Annales UMCS Informatica AI X, 2 (2010) 41-50
DOI: 10.2478/v10065-010-0051-7

Annales UMCS
Informatica
Lublin-Polonia
Sectio AI
http://www.annales.umcs.lublin.pl/

Speculative execution plan for multiple query execution systems

Anna Sasak*, Marcin Brzuszek

Institute of Computer Science, Maria Curie Sklodowska University,
pl. M. Curie-Sklodowskiej 1, 20-031 Lublin, Poland.

Abstract – There are different levels at which parallelism can be introduced to the database system. Starting from data partitioning (intra-operator parallelism) up to parallelism of operation (inter-operator parallelism) that depends on a query granularity. The paper presents the parallelisation method based on speculative execution for the database systems which are expected to give answers to complex queries coming from different sources as soon as possible. Taking under consideration W of upcoming queries waiting for execution, the execution plan for the first query should be developed. This plan should give the largest benefit also for W-1 of the consecutive queries. Thus, in parallel to the first query, some excessive computations can be executed, which in further steps would reduce the execution time of the consecutive queries. The paper presents possible risks and benefits are using this method and also analyses of possible execution time reduction for different models of speculative parallelization [1].

1 Introduction

In relational databases, the mutual independence between two tables as well as between two tuples within a table, makes simultaneous processing of multiple tables or tuples possible. A database request often involves the processing of multiple tables. The ways in which a request accesses these tables and combines the intermediate results are defined by the computational model. For a single request, there is usually more than one way of execution. Query optimization is the process of determining the best execution path.

The objective of a database optimizer is to produce a good execution plan for a given query. The quality of plan is defined by a cost function which estimates response time or amount of resources used or combination of both. The optimizer searches a set of candidate plans which

*asasak@liza.umcs.lublin.pl

can be generated owing to transformation rules. The set of candidate plans is called the search space.

The cost model provides an abstraction of the parallel execution system in terms of operator cost functions and the database in terms of physical schema information. Database statistics includes distribution of attribute values in a relation. Attribute distribution is used to perform the most fundamental operation in a cost model, that is, computing the selectivity of predicates that appear in a query, which, in turn, helps determine the number of tuples satisfying the predicates [2].

The system must often process upcoming queries in a certain order. Thus the optimizer while creating the execution plan can take under consideration a few of consecutive awaiting queries. Such out-of-order method of analysing and query processing would be called a speculative execution[3]. Speculative action is considered as some work done in anticipation of an event taking place. This offers opportunities of gain or loss that depend on speculation accuracy[4]. In computer architecture, speculative execution is the process of executing instructions ahead of their normal schedule. As the database consists of the finite number of relations, there exists a finite set of queries that can be built over them. If a query references to an n number of relations of all N relations in the database, then for the benefit of expectant database tasks, some queries containing subsets of k referenced relations can be executed speculatively[5]. Then by creating some adequate data copies and transforming the expectant queries so that they use the pre-fetched data, the number of entities to scan would decrease.

Unfortunately, speculations and overheads are inseparable especially when there is a lot of modifying queries in line. As long as the benefits of successful speculative execution outweigh the total overhead of its use, the technique is considered a profitable activity.

The first part of the paper introduces a notation, which is used to estimate the cost of a query execution. The consecutive parts describe the proposed optimization method and conduct a discussion of such system efficiency. The summary outlines the direction of future research.

2 Notations and relational algebra

A parallel optimizer is expected to define an abstract parallel execution model. This model is similar to an algebraic machine that is a set of algorithms for relational operators with more complex rules to combine these algorithms and execute them in parallel. The process of choosing the query execution strategy consists of the series of transformations that replace algebraic expressions with some other equivalent expressions which can be computed more effectively. Most of SQL queries can be defined with a relatively small number of classical relational algebra operators. This paper introduces only a few operators, which in the further parts are used for query execution costs analysis.

The first operator – equivalent of the WHERE clause - is Selection marked as $\sigma_C(R)$. This operator is used to create a new relation based on the existing one, by choosing rows described by a condition or predicate. The arguments of the σ selection are the relation R and the condition C. As a result, a multiset that contains those entities of the R relation that meet the condition C is being created. The mentioned condition can contain:

(1) arithmetical operators (e.g. +, *) or textual operators (e.g. concatenation, LIKE) referencing to constants and/or attributes,
(2) comparisons of those terms (e.g. a<b, a+b=10),
(3) logical operators (AND, OR, NOT).

