

Query Optimization in Database Grid

Xiaoqing Zheng, Huajun Chen, Zhaohui Wu, and Yuxin Mao

Grid Computing Lab, College of Computer Science, Zhejiang University,
Hangzhou, 310027, China
{zxqingcn, huajunsir, wzh, maoyx}@zju.edu.cn

Abstract. DarGrid II is an implemented database grid system whose goal is to provide a semantic solution for integrating database resources on the web. Although many algorithms have been proposed for optimizing query-processing in order to minimize costs and/or response time, associated with obtaining the answer to query in a distributed database system, database grid query optimization problem is fundamentally different from distributed query optimization. These differences are shown to be the consequences of autonomy and heterogeneity of databases in database grid. Therefore, more challenges have arisen for query optimization in database grid than traditional distributed database. Following this observation, we present the design of a query optimizer in DartGrid II, and a heuristic, dynamic, and parallel query optimization approach for processing query in database grid is proposed.

1 Introduction

In presence of web, one critical challenge is how to globally publish, seamlessly integrate and transparently locate geographically distributed database resources with such "open" settings. DartGrid II proposes a semantic-based approach supporting the global sharing of database resources using grid as platform and dynamically integrates information from autonomous local databases managed by heterogeneous database management systems in the web environment.

We use ontologies to define conceptual model or standard terminology and relations between them in certain domain. Databases are semantically registered to the web service called Semantic Registry Service by mapping from relation attributes to the properties of ontology (standard terminology in given domain). End-user browses the ontology to generate a visual conceptual query by Semantic Browser developed for DartGrid II, and then Semantic Query Service translates a semantically enriched query into a distributed query plan by mapping from shared ontology to local database schemas. Query results will be returned to user as semantically wrapped format and presented in Semantic Brower.

For a query involving more than one database, global query optimization should be performed to achieve good overall system performance. Because there are some fundamental differences between traditional distributed Database Management System (DBMS) and Database Grid System (DBGS) which stem from autonomy and heterogeneity of database nodes participating in DBGS, query optimization techniques in distributed DBMS can not nontrivially and directly be applied to DBGS. Site autonomy in DBGS refers to the situation whereby each database node retains

complete control over local data and processing. This has a number of implications for query optimization in DBGS. [Veijalainen and Popescu-Zeletin, 1988] classify site autonomy into three types: design, communication, and execution.

Design autonomy implies that database nodes are responsible for optimizing local access paths and query processing methods. Consequently, reliable statistical information which is needed for effective global query optimization is not readily available and may not remain accurate as database nodes change over time. Communication autonomy in DBGS means that a database node independently determines what information it will share with the global system, when it participates in the database grid, and also when it will stop participating. This adds to the complexity of query processing and optimization since any database node system may terminate its services without any advance notice. Execution autonomy results in the situation whereby the global system interfaces with database nodes at their external user interfaces, and hence is not able to influence how query processing is being carried out in the database nodes. This means that there is no opportunity for low-level cooperation across systems and hence primitive query processing techniques proposed for distributed database system may no longer be applicable. For example, the semijoin and pipeline operation may hard to implement in efficient way for lack of facilities provided by low-level and underlying system environment.

In query optimization for distributed DBMS it is assumed that component sites are equal in terms of their processing capability. This assumption no longer seems reasonable in the context of DBGS since database nodes may vary drastically in terms of their availability and processing costs. Furthermore, the same real world object may be represented in more than one database node, but these representatives are not always structurally compatible in DBGS. By the way, database grid system is also different from multidatabase system since DBGS supports and allows database nodes dynamically participate in or quit from the system and sets target for facing more "open" settings of web environment. However, the query processing problem is much more difficult in database grid environment than in centralized, distributed, and multi database. But it is very important for the success of system. We present the design of a query optimizer in DartGrid II, and a heuristic, dynamic, and parallel optimization approach for processing query in database grid. In the following discussion, the global data modal is assumed to be relational for convenient discussion.

The remainder of this paper is organized as follows. Section 2 presents a brief overview of query optimization in traditional distributed database. In Section 3, the architecture of a database grid query optimizer is proposed. Section 4 describes the algorithms for query optimization in database grid, and some experimental results have been discussed. The conclusions and future work are summarized in Section 5.

