Optimizing Queries with Foreign Functions in a Distributed Environment

Pauray S.M. Tsai and Arbee L.P. Chen, *Member*, *IEEE*

Abstract—Foreign functions have been considered in the advanced database systems to support complex applications. In this paper, we consider optimizing queries with foreign functions in a distributed environment. In traditional distributed query processing, selection operations are locally processed before joins as much as possible so that the size of relations being transmitted and joined can be reduced. However, if selection predicates involve foreign functions, the cost of evaluating selections cannot be ignored. As a result, the execution order of selections and joins becomes significant, and the trade-off for reducing the costs of data transmission, join processing, and selection predicate evaluation needs to be carefully considered in query optimization. In this paper, a response time model is developed for estimating the cost of distributed query processing involving foreign functions. We explore the property of the problem and find an optimal algorithm with polynomial complexity for a special case of it. However, finding the optimal execution plan for the general case is NP-hard. We propose an efficient heuristic algorithm for solving the problem and the simulation result shows its good quality. The research result can also be applied to the advanced database systems and the multidatabase systems where the conversion function defined for the need of schema integration can be considered a type of foreign functions.

Index Terms—Distributed environment, foreign function, query optimization, response time model, simulation.

1 Introduction

Relational database systems have been demonstrated as useful for classical applications. However, with the increasing complexity of data nowadays, traditional relational database systems are not adequate enough to support many complex applications, such as CAD/CAM systems and geographic information systems. Therefore, extended relational database systems [18], [32] and object-oriented database systems [6], [13], [14], [34] are developed to provide complex object support, data types, and functions. In order to efficiently process queries in these "advanced" database systems, the ability of traditional query optimizers needs to be improved.

In the past, several approaches for reducing the cost of function computation have been proposed. Kemper et al. [25] used the technique of function materialization (i.e., the precomputation of function results) to enhance the efficiency of query processing in an object-oriented database system. The object-oriented features such as classification, object identity, and encapsulation are exploited to reduce the rematerialization penalty incurred by update operations. Their approach has been implemented in the experimental object base system GOM [24]. In contrast to function materialization, the method of computing functions in query processing is adopted by most previous papers [7], [10], [19], [35]. For database systems using the latter approach, the query optimizer must take the cost of function computation into consideration because the cost can be very expensive.

In this paper, the problem of optimizing the query with foreign functions in a distributed environment is investigated. A foreign function is considered to be an operation external to the database, which is time-consuming to compute [7]. In traditional distributed query processing [2], [4], [28], [33], selection operations are locally processed before joins as much as possible so that the size of relations being transmitted and joined can be reduced. However, if selection predicates involve foreign functions, the cost of evaluating selections cannot be ignored. As a result, the execution order of selections and joins becomes significant, and the trade-off for reducing the costs of data transmission, join processing, and selection predicate evaluation needs to be carefully considered in query optimization. To our knowledge, the problem of optimizing queries with foreign functions in a distributed environment has not been explored in the prior literature. All the related work which studied query optimization techniques for foreign functions focused on the centralized database system. The details of the related work will be described in Section 2.

In this paper, we use the technique of caching the return values of function calls in query processing. Namely, function computation costs are charged once for each distinct function input. Based on the idea, a response time model is developed to estimate the cost of distributed query processing involving foreign functions. We explore the property of the problem and find an optimal algorithm with polynomial complexity for a special case of it. However, finding the optimal execution plan for the general case is NP-hard. We propose an efficient heuristic algorithm to solve the problem and the simulation result shows its good quality compared with the optimal execution plan. Moreover, from

- *P.S.M. Tsai is with the Department of Information Management, Ming Hsin Institute of Technology, Hsin-Feng, Hsinchu 304, Taiwan, Republic of China. E-mail: pauray@mis.mhit.edu.tw.*
- *A.L.P. Chen is with the Department of Computer Science, National Tsing Hua University, Hsinchu 300, Taiwan, Republic of China. E-mail: alpchen@cs.nthu.edu.tw.*

Manuscript received 11 Mar. 1996; revised 7 July 2000; accepted 19 Jan. 2001; posted to Digital Library 7 Sept. 2001.
For information on obtaining reprints of this article, please send e-mail to: tkde@computer.org, and reference IEEECS Log Number 104335.