The next operator – equivalent of the SELECT clause - is Projection. It creates a new relation based on the existing one by choosing a group of columns. If R is a relation then $\Pi L(R)$ stands for a projection of R on the list L. The list L can contain the following elements:

(1) A single attribute of the R relation.
(2) An expression $a \to b$, where a and b are the names of attributes, which means that in the resulting relation the name of an attribute a will be replaced by b.
(3) An expression $E \to z$, where E stands for an expression containing attributes of the relation R, constants, logical operators and textual operators and z stands for the name of the new attribute. Values of this attribute are obtained as results of the expression E e.g. $a + b \to z$ as a list element stands for a sum of a and b arguments marked as z.

The result of projection originates from application of the list L operators to the entities from the relation R. It creates a relation with schema described by the list L and its transformations.

The third operator is Product. That is the Cartesian product from the set theory. Its elements are all possible ordered pairs made of entities of input relations. It is an equivalent of the relations list of the WHERE clause. The product of RxS relations is a new relation built of the attributes of R and S relations. If the particular entity r occurs p times in a R relation and an entity s occurs q times in a S relation, then the rs entity will occur $p * q$ times in the RxS product.

The last introduced operator is Join, which is an equivalent of the JOIN clause. The most common form of Join is a natural join. The natural join of the R and S relations is represented by $R \bowtie S$ and is equivalent to $\Pi L(\sigma C(R \times S))$, where C stands for a condition that compares pairs of chosen attributes of R and S.

The result of natural join can be computed by the consecutive application of three operators: x, σ and Π. Thus the more efficient way is to compute it on the spot. There exists a wide variety of methods of tracking down pairs of matching entities from the R and S relations. Yet for now those methods are not the subject of this paper. Join estimation for the presented analysis assumes that the order of joins has no impact on the general operation cost, and thus is not considered in an execution plan[6]. If the condition C is a single term of $a = b$ form where a and b are the attributes of R and S respectively, then such join is called an equi-join and is represented by $R \bowtie_C S$. On the contrary to natural join an equi-join contains no attribute projection.

Example 2. Assume there is a relation Movie(movie_id, title, year, length, tape, studio, starName) and that the following query was formulated in SQL:

SELECT title, year FROM Movie WHERE length≥100 AND studio='Fox'.

This query expressed with relational algebra operators would have the following form:

$$\Pi_{title,year}(\sigma_{length \geq 100}(\text{Movie}) \cap \sigma_{studio='Fox'}(\text{Movie})),$$

or alternatively

$$\Pi_{title,year}(\sigma_{length \geq 100 \text{ AND } studio='Fox'}(\text{Movie})).$$

3 Cost estimation parameters

When an optimizer is to choose an execution plan, it is essential to introduce some parameters that describe operator cost. Two values T and V are introduced as parameters that measure the access cost for a relation. $T(R)$ stands for an approximate number of the R relation entities. In our analyses the means of storing the particular entities, and thus the possibility of accessing the whole blocks, is abandoned, assuming that each entity must be accessed separately. In such case the $T(R)$ value is also an evaluation of the number of storage accesses, required to read the whole R relation. $V(R, a)$ will stand for the number of distinct values in a column of the R relation. In general, if $[a1, a2, a3, ..., am]$ an attribute list of the R relation, then $V(R, [a1, a2, a3, ..., an])$ stands for the number of distinct values in the columns of the relation R corresponding with the attributes $a1, a2, a3..., am$.

During the execution of the selection, the number of the relation attributes stands still, but the number of entities decreases. In the case of the simplest selection, in which attributes are compared to constants, the size of the result can be easily estimated, if the number of distinct values of analysed attribute is known or possible to estimate. Let $S = \sigma_{a1=C}(R)$ where C is a certain constant. In such case $T(S) = T(R)/V(R, a1)$ can be accepted as an estimation. A regular distribution of the a1 attribute in the R relation is assumed for a simplification. For wider conditions containing inequalities, the value $1/3T(R)$ is assumed as a good estimation [7].