2 Related Work

Because it is a critical performance issue, query processing has received (and is still receiving) considerable attention in both centralized and distributed DBMS. A large number of different algorithms have already been developed for query optimization in database systems. The numerous algorithms employed in various applications have already been proposed for query optimization which can roughly be divided into three

categories or are combination of such basic algorithms: exhaustive search, heuristics and randomized algorithms [Kossmann and Strocker, 2000].

Typical exhaustive search algorithm is dynamic programming [Selinger et al., 1979], and [Ono and Lohman, 1990] is its improvement. All proposed algorithms of this class have exponential time and space complexity and are sure to find the best plan according to the specific cost model. Other transformation-based techniques with top-down dynamic programming are EXODUS [Graefe and DeWitt, 1987] and Volcano [Graefe and McKenna, 1993]. [Kossmann and Strocker, 2000] presents a new class of query optimization algorithms that are based Iterative Dynamic Programming (IDP) and declares that IDP algorithm can produce the best plan of all known algorithms in the situation in which dynamic programming is not viable because of its high complexity.

Heuristics algorithms have polynomial time and space complexity, but they produce plans are often more expensive than those of exhaustive search algorithms. The most outstanding representatives of this class based on "minimum selectivity" and "greedy principle" are in [Palermo, 1974], [Swami, 1989], [Shekita et al., 1993], and [Steinbrunn et al., 1997].

To avoid the high cost of exhaustive search, randomized strategies, such as Simulated Annealing [Ioannidis and Wong, 1987] and Genetic Algorithms [Jiunn-Chin et al., 1996] have been proposed. They try to find a good solution, not necessarily the best one, but avoid the high cost of optimization. The best known randomized algorithm is called 2PO which is a combination of iterative improvement and simulated annealing [Ioannidis and Kang 1990]. Other representative of this class are in [Lanzelotte et al., 1993] and [Galindo-Legaria et al., 1994].

Other important approaches of query optimization should be given appropriate attention. SDD-1 [Bernstein et al., 1981] is derived from an earlier method called the "hill-climbing" algorithm, which has the distinction of being the first distributed query processing algorithm. The main problem of SDD-1 is that the algorithm may get stuck at a local minimum cost solution and fail to reach the global minimum. R* [Selinger and Adiba, 1980] is a substantial extension of the techniques developed for system R's optimizer. Therefore, it uses a compilation approach where an exhaustive search of all alternative strategies is performed in order to choose the one with the least cost. The Parallel Nested Loop algorithm [Bitton et al., 1983] is the simplest one and the most general. The algorithm of Parallel Hash Join for specific multiprocessor architecture is given in [Valduriez and Gardarin, 1984].

3 Design of Database Grid Query Optimizer

The role of a query processor in DartGrid II is to map a high-level semantic query on ontologies into a sequence of database operations on relevant database nodes. The semantic mapping between ontologies and relational database is shown in Figure 1. We refer to database grid query optimization as generation a query execution plan for a given query defined over the collection of database nodes. The goal of a database grid optimizer (GOQ) may be summarized as follows: given a semantic query on a database grid, find a corresponding execution strategy that minimizes a system cost function that includes I/O, CPU, and communication costs. After translating an

ontology-based semantic query to global relational calculus, an execution strategy is specified in terms of relational operations and communication primitives (send/receive) applied to the database nodes. Therefore, the complexity of relational operations that affect the performance of query execution is of major importance in the design of a query optimizer. Figure 2 shows the architecture of a database grid query optimizer.

When GOQ receives a semantic query from end-user, Semantic Parser checks the syntax and semantics of the query using the schema information provided by Global Catalogue, and then, rewrites from the original query to the equivalent global relational query (see [Zhaohui Wu et al., 2004] for more details). Query Decomposer module eliminates redundant predicates in the query and decomposes it into simple relation execution at local database nodes. Some simple heuristic rules, such as applying unary operations (select/project) as soon as possible, can be used in this phase to improve performance. Subsequently, Plan Generator chooses the best point in the solution space of all possible execution strategies using database statistics and run-time system parameters (system and network workload) in terms of the cost model that typically refers to weighted combination of I/O, CPU and

