July 2006, Vol.21, No.4, pp.597–608 *J. Comput. Sci. & Technol.*

Dynamic Query Optimization Approach for Semantic Database Grid

Xiao-Qing Zheng (郑骁庆), Hua-Jun Chen (陈华钧), Zhao-Hui Wu (吴朝晖), and Yu-Xin Mao (毛郁欣)

Grid Computing Lab, College of Computer Science, Zhejiang University, Hangzhou 310027, P.R. China

E-mail: zxqingcn@zju.edu.cn, huajunsir@zju.edu.cn, wzh@zju.edu.cn, maoyx@zju.edu.cn

Revised May 6, 2006.

Abstract Fundamentally, semantic grid database is about bringing globally distributed databases together in order to coordinate resource sharing and problem solving in which information is given well-defined meaning, and DartGrid II is the implemented database gird 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 traditional distributed query optimization. These differences are shown to be the consequences of autonomy and heterogeneity of database nodes in database grid. Therefore, more challenges have arisen for query optimization in database grid than traditional distributed database. Following this observation, the design of a query optimizer in DartGrid II is presented, and a heuristic, dynamic and parallel query optimization approach to processing query in database grid is proposed. A set of semantic tools supporting relational database integration and semantic-based information browsing has also been implemented to realize the above vision.

Keywords database integration, query optimization, semantic database grid

1 Introduction

The Web has changed the way people communicate with each other and the way business is conducted by means of Hypertext Markup Language, Hypertext Transfer Protocol, search engines and browsers. Most of today's web content is suitable for human to read. Typical use of the web today involves people's seeking and making use of information, searching for and getting in touch with other people, reviewing catalogs of online stores and ordering products by filling out forms. But the main obstacle to providing better support to web users is that, at present, the meaning of web content is not machine-accessible. The user must either wade through thousands of irrelevant texts or make intelligent guesses of keywords to narrow down the selection. An optional approach is to represent web content in a form that is more easily machine-processable and to use intelligent techniques to take advantage of these representations[1]. This is the main intention of the semantic web that enables the web resources to be understood by the resource usage and service mechanisms such as the search engines and browsers. At the same time, grid computing is the technology that enables a large-scale distributed computing system to carry out the controlled sharing of computing resources. The semantic web and the grid focus on different aspects of application demands. They should converge on the same final target in the long run, and each area will benefit from the progress of other areas. For example, the extension of the grid in semantics and the extension of the semantic web in computation ability converge at the semantic grid[2]. However, most of existing data are stored in relational databases. Therefore, for semantic grid to achieve its full potential, one critical challenge is how to globally publish, seamlessly integrate and transparently locate geographically distributed database resources with such open settings. Web-scale database integration has been one of the main driving forces for both semantic web research and grid development. DartGrid II proposes a semantic-based approach and provides a set of tools and middlewares to support the global sharing of database resources using grid as platform and dynamically integrate information from autonomous local databases managed by heterogeneous database management systems in dynamic, open and multi-institutional environment. All services in DartGrid II are implemented by employing Globus Grid Toolkit 4 and conform to the standard of open grid services architecture (OGSA) and the specification of open grid services infrastructure (OGSI).

The value of applying semantic web technologies to the information and knowledge in grid application was immediately apparent. It enables user, agent and program to maximumly reuse software, services, information and knowledge. In particular, the semantic web resource description framework (RDF) and the web ontology language (OWL) became W3C recommendations. As grid developers have found a need for interoperable metadata they too are turning to RDF, and grid application developers in domains, e.g., life sciences are already working with ontologies—shared vocabularies

Regular Parer

Supported by the Subprogram of the National Basic Research 973 Program of China under Grant No. 2003CB317006, the National Science Fund for Distinguished Young Scholars of NSFC under Grant No. NSFC60533040, the Program for New Century Excellent Talents of China under Grant No. NCET-04-0545, and the NSFC under Grant No. NSFC60503018. This paper is a substantially revised and extended version of the paper, Query optimization in database Grid, *Proc. 4th Int. Grid and Cooperative Computing, Beijing, China, November 30–December 3, 2005.*

which can be expressed in OWL[3]. We embody and exploit these standards and use ontology to define conceptual model or standard terminology and the relations between them in certain domain. Databases in DartGrid II are semantically registered to the grid service called semantic registry service by mapping the schemata of relational databases to the semantics of ontologies. End-user browses the ontologies 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 converting the shared ontologies to the schemata of local databases. Query results will be returned to user as semantically wrapped format (RDF) and presented in semantic browser. Consequently, DartGrid II provides a uniform access to diverse database resources for supporting global query, data mining, e-learning and other high-level applications in which every database node can join or quit the system dynamically.

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 gird system (DBGS), which stem from autonomy and heterogeneity of the database nodes participating in DBGS, query optimization techniques in distributed DBMS cannot trivially 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 classifies site autonomy into three types: design, communication, and execution in [4].

Design autonomy implies that the database nodes are responsible for optimizing local access paths and query processing methods. Consequently, reliable statistical information that is needed for effective global query optimization is not readily available and may not remain accurate as the 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 it 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 be 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 objects may be represented in more than one database nodes, but these representatives are not always structurally compatible in DBGS. By the way, database grid system is also different from multi-database system since DBGS supports and allows the database nodes dynamically participate in or quit 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 and distributed 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 gird. In the following discussion, the global data model 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. In Section 5, the implementation of some visual semantic tools and the application in traditional Chinese medicine are introduced. The conclusions and future work are summarized in Section 6.