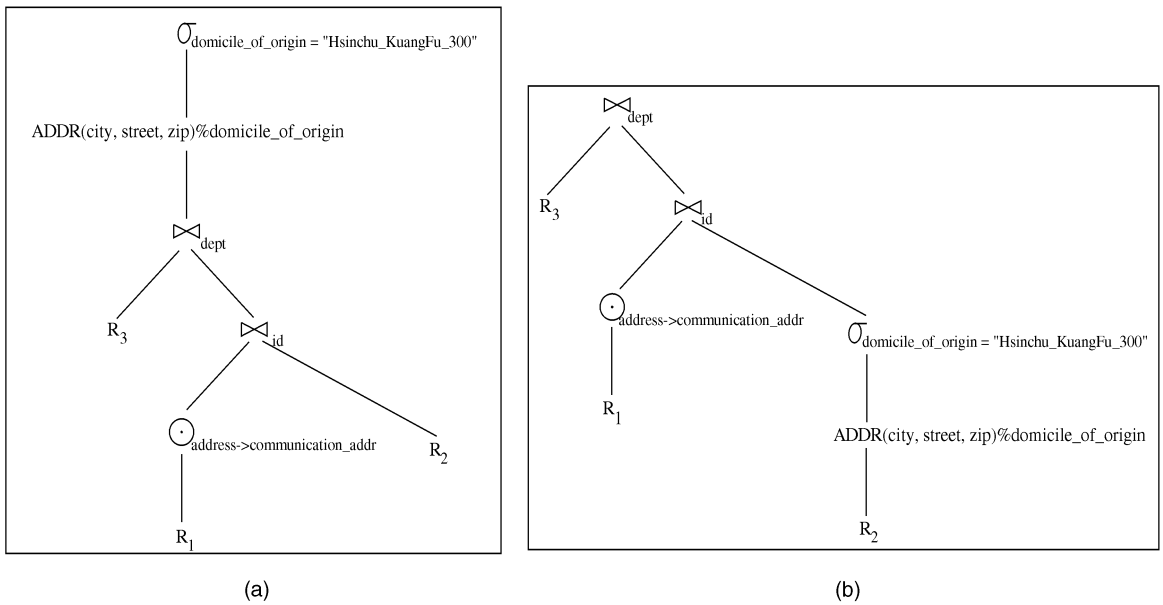

Fig. 1. Two alternative execution plans.

the simulation result, the efficiency of the execution plan generated by the proposed heuristic algorithm greatly outperforms that by the "initial" strategy (i.e., evaluating all the selection predicates before joins), which demonstrates the significance of our work.

The main purpose of this paper is to investigate the effect of foreign functions on distributed query processing. The proposed heuristic algorithm is useful for optimizing queries with foreign functions in a distributed environment. The research result can also be applied to the advanced database systems and the multidatabase systems [1], [5], [12] where the conversion function defined for the need of schema integration can be considered a type of foreign functions. In the following, an example is given to further motivate our approach.

Example 1. Consider a multidatabase system consisting of three individual databases. Relations

$$R_1(id, name, class, dept, address)$$

and

$$R_2(id, parent, city, street, zip),$$

which are stored in databases DB_1 and DB_2, respectively, record the separate information about all the students at X University. In database DB_3, the relation $R_3(name, class, dept)$ records the information about worker_students at Y University. Assume a logically integrated global view is created as a uniform interface *for* the USER to retrieve data in the multidatabase system. The global view is shown below.

$$X\_Student(id, name, class, dept, parent, communication\_addr, domicile\_of\_origin)$$
$$Y\_Worker\_Student(name, class, dept).$$