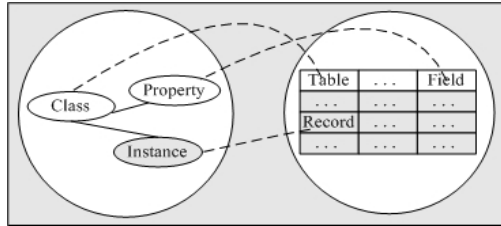


Fig. 1. The semantic mapping between ontologies and relational database

When GOQ receives a semantic query from end-user, Semantic Parser checks the syntax and semantics of the query using the schema information provided by Global Catalogue, and then, rewrites from the original query to the equivalent global relational query (see [Zhaohui Wu et al., 2004] for more details). Query Decomposer module eliminates redundant predicates in the query and decomposes it into simple relation execution at local database nodes. Some simple heuristic rules, such as applying unary operations (select/project) as soon as possible, can be used in this phase to improve performance. Subsequently, Plan Generator chooses the best point in the solution space of all possible execution strategies using database statistics and run-time system parameters (system and network workload) in terms of the cost model that typically refers to weighted combination of I/O, CPU and

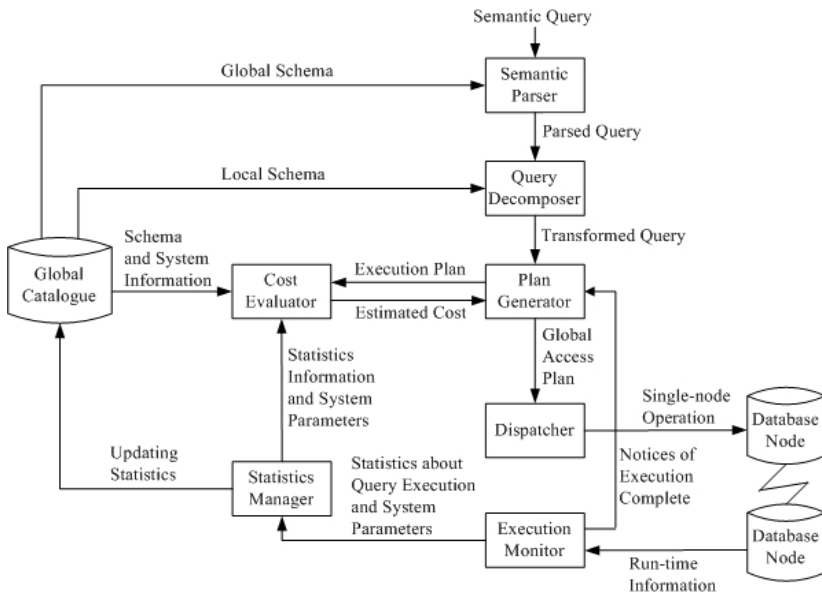


Fig. 2. Architecture of database grid query optimizer

communication costs. An immediate method for producing the sequence of operation is to search the solution space, exhaustively predict the cost of each strategy, and select the strategy with minimum cost. Although this method is effective in selecting the best strategy, it may incur a significant processing cost for the optimization itself, since the solution space can be very large.

The most powerful search strategy used by query optimizers is dynamic programming, which start from base relations, joining one more relation at each step until complete plans are obtained. Dynamic programming is almost exhaustive and assures that the "best" of all plans is found. It incurs an acceptable optimization cost when the number of relation in the query is small. However, this approach becomes too expensive when the number of relations is greater than 5 or 6. The worse is that the dynamic programming dramatically dependent on accurate statistics information. But in the environment of the database grid with numerous database nodes that will dynamically participate or quit the grid system, it is impossible and highly expensive to maintaining accurate and complete database statistics or errors in these estimates can lead to bed performance, therefore, dynamic programming and its variances are not suitable to database grid situation.