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 DBMSs. 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[5].

Typical exhaustive search algorithm is dynamic programming proposed by Selinger *et al.* in [6], and [7] 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[8] and Volcano[9]. Kossmann and Storcker[5] present a new class of query optimization algorithms that are based on iterative dynamic programming (IDP) and declare 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 the plans they produced 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 introduced by Palermo[10], Swami[11], Shekita *et al.*[12] and Steinbrunn *et al.*[13]. These algorithms construct plans in a bottom-up way, that carry out a very simple and rigorous selection of the join order. With every iteration of the minimum selectivity or greedy loop, these algorithms apply a plan evaluation function in order to select the next best join.

To avoid the high cost of exhaustive search, randomized strategies, such as simulated annealing proposed by Ioannidis and Wong[14] and genetic algorithms by Wang *et al.*[15] have been proposed. They try to find a good solution, not necessarily the best one, but avoiding the high cost of optimization. The best known randomized algorithm is called 2PO that is a combination of iterative improvement and simulated annealing proposed by Ioannidis and Kang[16]. Other representatives of this class are in [17, 18].

Other important approaches of query optimization should be given appropriate attention. SDD-1[19] is derived from an earlier method called *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*[20] 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 proposed by Bitton *et al.*[21] is the simplest one and the most general. The algorithm of parallel hash join for specific multiprocessor architecture is given in [22].

Zhuge *et al.*[23] also propose an interesting query routing approach in a peer-to-peer (P2P) semantic link network. He points out that the scalability and autonomy make the P2P network a promising underlying infrastructure for a scalable semantic or knowledge grid. The proposed approach consists of an automatic semantic link discovery method, a tool for building and maintaining P2P semantic link networks (P2PSLNs), a semantic-based peer similarity measurement for efficient query routing, and the schema mapping algorithms for query reformulation and heterogeneous data integration.

Although many algorithms have been proposed for optimizing query-processing in centralized, distributed, and multi- database systems, database grid query optimization problem is fundamentally different from them. Therefore, the existing query processing and optimization technologies must be re-examined and another approach should be developed for database grid.

3 Design of Database Grid Query Optimizer

We would like to give the layered architecture of DartGrid II (See Fig.1) before discussing the design idea of database grid query optimizer. Generally, DartGrid II can be divided into five layers: resource layer, basic service layer, semantic service layer, collective service layer and client layer, and these elements can be organized in hierarchical or P2P model dynamically. The main functionality and characteristics of each layer are elaborated of as follows.

Resource layer. The resources of this layer are believed to be the most fundamental resources within a virtual organization. Data resources include not only database resources, but also file and multimedia resources (although we focus our attention on the database resources in this paper).

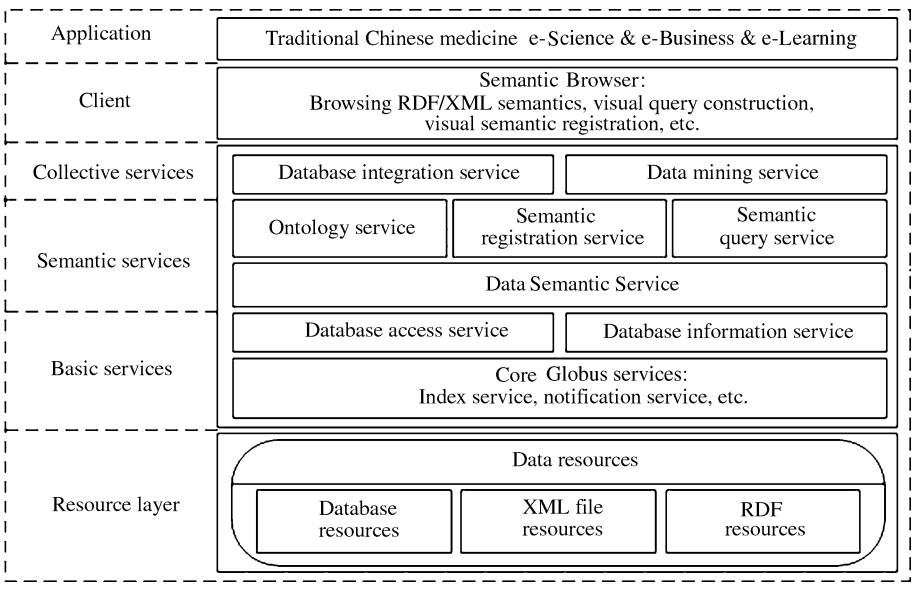


Fig.1. Architecture of DartGrid II.

Basic service layer. We built database access and database information service upon the Globus grid services at this level. Database access service supports typical remote operations on database contents, including retrieval, insertion, deletion of data, and modification of data schema. Database information service supports inquiries about the meta-information of databases. This meta-information consists of schema definition, the description of database management systems, privilege information and statistics information (including CPU utilization, available storage space, active session number, etc.).

Semantic service layer. Semantic layer is the level where ontology-based interaction happens. Database semantic service is used to export the relational schemata of source databases as RDF/OWL semantic descriptions typically by local database administrators. Ontologies can be viewed as mediated schema and are published by ontology service for user to browse RDF/OWL or define semantic queries. Semantic registration service establishes the mappings from source data semantics to shared ontologies. Semantic query service accepts semantic queries, inquires of semantic registration service to determine which databases are capable of providing the answer, then rewrites the queries in terms of relational schema. The semantic queries will be ultimately converted into SQL queries.