While estimating the join cost it is assumed that the attributes of join $R \bowtie S$ are the primary key in S and a foreign key in R. The estimated cost of join with such assumption is as follows. Let $V(R, a) \leq V(S, a)$. Then for each entity t from R there is a possibility as $1/V(S, a)$, that it can be paired with an entity from S. As S contains $T(S)$ of entities, thus the expected value of the number of entities to be paired with t is $T(S)/V(S, a)$. As there are $T(R)$ entities in R, thus the estimated value of $R \bowtie S$ is $T(R)T(S)/V(S, a)$. If $V(R, a) \geq V(S, a)$, then $T(R \bowtie S) = T(R)T(S)/V(R, a)$. In general $T(R \bowtie S) = T(R) * T(S)/max(V(R, a), V(S, a))$[7].

4 Cost estimation parameters

Let us consider a relational database compound of N tables connected between each other by foreign keys. This database is a cohesive entity in the sense that any group of relations not connected to other relations in this database cannot be distinguished. Answers for database queries are returned by a multiple query execution system. The successive queries are joining the queue to the system. Each of awaiting queries contains references to n $(1 <= n <= N)$

relations from the database. The execution system has at its disposal Nproc processors, one of which is assigned to the first query from the queue (nonspeculative computations) while Nproc-1 processors can be assigned to some additional computations whose purpose is to reduce the system answer time [8].

As a computational step we consider the time in which the first processor returns the data required by a non speculative query, and the rest of Nproc-1 processors execute certain speculative computations. In each computational step, w of W queries awaiting for execution are analysed (sliding-window method)[9]. Based on those analyses, tasks should be assigned to free processors so there would be the highest possible gain from the point of view of the w-1 consecutive queries. Execution of those operations in parallel with the first query would allow for reduction of execution time of the following queries.

At any given moment during the system activity, the graph representing the structure of w analysed queries is available [10, 11]. The construction of this graph must comply with the following assumptions:

(1) Two types of the vertexes are allowed.
   - That representing a certain relation that is a part of a query.
   - That representing the conditions that bound the specific attributes of the relation.

(2) The edges between given vertexes signify the presence of the reference between two relations or the presence of a certain condition with reference to the attribute of the given relation.

(3) Each edge has a weight assigned to it. Weight is interpreted as a probability (frequency) of the presence of connection between two given vertexes.

Schematically the relations will be labelled with the capital letter according to the index: Ri where i stands for the number of each individual relation. Fields of the Ri table will be named fRi,j where i stands for the relation number, to which this field belongs, while j stands for the consecutive number of the field in the range of this relation. A field of the special meaning will be named pk_fRi,0. It will be the primary key for this table. Additionally, a group of special fields might appear in a table. The names of fields in this group will be fk_fRi,j_fRk,0 and they will be a representation of the foreign key to the field being the primary key in the Rk relation. The special case might be the primary key, which consists of the fields that are the foreign keys, with names in the pattern of pk_fk_fRi,j_fRk,0.

5 The efficiency analysis

Let us consider the database containing seven tables. Fig. 1. presents the relations schema for the following database structure:

R1(pk_fR1,0; fR1,1;...;fR1,7); **R2**(pk_fR2,0; fR2,1;...;fR2,15; fk_fR2,16_fR1,0);
R3(pk_fR3,0; fR3,1;...;fR3,4, fk_fR3,5_fR4,0); **R4**(pk_fR4,0; fR4,1;...;fR4,6); **R5**(pk_fR5,0; fR5,1;...;fR5,10; fk_fR5,11_fR1,0; fk_fR5,12_fR3,0); **R6**(pk_fR6,0; fR6,1;...;fR6,3);
R7(pk_fk_fR7,0_fR6,0;pk_fk_fR7,1_fR5,0);

The following queries wait for execution:

K1: select * from **R6** where fR6,3>C1
K2: select * from **R3** join **R4** on R4.pk_fR4,0=R3.fk_fR3,5_fR4,0 where R4.fR4,6=C2
K3: update **R4** set fR4,4=C3 where fR4,1>C4
K4: select * from **R3** join **R4** on R4.pk_fR4,0=R3.fk_fR3,5_fR4,0 where R4.fR4,6<C5 (it is stated that C5>C2)
K5: select * from **R1** join **R2** on R1.pk_fR1,0=R2.pk_fR2,0 join **R5** on R5.fk_fR5,11_fR1,0=R1.pk_fR1,0 where R2.fR2,1>C6
K6: select * from **R1** join **R5** on R5.fk_fR5,11_fR1,0=R1.pk_fR1,0 where R5.fR_5,1<C7