We choose another popular way of heuristics to reduce the cost of exhaustive search, whose effect is to restrict the solution space so that only a few strategies are considered. A common heuristic is to minimize the size of intermediate relations and we use this search strategy in dynamic way. Firstly, Plan generator sends Dispatcher to perform unary operations at each relevant database node. Execution Monitor will collect results of operations executed and run-time system parameters, send this information to Statistics Manager for updating Global Catalogue to reflect the effect of change, and synchronously notify Plan Generator for the end of execution. Then, Plan Generator determines the next join operation with the most selective which will produce the minimum size of intermediate results by assistant of Cost Evaluator. System runs step by step as above way and execute join operation in parallelism as can as possible to minimize response time (more discussion in Section 4). At any point of execution, the choice of the best next operation can be based on accurate knowledge of the results of the operations executed previously. Therefore, database statistics are not needed to estimate the size of intermediate results. However, little statistic information may still need and will update periodically by issuing queries to database nodes with proper sampling technique during off-peak hours. The main advantage of this strategy over static query optimization is that the actual sizes of intermediate relations are available to the query processor, thereby minimizing the probability of a bad choice and it is very attractive and suitable for database grid.

4 Query Optimization Approach

The goal of query optimization is to find an execution strategy for query which is close to optimal. But remember that finding the optimal solution is computationally intractable. We propose a heuristic, dynamic, and parallel query optimization to meet this computation complexity, heuristics for reducing solution space, dynamic for generating better execution sequences, and parallelism for minimizing response time.

4.1 Cost Model

An optimizer's cost model includes cost functions to evaluate the cost of operators, statistics and formulas to predict the sizes of intermediate results. Two common objectives are minimum total cost and minimum response time. Total cost is the sum of all times incurred in processing the operations of query at various database nodes and in transferring intermediate results among participating database nodes. Response time of query is the time elapsed for executing query. We adopt the objective of minimum total cost in our query optimization with making the best use of parallelism if possible to reduce response time.

A general formula for determining the total time of transferring x tuples from node i to node j and processing x tuples with y tuples (which are already at node j) at node j can be specified as follows:

$$Total_time_{ij}(x, y) = Init + Ship_{ij}(x) + Process_j(x, y) \quad (1)$$

Where *Init* is the cost for generating nest query operation, startup of transmission, and control of security; The $Ship_{ij}$ is the cost function of transferring x tuples from node i to node j , which depends on channel bandwidth, error rate, distance, and other line characteristics; The $Process_j$ is the cost function of local processing by the database nodes j (typically refers to times of disk I/Os and CPU instruction).

Remember that above formula does not consider some factors like network and other dynamic characteristics of database nodes. In grid environment, we can not ignore these factors and should modify above formula to reflect the effect of dynamic changes about system and network. We use SL_j to represent the factor of workload at node j (20% for example), and let the result of $Process_j(x, y)$ divided by $(1 - SL_j)$ ($Process_j(x, y) / (1 - SL_j)$) to evaluate the cost of local processing at node j instead of original $Process_j(x, y)$. Adjustment in communication cost is similar to above discussion. Consider that NL_{ij} represent the factor of network load from node i to node j , and we substitute $(Ship_{ij}(x) / (1 - NL_{ij}))$ for above $Ship_{ij}(x)$.

Join selectivity factors for some pairs of relations are required in order to predict the size of intermediate result which is the proportion of tuples participating in the join. The join selectivity factor, denoted SF_j , of relation R and S is a real value between 0 and 1:

$$SF_j(R, S) = card(R \bowtie S) / card(R) * card(S) \quad (2)$$

$card(R)$ represents the number of tuples of the relation R , and \bowtie denotes the natural join. We say that a join has better selectivity if it has a smaller join selectivity factor. When the join selectivity factor between relation R and S is not available in Global Catalogue during the run-time, we use upper bound ($card(R \times S) = card(R) * card(S)$) divided by a constant to reflect the fact that the join result is smaller than that of the Cartesian product. The sampling method should be used to obtain information about the system, such as, startup time, transmission rate, join selectivity factor and process overhead. Such statistic information is stored in Global Catalogue and maintained by Statistics Manger. Database grid query optimizer retrieves that information when a cost needs to be evaluated for a global access plan.