Collective service layer. This layer includes data mining service and database monitoring service. Upon the semantic services, a data mining service has been developed for distributed knowledge discovery and data mining. Database monitoring service maintains an entire view of current available database resources and the information about their status. This service also provides some remote control functionalities such as dynamically adding database resources, remotely starting up database services and so forth.

Semantic browser client. DartGrid II provides end-user with a uniform visual interface, called semantic browser to interact with various services and manage large-scale resources. Semantic browser mainly offers the following functionalities: browse the RDF/OWL semantics graphically, visually construct a semantic query and perform semantic mapping from resource schema to RDF/OWL ontologies. Now we put this browser on the Web.

The role of a query processor in DartGrid II is to map a high-level semantic query on the ontologies into a sequence of database operations on the relevant database nodes. We can dynamically create the semantic mapping between ontologies and distributed information bases. As shown in Fig.2, the RDF model is directly connected with the schema of relational database. We refer to database grid query optimization as generating 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 a 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 query optimizer. Fig.3 shows the architecture of the database grid query optimizer.

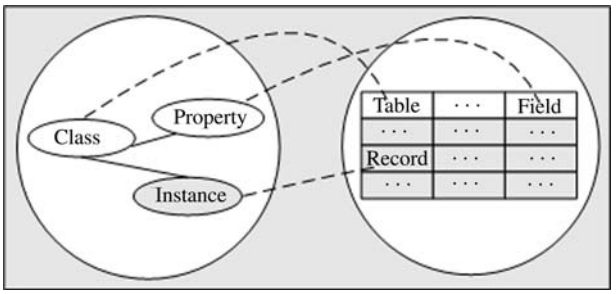


Fig.2. Mapping between the semantics of ontologies and the schema of relational database.

When GOQ receives a semantic query from an 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. Formal semantics and reasoning support is provided through the mapping of web ontology language on logics. The current advanced web ontology language (OWL) is based on description logics. OWL builds on RDF and RDF Schema and uses RDF's XML-based syntax. The rules for converting the concept expression of description logics to SQL are discussed in [24]. 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 communication costs. An immediate method for producing the sequence of operations 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.

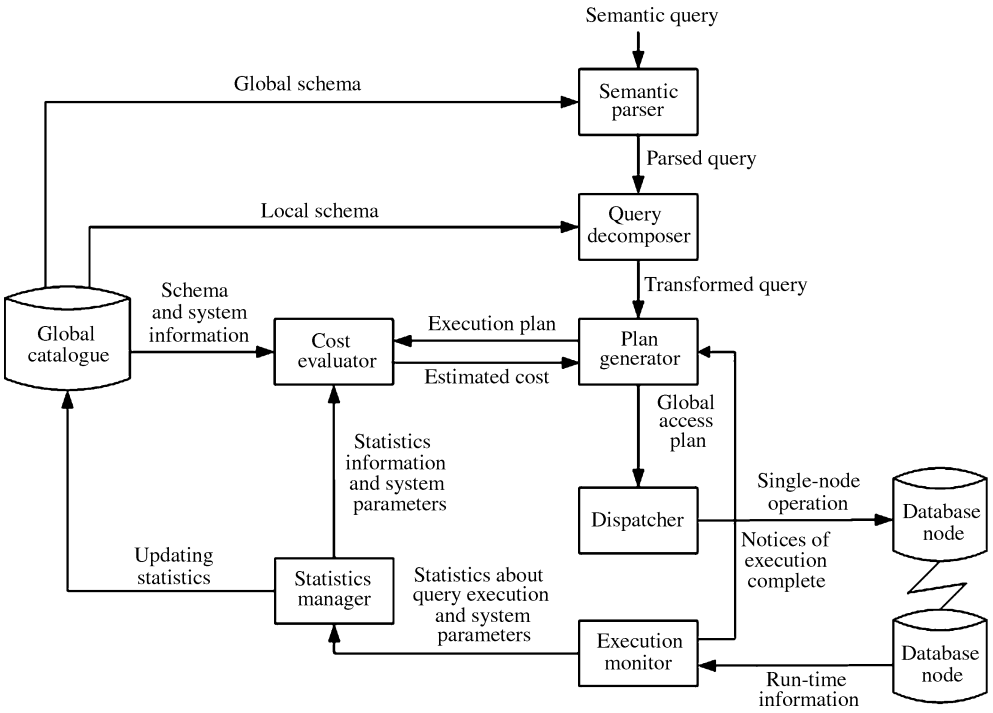


Fig.3. Architecture of database grid query optimizer.

The most powerful search strategy used by query optimizers is dynamic programming, which starts 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 relations 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 depends on accurate statistics information. But in the environment of the database grid with numerous database nodes that will dynamically participate in or quit the grid system, it is impossible and highly expensive to maintain accurate and complete database statistics or errors in these estimates can lead to bad performance. Therefore, dynamic programming and its variances are not suitable to the situation of database grid.

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 the 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 in the above way and executes join operations in parallelism to minimize response time (more discussion in Section 4). At any point of execution, the choice of the 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 the 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 the 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: heuristic for reducing solution space, dynamic for generating better execution sequences, and parallel for minimizing response time.