The attribute *address* in R_1 denotes the communication address, which is renamed to $communication\_addr$ in view $X\_Student$. The aggregation of attributes *city*, *street*, and *zip* in R_2 represents the domicile of origin, which is replaced by the attribute $domicile\_of\_origin$ in view $X\_Student$. Assume the conversion function $ADDR$ is defined to do the aggregation

$$ADDR(e_1, e_2, e_3) = e_1 + '' \_ '' + e_2 + '' \_ '' + e_3,$$

where operator "+" denotes concatenation. The instances in $X\_Student$ can be obtained by performing renaming on $R_1.address$ and the conversion function

$$ADDR(R_2.city, R_2.street, R_2.zip)\%domicile\_of\_origin,$$

where $domicile\_of\_origin$ represents the corresponding attribute for the return value of the function, then joining the converted relations of R_1 and R_2 on the key attribute *id*. The instances in $Y\_Worker\_Student$ is corresponding to those in R_3. Let us consider the following query.

SELECT *.*
FROM $X\_Student$, $Y\_Worker\_Student$
WHERE $X\_Student.dept = Y\_Worker\_Student.dept$
and $X\_Student.domicile\_of\_origin = ''\text{Hsinchu\_KuangFu\_300}''$

Fig. 1 shows two execution plans for the query, in which the notation $\odot_{address \to communication\_addr}$ denotes the rename operation. In Fig. 1a, the conversion function and the selection

$$domicile\_of\_origin = ''\text{Hsinchu\_KuangFu\_300}''$$

are delayed to be executed until joins are processed, while they are considered to be executed before the join processing in Fig. 1b. The selection predicate

$$''domicile\_of\_origin = \text{Hsinchu\_KuangFu\_300}''$$

and the conversion function

$$"ADDR(city, street, zip)\%domicile\_of\_origin"$$

can be considered as

$$"ADDR(city, street, zip) = \text{Hsinchu\_KuangFu\_300}"$$

with foreign function "ADDR." Since the technique of caching return values of "ADDR" is used and the number of distinct inputs for the function may decrease after join processing, the cost to compute "ADDR" after join processing may be lower than that before join processing. Therefore, execution plan (a) may be better than execution plan (b). To efficiently process the query, a cost model is needed to determine a better execution plan.

This paper is organized as follows: The related work is described in Section 2. In Section 3, the problem of minimizing response time for the query with foreign functions is formulated. Moreover, the property of the problem is discussed. In Section 4, an optimal algorithm with polynomial complexity for a special case of it is proposed. Section 5 presents heuristic algorithms for the general case. The simulation results of comparing the execution plan generated by the heuristic algorithm and the optimal plan are presented in Section 6. Moreover, the accuracy of the cost model is discussed. Finally, we conclude with the future work in Section 7.

2 Related Work

Abstract data type (ADT) is a form of function, which has been used in many database applications [3], [22], [31]. To efficiently process a query with ADT functions, the computing cost of the ADT functions should be lowered as much as possible. Chen et al. [10] proposed an approach for optimizing queries with ADT functions. In this work, the expensive ADT function is replaced with a sequence of simpler functions to reduce the cost of query processing. Chaudhuri and Shim [7] studied the problem of optimizing queries with foreign functions. A declarative rule language was provided for expressing the semantic information for the optimization of foreign functions. The purpose of the rewrite rules described in [7] is similar to the rewrite rules for query optimization [30] used in Starburst. However, the rewrite rules in [30] do not involve the optimization of foreign functions. The approaches of [7] and [10] are both heuristics and the effect of joins on evaluating selection predicates with foreign functions is not considered.

Yoo and Sheu [35] investigated optimizing queries with procedural methods and control statements in an object-oriented and symbolic information system. If the cardinality of the resultant relation from joining two relations is smaller than the cardinality of the relation to be joined in which a method is executed for each tuple, then the evaluation of the method is deferred until the join is processed; otherwise, the method is performed before join processing. The execution order of multiple joins and methods is not discussed. Levy et al. [27] proposed an optimization technique, called predicate move-around, to efficiently process queries with aggregate operations and constructs such as NOT EXISTS. However, the predicates are assumed inexpensive.