4.2 Algorithm Description

As mentioned in the introduction section, the same real world object may be represented in more than one node in a database grid system (see Figure 3). The top of the Figure 3 is a semantic view of ontologies and the bottom is four different database nodes. The attributes of relation register to ontology properties according to their semantic meaning. Node 1 and 2 have medicine as well as therapy relation, whereas node 3 only possesses medicine relation, and node 4 only has therapy relation. Medicine relation keeps track of medicine names and their ingredients. Diseases and their available medicine are stored in therapy relation. In distributed database, if a relation R is horizontally decompose into fragments R_1, R_2, \dots, R_n , and data item d_i is in R_j , it is not in any other fragment R_k ($k \neq j$). This criterion ensures that the horizontal fragments are disjoint, but this rule is not followed when refer to database grid. Therefore, the union of two relations (we call it subrelation) which materialize same relational schema at different nodes is not always equal to empty. Let us illustrate the idea of our algorithm using the following query:

Example 1. "Retrieve all diseases and their available medicine as well as its ingredient". The following are two feasible access plans to execute it:

Access Plan 1: Send all relations to node 1; perform $\cup_{(nodes = 1,2,3)} R_{medicine}$ and $\cup_{(nodes = 1,2,4)} R_{therapy}$ there (\cup denotes union operation); and execute $R_{medicine} \bowtie R_{therapy}$ at node 1.

Access Plan 2: Send $R_{therapy}$ ($node = 1$ and 2 and 4) to each node of 1, 2, and 3; At each node i ($i = 1, 2,$ and 3), execute $\cup_{(nodes = 1,2,4)} R_{therapy}$ and $R_{medicine}(node = i) \bowtie R_{therapy}$ respectively. Send the results of former execution at node 2, 3 to node 1; perform union of all results at node 1.

However, above two access plan have an expensive cost obviously. But this scenario often happens in database grid, we propose *DG-PHJ* algorithm (see Algorithm 1) by extending the parallel hash join algorithm [Özsu and Valduriez, 1991] to meet this difficulty. The basic idea of *DG-PHJ* algorithm is to partition

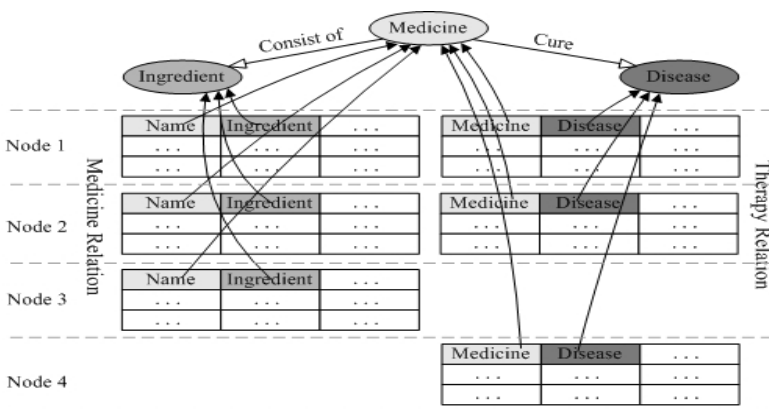


Fig. 3. Materialization of scheme relation schema at different nodes in database grid

relation R and S into the same number p of mutually exclusion sets R_1, R_2, \dots, R_p , and S_1, S_2, \dots, S_p by employing hash function on the join attribute, such that

$$R \bowtie S = \cup_{(i=1 \text{ to } p)} (R_i \bowtie S_i) \quad (3)$$

Algorithm 1. DG-PHJ

input: R_1, R_2, \dots, R_m : subrelations of relation R ;
 S_1, S_2, \dots, S_n : subrelations of relation S ;
 JP : join predicate.
output: T_1, T_2, \dots, T_p : result subrelations.
begin: $\{JP \text{ is } R.A = R.B \text{ and } h \text{ is a hash function}$
that returns an integer number in $[1, p]\}$
for $i = 1$ **to** m **do in parallel** {hash R on the join attribute}
begin
 $R_{ij} \leftarrow$ apply $h(A)$ to R_i ($j = 1, \dots, p$)
for $j = 1$ **to** p **do**
send R_{ij} to node j
end-for
end-for
for $i = 1$ **to** n **do in parallel** {hash S on the join attribute}
begin
 $S_{ij} \leftarrow$ apply $h(B)$ to S_i ($j = 1, \dots, p$)
for $j = 1$ **to** p **do**
send S_{ij} to node j
end-for
end-for
for $j = 1$ **to** p **do in parallel** {perform the join at each p -node}
begin
 $R_j \leftarrow \cup_{(i=1 \text{ to } m)} R_{ij}$ {receive from R -nodes}
 $S_j \leftarrow \cup_{(i=1 \text{ to } n)} S_{ij}$ {receive from S -nodes}
 $T_j \leftarrow \text{JOIN}(R_j, S_j, JP)$
end-for
end. {DG-PHJ}