4.1 Cost Model

The task of an optimizer is nontrivial since for a given query there can be a large number of possible execution plans. Thus, query optimization can be viewed as a difficult search problem. In order to solve this problem,

we need to provide a cost estimation technique so that a cost may be assigned to each plan in the search space. Intuitively, this is an estimation of the resources needed for the execution of the plan.

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 time incurred in the operations of query in participating database nodes and transferring intermediate results among them. Response time 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:

$$Total\_time_{ij}(x, y) = Init + Ship_{ij}(x) + Process_j(x, y).$$

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

Note that above formula does not consider some factors like network and other dynamic characteristics of database nodes. In grid environment, we cannot 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 $Process_j(x, y)$ be 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} represents the factor of network load from node i to node j ($NL_{ij} = NL_{ji}$), 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 results which are the proportions of tuples participating in the join. The join selectivity factor, SF_J, of relation R and S is a real value between 0 and 1:

$$SF_J(R, S) = card(R \bowtie S)/card(R) * card(S).$$

where $card(R)$ represents the number of tuples of 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 relations R and S is not available in global catalogue during the run-time, we use the 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 manager. Database grid query optimizer retrieves the 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 objects may be represented in more than one nodes in a database grid system (see Fig.4). The top of Fig.4 is a semantic view of ontologies and the bottom is four different database nodes. The attributes of relation register to the properties of ontologies according to their semantic meaning. Nodes 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 decomposed into fragments $R_1, R_2, \ldots, R_n$ 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 referred to database gird. Therefore, the union of two relations (we call it subrelation) that materialize the same relational schema at different nodes is not always equal to empty. Let us illustrate the idea of our algorithms using the query in Example 1.

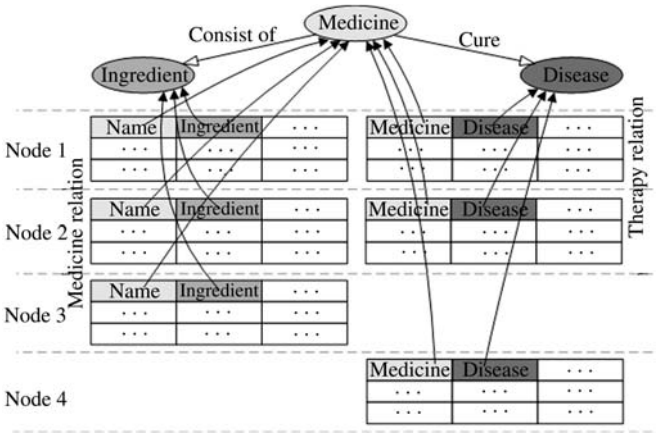


Fig.4. Materialization of the same relation schema at different nodes in database grid.

Example 1. Retrieve all diseases and their available medicines as well as ingredients. 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 $\cup_{(nodes=1,2,4)} R_{therapy}$ ($\cup$ denotes union operation); and execute $R_{medicine} \bowtie R_{therapy}$ at node 1.

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

However, above two access plans have expensive cost obviously. But this scenario often happens in database grid situation and we propose DG-PHJ algorithm (see Algorithm 1) by extending the parallel hash join algorithm introduced by Özsu and Valduriez in [25] to meet this difficulty. The basic idea of DG-PHJ algorithm is to partition relations R and S into the same number p of mutually exclusion sets $R_1, R_2, \ldots, R_p$ and $S_1, S_2, \ldots, S_p$ by employing hash function on the join attribute, such that:

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

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.

Algorithm 1. DG-PHJ

input: $R_1, R_2, \ldots, R_m$: subrelations of relation R;
$S_1, S_2, \ldots, S_n$: subrelations of relation S;
JP: join predicate.

output: $T_1, T_2, \ldots, T_p$: result subrelations.

begin{JP is $R.A = S.B$ and h 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 ($j = 1, \ldots, p$)
    for $j = 1$ **to** p **do**
      send R_{ij} to node j
    endfor
  endfor
  for $i = 1$ **to** n **do in parallel** {hash S on the join attribute}
  begin
    $S_{ij} \leftarrow$ apply $h(B)$ to S_i ($j = 1, \ldots, p$)
    for $j = 1$ **to** p **do**
      send S_{ij} to node j
    endfor
  endfor
  for $j = 1$ **to** p **do in parallel** {perform the join at each p-node}
  begin
    $R_j \leftarrow \cup_{(i=1\ to\ m)} R_{ij}$ {receive from R-nodes}
    $S_j \leftarrow \cup_{(i=1\ to\ n)} S_{ij}$ {receive from S-nodes}
    $T_j \leftarrow$ JOIN(R_j, S_j, JP)
  endfor
end {DG-PHJ}

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. Fig.5 shows the application of DG-PHJ algorithm with Example 1. We assume that the results are produced at nodes 1, 2, and 3. The arrow in Fig.5 indicates data transfer. However, DG-PHJ algorithm can only 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 (see Algorithm 2) that follows the idea of parallel nested loop algorithm introduced by Özsu and Valduriez in [25].

