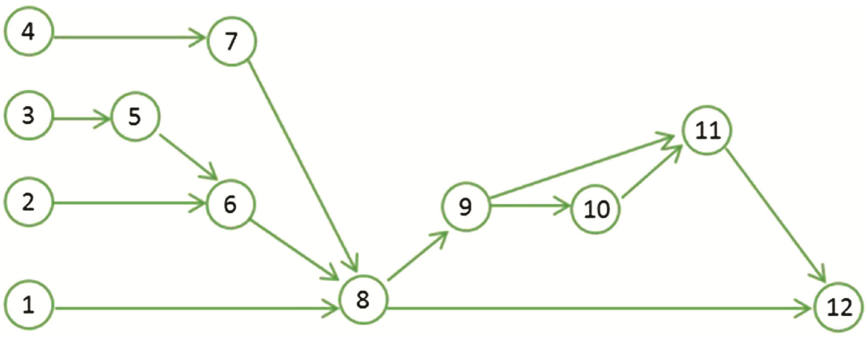

Fig. 8. Query information graph

The optimization methods, elaborated for classic algorithms, could be applied to such graph too:

- By the amount of computing resources;
- By the runtime length;
- Taking the communication into account;
- Methods of multiparametric optimization.

Classic evaluation methods of acceleration, parallelization efficiency, compute density, and others, which are also elaborated for classical algorithms, fit queries represented by information graphs as well.

In conclusion it is important to note that horizontal fragmentation appends additional vertices and edges to an information graph. This also corresponds to one of the stages of classical parallel algorithms design called algorithm decomposition. After obtaining the estimates of acceleration and compute density of the query, it is possible to inverse the process through one of the known methods – fragment upsizing – for those parts of the query, which will allow doing this without any loss of computational efficiency. However only the information graph and mathematical methods of its analysis and equivalent transformation can answer the question, which fragments will it be.

4 Conclusion

The research shows that, if the parts of a complex query are represented as independent client queries, parallelization of complex queries can be achieved by capabilities of standard SQL regardless of a SQL server. Optimization methods and methods of parallel algorithms modeling based on matrix algebra and graph theory can be applied to such queries with success. Moreover, by implementing a special program wrapper it is possible to analyze the structure and to transform any complex user query written in standard SQL.

It should be noted that, for instance, CouchBase DBMS already has capabilities for automated parallel computing, including on multicore processors. But, firstly, CouchBase is a document-oriented DBMS, and it works in NoSQL approach. Conversion of existing large relational databases to CouchBase in order to accelerate query performance is ineffective. Secondly, it has not been fully investigated, how effective is its mechanism for automatic parallelization compared to efficient manual parallelization. The proposed approach is implemented in an interactive form that allows users to improve their work by creating queries based on the options offered by the parallelization program.

Another advantage of the proposed approach is the ability to use it effectively for sparse databases. In this case, the approach can be effectively implemented on the basis of an ideology used in dataspaces. Thus during the fragmentation of a query empty slots in database will fall into separate fragments. The following application of known optimization algorithms for runtime length or data size to the resulting information graph will completely remove the empty fragments from the processing. As a result, the

acceleration will be achieved not only by a query execution on the computing system with parallel processing, but also due to consolidation of data in the structure and elimination of query processing operations on empty fragments.

References

1. Feng, L., Li, Q., Wong, A.: Mining inter-transactional association rules: generalization and empirical evaluation. In: Kambayashi, Y., Winiwarter, W., Arikawa, M. (eds.) DaWaK 2001. LNCS, vol. 2114, pp. 31–40. Springer, Heidelberg (2001)
2. http://www.computerworld.com/article/2513954/data-center/data-growth-remains-it-s-biggest-challenge–gartner-says.html
3. http://www.eweek.com/enterprise-apps/scaleout-introduces-analytics-server-to-crunch-big-data
4. Akal, F., Böhm, K., Schek, H.-J.: OLAP query evaluation in a database cluster: a performance study on intra-query parallelism. In: Manolopoulos, Y., Návrat, P. (eds.) ADBIS 2002. LNCS, vol. 2435, pp. 218–231. Springer, Heidelberg (2002)
5. Sanjay, A., Narasayya, V.R., Yang, B.: Integrating vertical and horizontal partitioning into automated physical database design. In: Proceedings of the ACM SIGMOD International Conference on Management of Data, pp. 359–370, June 2004
6. https://eden.dei.uc.pt/~pnf/publications/Furtado_survey.pdf
7. Chen, Y., Dehne, F., Eavis, T., Rau-Chaplin, A.: Parallel ROLAP data cube construction on shared-nothing multiprocessors. Distrib. Parallel Databases **15**(3), 219–236 (2014)
8. DeWitt, D.J., Gray, J.: Parallel database systems: the future of high performance database systems. Commun. ACM **35**(6), 85–98 (1992)
9. Valduriez, P.: Parallel database systems: open problems and new issues. Distrib. Parallel Databases **1**(2), 137–165 (1993). doi:10.1007/BF01264049