the same hash function is applied to join attribute to partition R and S . Each individual join ($R_i \bowtie S_i$) is executed in parallel, and the join results are produced at p nodes. These p nodes actually are selected at run time based on the load of the system and network. Since operations can be executed in parallel at different nodes, the response time of query will be significantly less than its total cost. Figure 4 shows the application of *DG-PHJ* algorithm with Example 1. We assumed that the results are produced at nodes 1, 2, and 3. An arrow in the Figure 4 indicates a data transfer. However, *DG-PHJ* algorithm only can be applied when join predicate is equijoin that happens most frequently. The execution strategy of access plan 2 in Example 1 can be used in the situation of arbitrarily complex join, and we call this *DG-PNL* algorithm which follows the idea of parallel nested loop algorithm [Özsu and Valduriez, 1991].

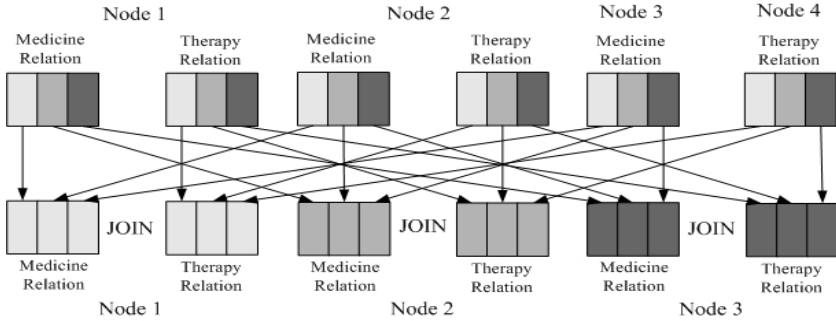


Fig. 4. Example of *DG-PHJ* Algorithm

After discussion of the cost model, *DG-PHJ* and *DG-PNL* algorithm, we now can propose the global algorithm for query optimization in database grid. As shown in Algorithm 2. The algorithm of *DG-QOA* works as follows. Firstly, each the best local

Algorithm 2. *DG-QOA*

input: query q on relations R_1, R_2, \dots, R_n ;
 each R_i have m_i subrelations $R_{i1}, R_{i2}, \dots, R_{im_i}$;
 database statistics, system characteristics, and networks load information.

output: result of the query.

Begin

for $i = 1$ to n do

for $j = 1$ to m_i do

optionPlan($\{R_{ij}\}$) = accessPlan(R_{ij}) {generate all possible local access plan}

optimalPlan(R_{ij}) = prunePlans(optionPlan($\{R_{ij}\}$)) {choose the best local access plan}

run(optimalPlan(R_{ij})) {run all one-subrelation queries}

end-for

end-for

$S = \{R_1, R_2, \dots, R_n\}$ { S is a set of relations that need to execute join operation}

While $|S| < 1$ do

{choose the join pair that will produce the minimum size of intermediate results}

chooseJoinPair(R_x, R_y) where $(R_x, R_y) \subset \{R_1, R_2, \dots, R_n\}$

if R_x and R_y all just have one subrelation **then**

$R_z = \text{join}(R_x, R_y)$ {perform join operation in common way}

else if the join predicate on R_x and R_y is equijoin **then**

$R_z = \text{DG-PHJ}(R_x, R_y)$

else

$R_z = \text{DG-PNJ}(R_x, R_y)$

end-if

end-if

$S = (S \setminus R_x, R_y) \cup (R_z)$

end-while

assemble query results and send to the final node

end. {*DG-QOA*}

access plan of unary operations (select/projection) at relevant database nodes is generated and executed, and then, the *DG-QOA* algorithm determines first join pair (R_x, R_y) which will produce the minimum size of intermediate results with the smallest selectivity. In this phase, if both R_x and R_y have no more than one subrelation, join between R_x and R_y will be performed in the common way that just like traditional distributed database, We consider using index information and semijoin technique to minimize total cost in this situation that each operator of 2-way join just has one subrelation. Otherwise, we check the join predicate. If it is arbitrarily complex other than equijoin, the *DG-PNJ* algorithm should be applied, and if not, the algorithm *DG-PHJ* will be used. We delete R_x and R_y relations from the set S , and insert new relation R_z which produced by $R_x \bowtie R_y$ to the set S . After that, the *DG-QOA* algorithm chooses the next join pair which will produce the minimum size of intermediate relations according to accurate information about the outcomes of foregoing execution, and dynamic system characteristics and networks load. System does this loop until no join operation should be performed. Finally, the query results are assembled and sent to the final node that produces this query.