Algorithm 2. DG-PNL

input: $R_1, R_2, \ldots, R_m$: subrelations of relation R;
$S_1, S_2, \ldots, S_n$: subrelations of relation S;
JP: join predicate.

output: $T_1, T_2, \ldots, T_p$: result subrelations.

begin
  for $i = 1$ **to** m **do in parallel** {send R entirely to each S-node}
    send R_i to each node containing the relation of S
  endfor
  for $j = 1$ **to** n **do in parallel** {perform the join at each S-node}
  begin
    $R \leftarrow \cup_{(i=1\ to\ m)} R_i$ {receive R_i from R-nodes; R is replicated on S-nodes}
    $T_j \leftarrow$ JOIN(R, S_j, JP) {JOIN is a generic function}
  endfor
end {DG-PNL}

After discussion of the cost model, DG-PHJ and DG-PNL algorithms, now we can propose the global algorithm for query optimization in database grid, as shown in Algorithm 3. The algorithm of DG-QOA works as follows. Firstly, each best local access plan of unary operations (select/projection) at the relevant database nodes is generated and executed, and then, DG-QOA algorithm determines the first join pair (R_x, R_y) that 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, the join between R_x and R_y will be performed in the common way that just likes traditional distributed database. We consider using index information and semijoin technique to minimize the 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 produced by $R_x \bowtie R_y$ to the set S. After that, 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, dynamic system characteristics and networks load. The 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.

Algorithm 3. DG-QOA

input: query q on relations $R_1, R_2, \ldots, R_n$; each R_i has m_i subrelations $R_{i1}, R_{i2}, \ldots, R_{im_i}$; 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}
    endfor
  endfor
  $S = \{R_1, R_2, \ldots, R_n\}$ {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 $(R_x, R_y) \subset \{R_1, R_2, \ldots, R_n\}$
    if R_x and R_y all just have one subrelation **then**
      R_z = 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 = DG-PHJ(R_x, R_y)
    else
      R_z = DG-PNJ(R_x, R_y)
    endif
    endif
    $S = (S \backslash R_x, R_y) \cup (R_z)$
  endwhile
  assemble query results and send to the final node
end {DG-QOA}

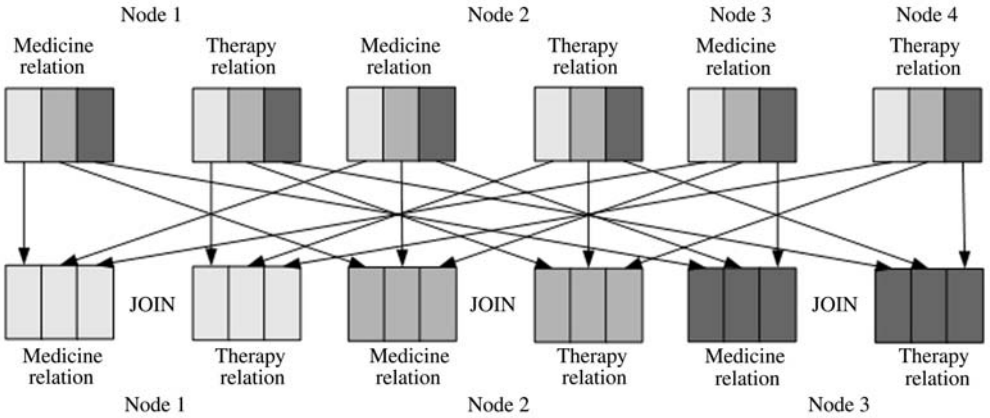


Fig.5. Example of DG-PHJ algorithm.

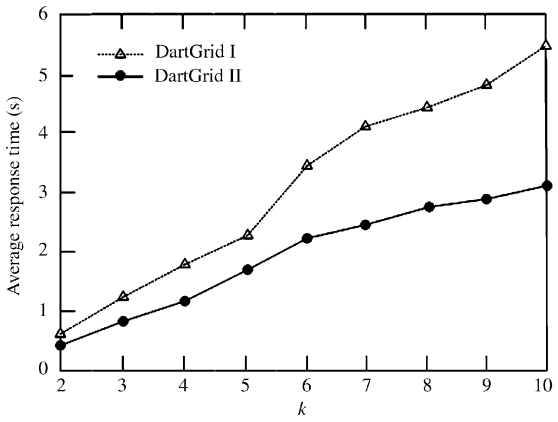


Fig.6. Average response time of DartGrid I and DartGrid II, varying k in 10-way chain query.

4.3 Experimental Analysis

The original motivation of designing and developing DartGrid II is to provide a platform for Traditional Chinese Medicine (TCM) grid that supports TCM language knowledge storage, concept-based information retrieval and information integration. We carry out performance experiments on TCM grid that includes 17 database nodes here. However, no such work for database gird has been reported, we analyze experimental results compared to DartGrid I without using the query optimization approach proposed in this paper.

As shown in Fig.6, we can see that the average response time of DartGrid I and DartGrid II increase in 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 II dramatically goes up after 5-way query, otherwise, that of DartGrid II increases comparatively smoothly.

5 Implementation and Application

TCM is a complete system of medicine encompassing the entire range of human experience. Thousands of scientific studies that support traditional Chinese medical treatments are published yearly in journals around the world. However, even patients who benefit from treatments such as acupuncture or Chinese herbal therapy may not understand all the components of the TCM

system. That may be, in part, because TCM is based on the dynamic understanding of energy and flow that has more to do with western physics than western medicine. TCM embodies rich dialectical thoughts, such as the holistic connections and the unity of yin and yang[26].