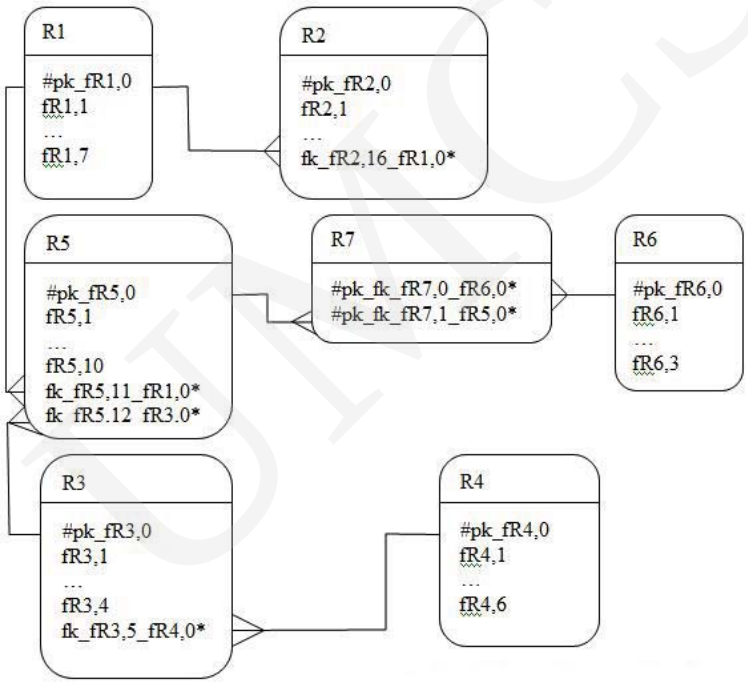

Fig. 1. Relation schema for the database.

The execution analysis is conducted for the following parameters that describe the system: $NProc = 3$ stands for the number of available processors, $w = 4$ stands for the window size and thus the number of queries analysed in one computational step, $k = 1, 2, ..., NProc$ is the number of the processor in the group, CompStep is the number of the consecutive computational step. Fig. 2 presents the graphs for the consecutive computational steps describing the relations and the conditions that occur in queries covered by the window.

In the first step, in fact, there are two separate graphs available, as the relations set of the first query has no common point with that of the consecutive three queries. The numbers assigned to the edges are the weights. In the first step set to 1 except the edge between the $R3$ and $R4$ vertexes as this connetion occurs in two of queries in window. With regard to the graphs

separation, the first processor is assigned to the whole first – nonspeculative query ($K1$). In the other case, it would be worth considering splitting the first query into properly sized seeds just now. There are still two free processors remaining which can be assigned to speculative computations. The first conclusion is that three different conditions are referencing the $R4$ query, whereas the two of them relate to the same attribute and moreover, the $C5$ inequality is wider, and contains the constant comparison. It is obvious that one of the processors must be assigned to the $R4$ selection referencing the wider condition ($fR\_4, 6 < C5$), as it reduces the number of entities for the two of the awaiting queries – $K2$ and $K4$. There still remains one available processor and one special unassigned edge. This particular edge is distinguished with dotted line as it refers to the update query. Modifying queries need special attention as they influence the results of other awaiting queries and thus can cause the necessity to cancel the speculative results. Then the last processor is assigned to the $R4$ update. (In fact, speculative processors executing modifying queries create only a copy of data with modifications as the order of execution must be kept.) When the assigned tasks are finished, and the partial results are saved, the awaiting queries must be modified so that they take those changes into account, and then the window should be moved and the graph refreshed. As a result, the $K2$ and $K4$ queries would take the following form, and the window would include the $K5$ query.

K2: select * from **R3** join **R4_spec1** on R4_spec1.pk_fR4,0=R3.fk_fR3,5_fR4,0 where fR4,6=C2
K3: already executed, changes need confirmation
K4: select * from **R3** join **R4_spec1** on **R4_spec1**.pk_fR4,0=R3.fk_fR3,5_fR4,0
K5: select * from **R1** join **R2** on R1.pk_fR1,0=R2.pk_fR2,0 join **R5** on R5.fk_fR5,11_fR1,0=R1.pk_fR1,0 where R2.fR2,1>C6

In the second computational step the $K2$ query is computed nonspeculatively. Next in line is the $K3$ update query already executed speculatively so the remaining processors can be used for the benefit of the $K4$ and $K5$ queries. At least three separate options are available for computations assignment. First of them is finishing the $K4$ query (risky due to awaiting update query). Second, selection from the $R2$ relation and third, the $R1$ and $R5$ join. Excluding the risky option there remain two tasks and two available processors so the assignment is obvious. After results saving and queries modifications, the situation for $CompStep = 3$ would be as follows:

K3: already executed, changes need confirmation
K4: select * from **R3** join **R4_spec1** on **R4_spec1**.pk_fR4,0=R3.fk_fR3,5_fR4,0
K5: select * from **R2** join **R1_R5_spec** on R1_R5_spec.pk_fR1,0=R2.pk_fR2,0 where R2.fR2,1>C6
K6: select * from **R1_R5_spec** where R1_R5_spec.fR_5,1<C7

The first query waiting for the execution in the third computational step is an already executed update query. Thus, while confirming the results of $K3$ and before speculative assignments can be done, any subsequent speculations referencing the same relations as $K3$ must be cancelled. As $K3$ modified at least one row of the $R4$ relation the $K4$ query has to return to its previous form, that is not using the speculative results. When this process is over the remaining queries are as follows:

K4: select * from **R3** join **R4** on **R4**.pk_fR4,0=R3.fk_fR3,5_fR4,0 where fR4,6<C5
K5: select * from **R2** join **R1_R5_spec** on R1_R5_spec.pk_fR1,0=R2.pk_fR2,0 where R2.fR2,1>C6
K6: select * from **R1_R5_spec** where R1_R5_spec.fR_5,1<C7

Two remaining processors can be assigned to a few different operations but selections are always preferred as there is always reduction of tuples expected. Thus the remaining processors would compute the $R4$ selection and the $R2$ selection respectively. After results saving and queries modifications the situation for the last $CompStep = 4$ would be as follows:

K4: select * from **R3** join **R4_spec** on **R4_spec**.pk_fR4,0=R3.fk_fR3,5_fR4,0
K5: select * from **R2_spec** join **R1_R5_spec** on R1_R5_spec.pk_fR1,0=R2.pk_fR2,0
K6: select * from **R1_R5_spec** where R1_R5_spec.fR_5,1<C7

As the forth computational step is the last one in our example the last processors assignment will be simple since there are three queries waiting and three available processors, and thus each query will be computed by one processor.

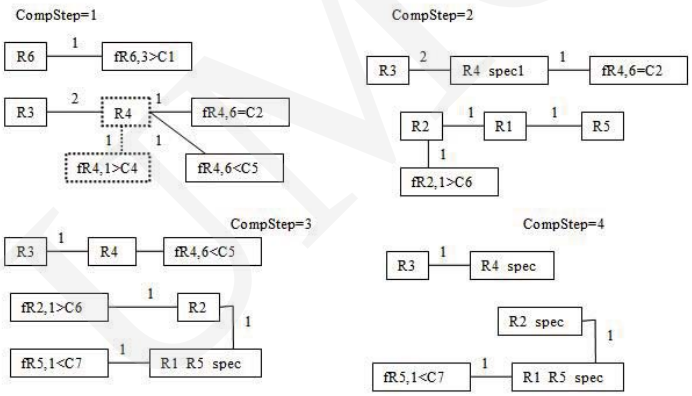

Fig. 2. The relations graphs for the consecutive computional steps.

After analysing the course of execution let us analyse its cost. In cost analyses only "SELECT" queries are taken under consideration and thus the $K3$ query is ignored. For the sequential execution – by sequential execution it is assumed executing each query separately, even in parallel but without any benefits of partial results – the costs estimation presents as follows:

$$T(seq) = T(K1) + T(K2) + T(K4) + T(K5) + T(K6) = 1/3T(R6)$$
$$+T(R4)/V(Ri, fRi, j) + T(R3 \bowtie R4) + T(R3 \bowtie R4)$$
$$+1/3T(R4) + T(R1 \bowtie R2 \bowtie R5) + 1/3T(R2) + T(R1 \bowtie R5) + 1/3T(R5).$$

The estimation for the speculative execution would be:

$$T(spec) = T(nCompStep = 1) + T(nCompStep = 2)$$
$$+T(nCompStep = 3) + T(nCompStep = 4).$$

Unfortunately, due to the update query and missed speculation, the comparison between $T(seq)$ and $T(spec)$ does not give a clear profit and loss account. Thus it is better to compare only the chosen fragments of computations which are a good example of adopted method performance. Let us start with the cost estimation of the sequential execution of two queries $K5$ and $K6$:

$$T(K5\_K6\_seq) = T(K5) + T(K6)$$
$$= T(R1 \bowtie R2 \bowtie R5) + 1/3T(R2) + T(R1 \bowtie R5) + 1/3T(R5).$$