Kemper et al. [26] proposed the "bypass" technique for optimizing disjunctive queries with expensive predicates. The idea of the bypass processing is to avoid the evaluation of expensive predicates whenever the result of the entire disjunctive selection predicates can be determined by testing other less expensive predicates. Hellerstein and Stonebraker [19] proposed an optimal algorithm, named predicate migration, to arrange the execution sequence of joins and selections with foreign functions, which has been implemented in POSTGRES [32]. Chaudhuri and Shim [9] proposed a conservative local heuristic which guarantees optimality for processing queries with user-defined predicates in several cases. The algorithms in [9], [19] are developed to handle queries with conjunctive expensive predicates in a centralized database system. Therefore, they are inapplicable to a distributed environment. The performance evaluation of the predicate migration algorithm and other predicate placement algorithms is discussed in [20]. Efficient techniques for caching expensive methods are proposed in [21], which can avoid redundant method invocation during query processing. The major differences between the cost models of [19] and our approach in this paper are 1) the change of the number of distinct attribute values for the selection predicate to be executed after the join operation is reflected in the cost model of our approach and 2) the transmission cost is also considered in our approach for a distributed environment.

The subquery predicate provided in SQL can be considered as a type of expensive selections. In System R^* [28], the evaluation of subqueries is deferred until simple selection predicates are computed. However, the R^* optimizer focuses on finding optimal access methods and join orders without considering the execution order of the expensive selections. Extensible query optimizers [16], [17], [29] present different frameworks for the incorporation of new optimization strategies. Graefe and McKenna [16] and Hass et al. [17] provide fixed strategies for searching and applying rules in the query optimization process, while Mitchell et al. [29] allow the extension of the control over the optimization process. The incorporation of our approach with the Epoq optimizer will be discussed in Section 7.

3 Minimizing Response Time for the Query with Foreign Functions

In this section, we study the problem of minimizing response time for the query with foreign functions. For the discussion, the query considered is of the form $\sigma_P(R_1 \bowtie_{p'} R_2)$, where R_1 and R_2 represent relations in different databases, P represents the conjunction of a set of selection predicates $p_1, p_2, \ldots, p_k$, and p' represents a join predicate. The case where a query involves more than one join will be discussed in Section 6.2.

A *simple selection predicate* is of the form $attr_i$ **op** C, where $attr_i$ represents a relation attribute, **op** denotes an operator such as ">," "<," or "=," and C represents a constant. A simple selection predicate is considered a zero-time operation. Conversely, a selection predicate is called an *expensive selection predicate* if it involves a foreign function. An expensive selection predicate is of the form

$Fun_j(attr_j)$ **op** C, where Fun_j represents a function with the relation attribute $attr_j$ as its parameter. Note that the following discussion can be extended to the case where a function has a set of attributes as its parameters. For each selection predicate p_i, $1 \leq i \leq k$, it can be a simple selection predicate or an expensive selection predicate. The join predicate p' considered is of the simple form such as $R_1.a_i = R_2.a_j$, where a_i and a_j are attributes of R_1 and R_2, respectively. For the convenience of discussion, we represent a simple selection predicate "$attr_i$ **op** C" as "$Fun_i(attr_i)$ **op** C," where Fun_i is considered a zero-time function and the return value of $Fun_i(attr_i)$ is equal to the value of $attr_i$.

The query $\sigma_P(R_1 \bowtie_{p'} R_2)$ is expressed as follows:

$$\begin{aligned}\sigma_P(R_1 \bowtie_{p'} R_2) &= \sigma_{p_1 \wedge p_2 \wedge \ldots \wedge p_k}(R_1 \bowtie_{p'} R_2) \\ &= \sigma_{(Fun_1(attr_1)\ \mathbf{op_1}\ C_1) \wedge (Fun_2(attr_2)\ \mathbf{op_2}\ C_2)} \\ &\quad {}_{\wedge \ldots \wedge (Fun_k(attr_k)\ \mathbf{op_k}\ \mathbf{C_k})}\ (R_1 \bowtie_{p'} R_2),\end{aligned}$$

where $\mathbf{op_i}$ and C_i denote an operator and a constant, respectively, $1 \leq i \leq k$. The attribute $attr_i$ is called the *associated attribute* of predicate p_i. If $attr_i$ is an attribute of R_1, the selection predicate p_i and the function Fun_i are called a *related selection predicate* and a *related function* for R_1, respectively; otherwise, they are called a related selection predicate and a related function for R_2.