With the development of information technology and wide use of the Internet, immense amount of disparate isolated medical databases, electronic patient record (EPR), hospital information systems (HIS) and knowledge sources were developed. Unfortunately, a large amount of complex, polysemous and ambiguous terminologies in TCM is the main obstacle for medical information sharing and reuse[27]. We developed a large TCM ontology in which about 8,000 class concepts and 50,000 instance concepts are defined together with 16 distributed groups over China. The 14 core sub-ontologies are defined according to the disciplines of TCM based on the most domain experts' viewpoints as shown in Table 1. We also developed a unified web accessible query system of TCM bibliographic databases and specific medical databases to address the distributed, heterogeneous information sources retrieval

Table 1. 14 Core Sub-Ontologies Defined Corresponding to the Disciplines of TCM

Sub-ontologies	Characterization of content
The basic theory of traditional Chinese medicine	Define TCM basic theoretical notions such as yin yang, five elements, symptoms, Etiology, Pathogenesis, Physiology and Psychology etc.
The doctrines of traditional Chinese medicine and relevant science	Define basic notions about the doctrines of TCM, which are based on Chinese ancient philosophy and TCM clinical practice. The concept knowledge of TCM relevant science is also defined.
Chinese materia medica	Contain the natural medicinal materials used in TCM practice such as plants, animals and minerals.
Chemistry of Chinese herbal medicine	Contain basic notions of chemical ingredients such as rhein, Arteannuin and Ginsenoside Rb2, which are distilled, separated, identified from herbal medicine and their structures are also measured.
Formula of herbal medicine	Define basic notions such as benzoin tincture ompound, rhubarb compound and Cinnabar compound etc., which are based on the theory of prescription of formula.
Acupuncture	Define basic notions of the modern acupuncture discipline, which is on the basis of thoughts of ancient acupuncture and uses traditional and modern techniques to study the issues of meridians, point, rules of therapy and mechanism etc.
Pharmaceutics and agriculture	Define basic concepts of medical techniques in the manufacture and process of medicinal materials and ontological categories of agriculture issues relevant to medicine planting.
Humanities	Define basic notions about terminology interpretation and relevant knowledge of TCM ancient culture and TCM theories.
Informatics and philology	Contain basic notions of TCM relevant informatics and philology.
Medicinal plants and other resources	Define the medicinal plants and other resources, which are used to health care and disease prevention/cure.
Other natural sciences	Define basic notions of TCM relevant disciplines, which study on the natural and physical phenomenon.
Prevention	Define basic notions of the science of preventing the occurrence and development of diseases.
Administration	Define basic notions of medical research organizations and relevant administration.
Geography	Contain the basic notions (e.g., toponym, climate and soil) that are relevant to TCM.

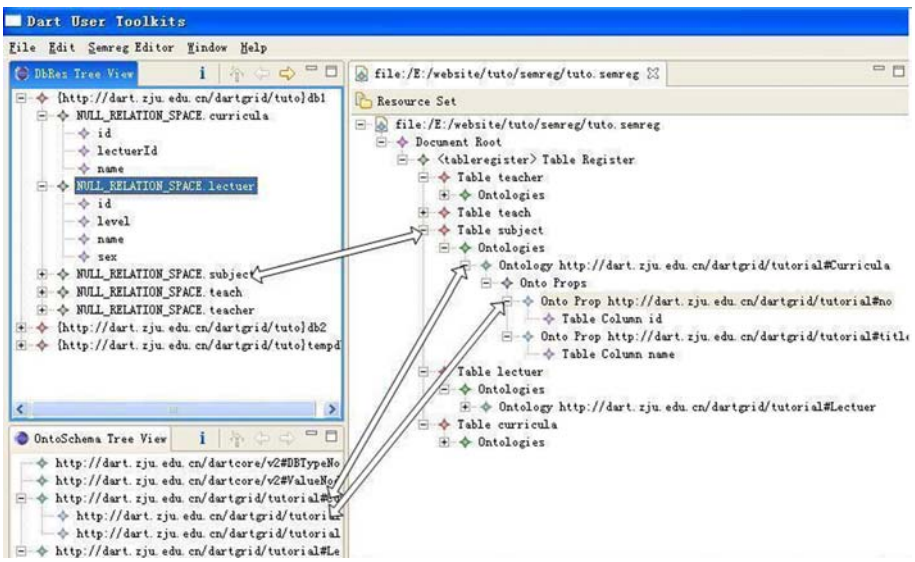


Fig.7. Interface of visual semantic mapping tool.

in TCM by using the technology of DartGrid II (knowledge discovery and data mining are also supported as high-level applications).

According to the techniques described in the previous sections, the DartGrid II system provides a uniform semantic query interface based on RDF model for the set of registered relational database resources. To perform querying at the semantic layer, we develop a semantic query language, Q3 (see [28] for more details). Every query in Q3 can be viewed as an OWL (web ontology language) class definition, and query processing is reduced as computing instances satisfying the querying concept definition. The statement of the semantic query about the name, usage and composition of Chinese medical formulas that can attend influenza is as follows:

```
q3:query
  q3:pattern [
    a tcm:Chinese_medical_formula;
    tcm:name [];
    tcm:usage_and_dosaga [];
    tcm:composition [];
    tcm:treat [ a tcm:disease;
    tcm:name "influenza";
    tcm:pathogenesis [];
    tcm:symptom_complex []
    ]
].
```

To speed up the process of semantic mapping between the semantics of ontologies and the schemata of relational databases, a visual semantic mapping tool is developed (see Fig.7). The top-left-hand panel shows the database schema hierarchy including the names of databases, tables and the corresponding fields, while the bottom-left-hand portion displays the concepts and properties of user-defined ontologies. The right panel of Fig.7 is the results of semantic mapping. These mapping can be defined easily by dragging and dropping left items to the appropriate position of the right part.