The estimation for the speculative execution:

$$T(K5\_K6\_spec) = 1/3T(R2) + T(R1 \bowtie R5)$$
$$+T(R2\_spec \bowtie R1\_R5\_spec) + 1/3(R1\_R5\_spec)$$
$$T(K5\_K6\_sek) - T(K5\_K6\_spec)$$
$$= T(R1 \bowtie R2 \bowtie R5) + 1/3T(R2) + T(R1 \bowtie R5)$$
$$+1/3T(R5) - 1/3T(R2) - T(R1 \bowtie R5)$$
$$-T(R2\_spec \bowtie R1\_R5\_spec) - 1/3(R1\_R5\_spec)$$
$$= T(R1 \bowtie R2 \bowtie R5) + 1/3T(R5)$$
$$-T(R2\_spec \bowtie R1\_R5\_spec) - 1/3(R1\_R5\_spec)$$
$$= T(R1 \bowtie R2 \bowtie R5) + 1/3T(R5)$$
$$-1/3T(R1 \bowtie R2 \bowtie R5) - 1/3(R1 \bowtie R5)$$
$$= 2/3T(R1 \bowtie R2 \bowtie R5) + 1/3T(R5) - 1/3(R1 \bowtie R5)$$
$$\approx 2/3T(R1 \bowtie R2 \bowtie R5)[asT(R5) \geq T(R1 \bowtie R5)] > 0$$

The second example refers to the $K2$ and $K4$ queries. Executed sequentially they require double $R3$, $R4$ join and double $R4$ scan due to two different conditions. Executing those queries speculatively we execute one $R4$ scan excessively as due to the update query speculations for $K4$ must be cancelled.

We have proved that speculative execution of sql queries gives profit in amount of reference to the number of relation entities, and by that also shortens the amount of time needed to get answers. Amount of profit will depend on the speculation accuracy and thus on the number of modifying queries in line. Additional factors influencing profits are the size of the data in given relations and distribution of given attribute values.

6 Summary

The presented SQL query speculative computation scheme seems to be a promising method for multiple query execution systems. Due to the usage of the inter-operator parallelism, it is possible to choose such granularity and execution plan which will generate as much profit as possible not only for the first awaiting query, but also for a group of the consecutive awaiting queries. Open for the discussion there remain the problems associated with modifying queries appearing on the list, which have substantial influence on accuracy of the speculative computations. Nonetheless the analysis presented in this paper proves that this method has a significant chance of giving promising results.

References

[1] Gonzales-Escribano A., Llanos Diego R., How things work: speculative parallelization, IEEE Press Computer 39(12) (2006): 126–128.

[2] Abdelguerfi M., Wong K., Parallel satabase techniques (Wiley – IEEE Computer Society, California, 1998): 28–34.

[3] Barish G., Knoblock C. A., Speculative plan execution for information gathering, Journal Artificial Intelligence 172(4-5) (2008): 413–453.

[4] Kaeli D., Yew P., Speculative execution in high performance computer architectures (CRS Press Taylor & Francis Group, USA, 2005): 224–229.

[5] Polyzotis N., Ioannidis Y., Speculative query processing, Online Proc. of the CIDR Conference (2003).

[6] Galindo-Legaria C. A., Rosenthal A., Outerjoin simplification and reordering for query optimization, ACM Transactions on Database Systems 22(1) (1997): 43–74.

[7] Garcia-Molina H., Ullman J. D., Widom J., Database system implementation (WNT, Warszawa, 2003): 275–281.

[8] Sasak-Okoń A., Brzuszek M., Speculative parallelization for multiple query execution systems, Polish J. of Environ. Stud. 18(3B) (2009): 321–326.

[9] Cintra M., Llanos D. R., Design space exploration of a software speculative parallelization scheme, IEEE Transactions on Parallel and Distributed Systems 16(6) (2005): 562–576.

[10] Galindo-Legaria C. A., Rosenthal A., Query graphs, implementing trees, and freely-reorderable outerjoins, Proc. of the 1990 ACM SIGMOD International Conference on Management of Data (1990): 291–299.

[11] Mahajan S., Jadhav V. P., A survey of issues of query optimization in parallel databases, International Journal of Computer Applications 11(3)(2010): 32–37.