3.1 Notation and Assumptions

Notation:

- The cardinalities of relations R_1 and R_2 are n_1 and n_2, respectively.
- The sizes of a tuple in R_1 and a tuple in R_2 are b_1 bytes and b_2 bytes, respectively.
- By renaming, the sets of related selection predicates for R_1 and R_2 are $\{p_1^1, p_1^2, \ldots, p_1^m\}$ and $\{p_2^1, p_2^2, \ldots, p_2^n\}$, respectively, where $m + n = k$. The sets of related functions for R_1 and R_2 are $\{Fun_1^1, Fun_1^2, \ldots, Fun_1^m\}$ and $\{Fun_2^1, Fun_2^2, \ldots, Fun_2^n\}$, respectively, and the parameters of functions

  $$Fun_1^1, Fun_1^2, \ldots, Fun_1^m, Fun_2^1, Fun_2^2, \ldots, Fun_2^n$$

  are

  $$R_1.attr_1^1, R_1.attr_1^2, \ldots, R_1.attr_1^m, R_2.attr_2^1, R_2.attr_2^2, \ldots, R_2.attr_2^n,$$

  respectively.
- The times for executing functions

  $$Fun_1^1, Fun_1^2, \ldots, Fun_1^m, Fun_2^1, Fun_2^2, \ldots, Fun_2^n$$

  are

  $$\lambda_1^1, \lambda_1^2, \ldots, \lambda_1^m,\ \lambda_2^1, \lambda_2^2, \ldots, \lambda_2^n,$$

  respectively.
- The selectivities for predicates

  $$p_1^1, p_1^2, \ldots, p_1^m, p_2^1, p_2^2, \ldots, p_2^n$$

  are

  $$\rho_1^1, \rho_1^2, \ldots, \rho_1^m, \rho_2^1, \rho_2^2, \ldots, \rho_2^n,$$

  respectively.
- The numbers of distinct values in

  $$R_1.attr_1^1, R_1.attr_1^2, \ldots, R_1.attr_1^m,$$
  $$R_2.attr_2^1, R_2.attr_2^2, \ldots, R_2.attr_2^n$$

  are $d_1^1, d_1^2, \ldots, d_1^m, d_2^1, d_2^2, \ldots, d_2^n$, respectively.
- The times for scanning a tuple of R_1 and a tuple of R_2 are g_1 and g_2, respectively.
- The transmission time per byte and the setup time of communication for transmitting R_h to the final site where the join is processed are X_h and E_h, respectively, $h = 1, 2$.
- For the join $R_1 \bowtie_{p'} R_2$, there is a probability of β_1 that a tuple in R_1 will find a match in R_2 and a probability of β_2 that a tuple in R_2 will find a match in R_1.
- The time for scanning a join tuple of $R_1 \bowtie_{p'} R_2$ is g_3.
- The time for reading/writing the result of a function computation from/into the cache is t_c.

Assumptions:

- (A1). The values of one attribute are assumed independent of those in another attribute.
- (A2). The technique of caching is used to store the return values of function calls. Thus, if the number of distinct values in $attr_i$ are n, then $Fun_i(attr_i)$ needs to be computed only n times. The cache is assumed to reside in the disk. Therefore, the time for accessing the cache has to be considered.
- (A3). The number of distinct values in an attribute of a relation is assumed proportional to the cardinality of the relation. (For example, let R_1' be the result of performing a selection on R_1. If the cardinality of R_1' is n_1', then the number of distinct values in $R_1'.attr_1^i$ are evaluated as $d_1^i \times \frac{n_1'}{n_1}$, , where $\frac{n_1'}{n_1}$ is considered to be the selectivity of the corresponding selection.)