Normal users can enter DartGrid II system by the web site of *DartSearch* that is similar to keyword-based search engines, such as AltaVista, Yahoo, and Google. We continue above example of influenza used in the illustration of Q3 language. The return of *DartSearch* is shown in Fig.8 while influenza is used as keywords. The items are listed in the correct order which are pertinent to the keywords of searching. At the bottom of each item there are concept, related concepts and the corresponding matching degree. The concept holds the instance from which the content of item is drawn and here it is traditional Chinese patent medicine for this example. The concept and related concepts are connected by semantic links. We can browse the relevant concepts conveniently by clicking on the appropriate links. After that, we can consider a set of RDFS (RDF schema) semantic constraints such as *rdfs:range*, *rdfs:domain*, *rdfs:subPropertyof*, and *rdfs:subClassof* to build a precious query. The final results of the query of "TCM Prescriptions that can cure influenza" are shown in Fig.9.

The system has been deployed at Chinese Academy of Traditional Chinese Medicine and currently provides access to over 70 databases including TCM herbal medicine databases, TCM compound formula databases, clinical symptom databases, traditional Chinese drug databases, traditional Tibetan drug databases, TCM product and enterprise databases, and so on. The current TCM ontologies include 28 RDF classes, 255 RDF properties, 9 *rdfs:subClassof* constraints, and 25 *rdfs:subPropertyof* constraints. We found the *rdfs:subPropertyof* is very useful in practice. Indeed, about 30% of the 70 TCM databases share some similar properties with each other.

Fig.8. Results of *DartSearch* by the first step of keyword-based search.

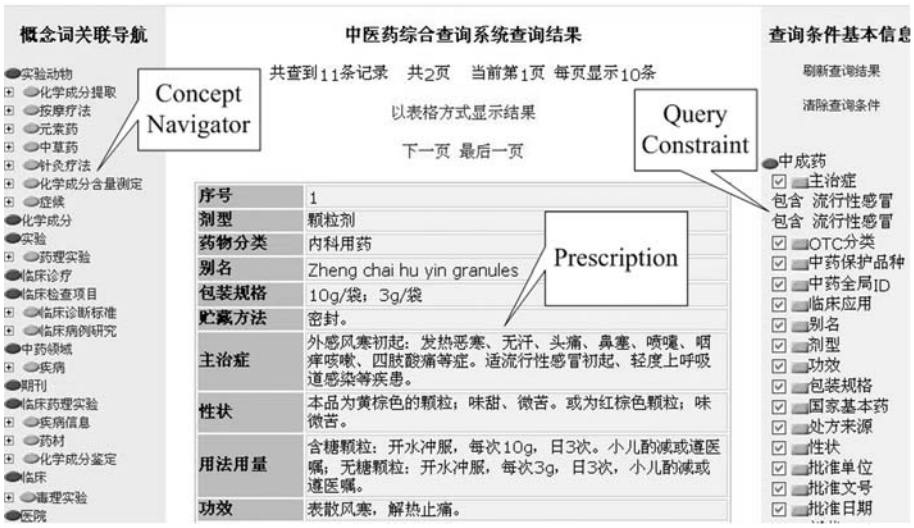


Fig.9. Final results of *DartSearch* by precious query.

6 Conclusion and Future Work

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

After simply discussing some major motivation of designing and developing DartGrid II and the corresponding algorithms for the query optimization, we present the architectural design of the database gird query optimizer that 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. The results of preliminary experiment show that our approach is not only efficient but also effective. A set of visual semantic tools and application in the TCM domain are reported.

For the future, some modules of DartGrid II should be modified for compatible objective with query optimization, and then, more experiments can be made on a large scale to improve the algorithms proposed in 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. However, practical scalability still needs to be tested if the number of databases becomes larger.

References

[1] Antoniou G, Harmelen F V. A Semantic Web Primer. Massachusetts Institute of Technology Press, 2004, pp.1–19.

[2] Zhuge H. Resource space grid: Model, method and platform. *Concurrency and Computation: Practice and Experience*, Wiley InterScience, 2004, 16(14): 1385–1413.

[3] Roure D D, Jennings N R, Shadbolt N R. The semantic grid: Past, present, and future. In *Proc. the IEEE*, 2005, 93(3): 669–681.

[4] Veijalainen J, Popescu-Zeletin R. Multidatabase systems in ISO/OSI environment. Standards in Information Technology and Industrial Control, N E Malagardis, T J Williams (eds.), Netherlands, 1988, pp.83–97.

[5] Kossmann D, Storcker K. Iterative dynamic programming: A new class of query optimization algorithms. *ACM Trans. Database Systems*, 2000, 25(1): 43–82.

[6] Selinger P G, Astrahan M M, Lorie R A, Price T G. Access path selection in a relational database management system. In *Proc. ACM SIGMOD Int. Management of Data*, Boston, Massachusetts, USA, May–June 1979, pp.23–34.

[7] Ono K, Lohman G. Measuring the complexity of join enumeration in query optimization. In *Proc. 16th Int. Very Large Data Bases*, Brisbane, Queensland, Australia, August 1990, pp.314–325.

[8] Graefe G, David J D. The EXODUS optimizer generator. In *Proc. ACM SIGMOD Int. Management of Data*, San Francisco, California, USA, May 1987, pp.160–172.