4.3 Experimental Analysis

Since the original motivation of designing and developing DartGrid II is to provide a platform for Tradition Chinese Medicine (TCM) grid. We carry out performance experiment on TCM grid that includes 17 database nodes. However, no such work for database grid has been reported, we analyze experimental results by comparing to DartGrid I without using the query optimization approach proposed by this paper (see Figure 5).

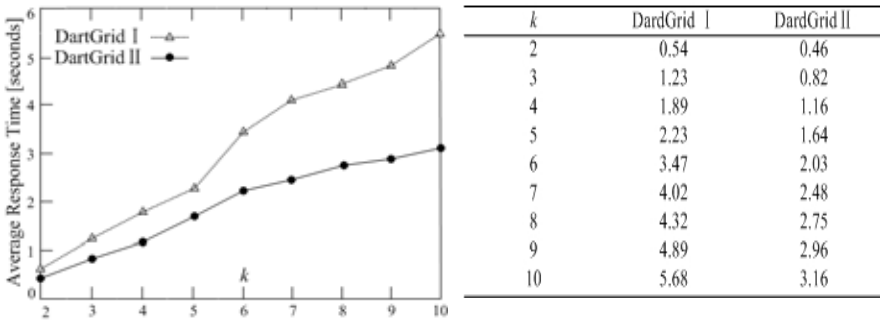


Fig. 5. Average Response Time of DartGrid I and DartGrid II, vary k , 10-way chain query

As shown in Figure 5, we can see that the average response time of DartGrid I and DartGrid II increase with k that denotes the numbers of relations need to join together. Obviously, DartGrid II with *DG-QOA* algorithm is more efficient than DartGrid I. Furthermore, the average response time of DartGrid I dramatically goes up after 5-way query, otherwise, that of DartGrid II increases comparatively smoothly.

5 Conclusion and Future Work

Different types of autonomy and heterogeneity which stem from design, communication, and execution aspects make query optimization in database grid fundamentally different from that in traditional DBMS. These challenges include translating ontology-based global semantic query to local access plan, lack of information about participating database nodes, dynamic and unpredictable system and network parameters, different local capabilities, and more constraints during query optimization in database grid. The existing query processing and optimization technologies must therefore be re-examined in the light of these observation.

After simply overview some major query processing approach, we present the architectural design of a database grid query optimizer which incorporates the query optimization techniques suggested in this paper. And then, the cost model used in database grid has been discussed. finally, we propose the query optimization algorithms (*DG-QOA*, *DG-PHJ*, and *DG-PNJ*) with heuristic, dynamic, and parallel characteristics, heuristics for reducing solution space, dynamic for generating better execution sequences, and parallelism for minimizing response time. The results of preliminary experiment show that our approach is not only efficient but also effective.

For the future, some modules of DartGrid II should be modified for compatible objective with query optimization, and then, more experiment can be made on a large scale to improve algorithm proposed by this paper. Though optimization problem is *NP*-hard, heuristic algorithms are deemed to be justified, another promising randomized and exhaustive research approach, especially Genetic Algorithm and Iterative Dynamic Programming, should be investigated deeply to determine whether these approaches can be integrated into DartGrid II for performance purpose.

Acknowledgement

The work is supported by China 973 fundamental research and development project: The research on application of semantic grid on the knowledge sharing and service of Traditional Chinese Medicine; Intel/University Sponsored Research Program: DartGrid: Building an information Grid for Traditional Chinese Medicine; and China 211 core project: Network based Intelligence and Graphics.