3.2 The Response Time Model

Assume relations R_1 and R_2 are located at site 1 and site 2, respectively. The expression $p_{1'} \rightarrow p_{2'} \rightarrow \ldots \rightarrow p_{h'}$ in a query execution plan indicates that selection predicates are evaluated in the sequence of $p_{1'}$ first, then $p_{2'}$, etc. Consider an execution plan for the query $\sigma_{p_1 \wedge p_2 \wedge \ldots \wedge p_k}(R_1 \bowtie_{p'} R_2)$:

$$\begin{aligned}&\sigma_{p_1^{i+1} \rightarrow p_1^{i+2} \rightarrow \ldots \rightarrow p_1^m \rightarrow p_2^{j+1} \rightarrow p_2^{j+2} \rightarrow \ldots \rightarrow p_2^n} \\ &((\sigma_{p_1^1 \rightarrow p_1^2 \rightarrow \ldots \rightarrow p_1^i} R_1) \bowtie_{p'} (\sigma_{p_2^1 \rightarrow p_2^2 \rightarrow \ldots \rightarrow p_2^j} R_2)).\end{aligned}$$

The execution plan is processed as follows:

1. *Initial selection processing*:

   $$\sigma_{p_1^1 \rightarrow p_1^2 \rightarrow \ldots \rightarrow p_1^i} R_1$$

   and

   $$\sigma_{p_2^1 \rightarrow p_2^2 \rightarrow \ldots \rightarrow p_2^j} R_2$$

   are processed in site 1 and site 2, respectively.
2. *Transmitting the reduced relations to the final site and performing the join*: The results of $\sigma_{p_1^1 \rightarrow p_1^2 \rightarrow \ldots \rightarrow p_1^i} R_1$ and

$\sigma_{p_2^1 \to p_2^2 \to \ldots \to p_2^j} R_2$ are transmitted to a final site where the join on predicate p' is performed.

3. *Final selection processing*: The selection $p_1^{i+1} \to p_1^{i+2} \to \ldots \to p_1^m \to p_2^{j+1} \to p_2^{j+2} \to \ldots \to p_2^n$ on the join result is performed.

Therefore, the response time for processing the execution plan includes the time for initial selection processing, the time for transmitting the reduced relations to the final site, the time for performing the join, and the final selection. They are further described in the following.

Initial Selection Processing Time: The time for the selection processing is dominated by the relation scan time, the time for accessing the cache, and the time for performing functions. For the computation of function Fun with parameter x, if the result of $Fun(x)$ has been cached, it requires t_c time to read the result; otherwise, $Fun(x)$ is performed and t_c time is needed to write the result into the cache. By assumptions (A1), (A2), and (A3), the times for processing $\sigma_{p_1^1 \to p_1^2 \to \ldots \to p_1^i} R_1$ and $\sigma_{p_2^1 \to p_2^2 \to \ldots \to p_2^j} R_2$ are evaluated as

$$g_1 \cdot n_1 + (n_1 \cdot t_c + \lambda_1^1 \cdot d_1^1) + (\rho_1^1 \cdot n_1 \cdot t_c + \rho_1^1 \cdot \lambda_1^2 \cdot d_1^2) + \ldots + (\Pi_{h=1}^{i-1}(\rho_1^h) \cdot n_1 \cdot t_c + \Pi_{h=1}^{i-1}(\rho_1^h) \cdot \lambda_1^i \cdot d_1^i)$$

and

$$g_2 \cdot n_2 + (n_2 \cdot t_c + \lambda_2^1 \cdot d_2^1) + (\rho_2^1 \cdot n_2 \cdot t_c + \rho_2^1 \cdot \lambda_2^2 \cdot d_2^2) + \ldots + (\Pi_{h=1}^{j-1}(\rho_2^h) \cdot n_2 \cdot t_c + \Pi_{h=1}^{j-1}(\rho_2^h) \cdot \lambda_2^j \cdot d_2^j),$$

respectively.