[9] Graefe G, Mckenna W J. The volcano optimizer generator: Extensibility and efficient search. In *Proc. 9th Int. Data Engineering*, Vienna, Austria, April 1993, pp.209–218.

[10] Palermo F P. A data base search problem. In *Proc. Int. 4th Symposium on Computer and Information Science*, Restion, Virginia, USA, 1972, pp.67–101.

[11] Swami A. Optimization of large join queries: Combining heuristics and combinational techniques. In *Proc. ACM Int. Management of Data*, Portland, Oregon, May 1989, pp.367–376.

[12] Shekita E, Young H, Tan K L. Multi-join optimization for symmetric multiprocessors. In *Proc. Int. Very Large Data Bases*, Dublin, Ireland, August 1993, pp.479–492.

[13] Steinbrunn M, Moerkotte G, Kemper A. Heuristic and randomized optimization for the join ordering problem. *The International Journal on Very Large Data Bases*, 1997, 6(3): 191–208.

[14] Ioannidis Y E, Wong E. Query optimization by simulated annealing. In *Proc. ACM SIGMOD Int. Management of Data*, San Francisco, California, USA, June 1987, pp.9–22.

[15] Wang J C, Horng J T, Hsu Y M. A genetic algorithm for set query optimization in distributed database systems. In *Proc. IEEE Int. Systems, Man, and Cybernetics*, Beijing, China, 1996, pp.14–17.

[16] Ioannidis Y E, Kang Y C. Randomized algorithms for optimizing large join queries. In *Proc. ACM SIGMOD Int. Management of Data*, Atlantic City, New Jersey, USA, May 1990, pp.312–321.

[17] Lanzelotte R, Valduries P, Zait M. On the effectiveness of optimization search strategies for parallel execution spaces. In *Proc. Int. Very Large Data Bases*, Dublin, Ireland, August 1993, pp.493–504.

[18] Galindo L C, Pellenkoft A, Kersten M. Fast, randomized join-order selection-why use transformations. In *Proc. 20th Int. Very Large Data Bases*, Santiago de Chile, Chile, September 1994, pp.85–95.

[19] Bernstein P A, Goodman N *et al.* Query processing in a system for distributed database (SDD-1). *ACM trans. Database System*, 1981, 6(4): 602–625.

[20] Selinger P G, Adiba M. Access path selection in distributed database management systems. In *Proc. 1st Int. Data Bases*, Aberdeen, Scotland, 1980, pp.204–215.

[21] Bitton D, Boral H, DeWitt D J, Wilkinson W K. Parallel algorithms for the execution of relational database operations. *ACM Trans. Database System*, 1983, 8(3): 324–353.

[22] Valduriez P, Gardarin G. Join and semi-join algorithms for a multi processor database machine. *ACM Trans. Databases System*, March 1984, 9(1): 133–161.

[23] Zhuge H, Liu J, Feng L, Sun X, He C. Query routing in a peer-to-peer semantic link network. *Computational Intelligence*, 2005, 21(2): 197–216.

[24] Wu Z H, Chen H J, Huang C *et al.* DartGrid: Semantic-based database grid. In *Proc. International Conference on Computational Science*, Kraków, Poland, 2004, pp.59–66.

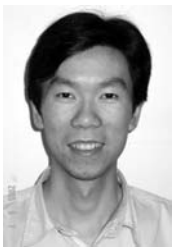
[25] Tamer M, Patrick V. Principles of Distributed Database Systems. Prentice Hall, Inc., 1999.

[26] Yin H H, Zhang R E. The Basic Theory of Traditional Chinese Medicine. Shanghai Scientific and Technical Publisher, Shanghai, May 1984.

[27] Zhou X Z, Wu Z H, Yin A N *et al.* Ontology development for unified traditional Chinese medical language system. *Journal of Artificial Intelligence in Medicine*, 2004, 32(1): 15–27.

[28] Cheng H, Wu Z H, Mao Y X. Q3: A semantic query language for dart database grid. In *Proc. Int. Grid and Cooperative Computing*, Wuhan, China, 2004, pp.372–380.

Xiao-Qing Zheng is a Ph.D. candidate in the College of Computer Science, Zhejiang University. He received his M.S. degree in management science and engineering in 2003. His research interests include knowledge representation and reasoning, grid computing, semantic web and multi-agent systems.



Hua-Jun Chen received his Ph.D. degree in computer science from Zhejiang University in 2005. He got his bachelor's degree in biochemical engineering in 2000. Since 2004, he works as an assistant professor in the College of Computer Science, Zhejiang University. His research interests include grid computing, semantic web and biometric computing.

Zhao-Hui Wu received his Ph.D. degree in computer science from Zhejiang University, China, and Kaiserslautern University, Germany, in 1993. Now he is a professor, doctoral postgraduate supervisor and vice head of the College of Computer Science, Zhejiang University. He invented the first KB-system developing tool, ZIPE in China, in 1990. He proposed the first coupling knowledge representing model, Couplingua, which embodies rule, frame, semantic network and nerve cell network and supports symbol computing and traditional data processing computing. His research interests include distributed artificial intelligence, semantic grid, ubiquitous embedded system and real-time system.

Yu-Xin Mao is a Ph.D. candidate in the College of Computer Science, Zhejiang University. He received his bachelor's degree in computer science from Zhejiang University in 2003. His research interests include grid computing, semantic web, web information sharing, visualization and search engine.