References

1. Donald Kossmann, Konrad Storcker. Iterative Dynamic Programming: A New Class of Query Optimization Algorithms. *ACM Transactions on Database Systems*, March 2000, 25(1): 43–82.
2. D. Bitton, H. Borat, D. J. DeWitt, W. K. Wilkinson. Parallel Algorithms for the Execution of Relational Database Operations. *ACM Trans. Database Syst.*, Sept. 1983, 8(3): 324-353.
3. Galindo-Legaria, C., Pellenkoff, A., Kersten, M. Fast, randomized join-order selection-why use transformations?. In *Proceedings of the 20th International Conference on Very Large Data Bases*, September 1994, 85-95.
4. Graefe, G. DeWitt, D. The EXODUS optimizer generator. In *Proceedings of the ACM-SIGMOD Conference on Management of Data*, May 1987, 160-172.

5. Graefe, G. McKenna, W. J. The volcano optimizer generator: Extensibility and efficient search. In Proceedings of the 9th International Conference on Data Engineering, April 1993, 209-218.
6. Hongjun Lu, Beng-Chin Ooi, Cheng-Hian Goh. Multidatabase query optimization: issues and solutions. Research Issues in Data Engineering, April 1993, 137-143.
7. H. Zhuge, J. Liu, L. Feng, X. Sun and C. He. Query Routing in a Peer-to-Peer Semantic Link Network. Computational Intelligence, 2005, 21(2): 197-216.
8. Ioannidis, Y. E. Kang, Y. C. Randomized algorithms for optimizing large join queries. In Proceedings of the 1990 ACM SIGMOD International Conference on Management of Data, May 1990, 312-321.
9. Jiunn-Chin Wang, Jorng-Tzong Horng, Yi-Ming Hsu. A genetic algorithm for set query optimization in distributed database systems. IEEE International Conference on Systems, Man, and Cybernetics, October 1996, 3: 14-17.
10. J. Veijalainen, Popescu-Zeletin. Multidatabase systems in ISO/OSI environment. Standards in Information Technology and Industrial Control, 1988, 83-97.
11. Lanzelotte, R., Valduries, P., Zait, M. On the effectiveness of optimization search strategies for parallel execution spaces. In Proceedings of the Conference on Very Large Data Bases, August 1993, 493-504.
12. M. Tamer Özsu, Patrick Valduries. Principles of Distributed Database Systems. Prentice Hall, Inc., 1999.
13. Ono, K., Lohman, G. Measuring the complexity of join enumeration in query optimization. In Proceedings of the 16th International Conference on Very Large Data Bases, August 1990, 314-325.
14. Palermo, F. P. 1974. A data base search problem. In Information Systems COINS IV, 1974, 67-101.
15. P. A. Bernstein, N. Goodman, et al., Query Processing in a System for Distributed Database (SDD-1). ACM trans. Database Syst, December 1981, 6(4): 602-625.
16. P. Valduries, G. Gardarin. Join and Semi-join Algorithms for a Multi Processor Database Machine. ACM Trans. Databases Syst, March 1984, 9(1): 133-161.
17. Qiang Zhu. Query Optimization in Multidatabase Systems. Proc. the Centre for Advanced Studies Conf. on Collaborative research, Nov. 1992, 111-127.
18. Selinger, P. G., Astrahan, M. M., Lorie, R. A., Price, T. G. Access path selection in a relational database management system. In Proceedings of the ACM SIGMOD International Conference on Management of Data, May-June 1979, 23-34.
19. Selinger, P. G., M. Adiba. Access Path Selection in Distributed Data Base Management Systems. In Proc. First Int. Conf. on Data Bases, 1980, 204-215.
20. Shekita, E., Young, H., Tan, K. -L. 1993. Multi-join optimization for symmetric multiprocessors. In Proc. Conf. on Very Large Data Bases, August 1993, 479-492.
21. Steinbrunn, M., Moerkotte, G., Kemper, A. Heuristic and randomized optimization for the join ordering problem. August 1997, 191-208.
22. Swami, A. Optimization of large join queries: Combining heuristics and combinational techniques. In Proceedings of the ACM Conference on Management of Data, May 1989, 367-376.
23. Y. E. Ioannidis, E. Wong. Query optimization by simulated annealing. In proc. ACM SIGMOD Int. Conf. on Management of Data, June 1987, 9-22.
24. Zhaohui Wu, Huajun Chen, Changhuang, Guozhou Zheng, Jiefeng Xu. DartGrid: Semantic-Based Database Grid. ICCS 2004, 59-66.