The Time for Transmitting the Reduced Relations: Let R_1' and R_2' be the results of processing $\sigma_{p_1^1 \to p_1^2 \to \ldots \to p_1^i} R_1$ and $\sigma_{p_2^1 \to p_2^2 \to \ldots \to p_2^j} R_2$, respectively. The cardinalities of R_1' and R_2' are estimated to be $n_1 \cdot \Pi_{h=1}^i \rho_1^h$ denoted by T_1' and $n_2 \cdot \Pi_{h=1}^j \rho_2^h$ denoted by T_2', respectively. Thus, the time for transmitting R_h' to the final site is $T_h' \cdot b_h \cdot X_h + E_h$, where $h = 1, 2$.

The Join Processing Time: We assume the worst-case time complexity of the join operation in this paper. That is, the amount of time needed for processing the join is proportional to the product of relation cardinalities. Therefore, the join processing time is computed as $D' \cdot T_1' \cdot T_2'$, where D' is a constant.

The Final Selection Processing Time: Let R' be the result of executing $R_1' \bowtie_{p'} R_2'$, and r_h' be the cumulative reduction factor of the number of distinct values in an attribute of R_h after the join processing, where $h = 1, 2$. The cardinality of R' is computed as $C' \cdot \beta_1 \cdot \beta_2 \cdot T_1' \cdot T_2'$, where C' is a constant, and r_1' and r_2' are evaluated to be $\Pi_{h=1}^i(\rho_1^h) \cdot \beta_1$ and $\Pi_{h=1}^j(\rho_2^h) \cdot \beta_2$, respectively. Therefore, the number of distinct values in attribute $R_h.attr_h^l$ of R' is estimated as $d_h^l \cdot r_h'$, $h = 1, 2$. Let n_3 be $C' \cdot \beta_1 \cdot \beta_2 \cdot T_1' \cdot T_2'$. The time for processing $\sigma_{p_1^{i+1} \to p_1^{i+2} \to \ldots \to p_1^m \to p_2^{j+1} \to p_2^{j+2} \to \ldots \to p_2^n} R'$ is dominated by the scan time for R', the time for accessing the cache, and the time for performing functions, which is evaluated as

$$g_3 \cdot n_3 + r_1' \cdot \sum_{k=i+1}^{m} (\lambda_1^k \cdot d_1^k \cdot \Pi_{h=i+1}^{k-1}(\rho_1^h)) + n_3 \cdot t_c \cdot \sum_{k=i+1}^{m} (\Pi_{h=i+1}^{k-1}(\rho_1^h)) + r_2' \cdot z' \cdot \sum_{k=j+1}^{n} (\lambda_2^k \cdot d_2^k \cdot \Pi_{h=j+1}^{k-1}(\rho_2^h)) + n_3 \cdot t_c \cdot z' \cdot \sum_{k=j+1}^{n} (\Pi_{h=j+1}^{k-1}(\rho_2^h)),$$

where z' is $\Pi_{h=i+1}^m(\rho_1^h)$. Note that $\Pi_{h=v_1}^{v_2} x^h = 1$ if $v_2 < v_1$.

3.3 Problem Formulation

In traditional query optimization methods, selections are considered zero-time operations and executed in an arbitrary order before as many joins as possible. Reconsider the query $\sigma_{p_1 \wedge p_2 \wedge \ldots \wedge p_k}(R_1 \bowtie_{p'} R_2)$. By the traditional optimization technique which neglects the cost of the selection, the query will be executed by the plan

$$(\sigma_{p_1^1 \to p_1^2 \to \ldots \to p_1^m} R_1) \bowtie_{p'} (\sigma_{p_2^1 \to p_2^2 \to \ldots \to p_2^n} R_2).$$

However, when the foreign function is involved in the selection predicate, executing all the selection predicates before the join may not be an optimal execution plan. Let F_1 and F_2 be sets $\{p_1^1, p_1^2, \ldots, p_1^m\}$ and $\{p_2^1, p_2^2, \ldots, p_2^n\}$, respectively. Our goal is to find an execution sequence S_1 for selection predicates in a subset B_1 of F_1, an execution sequence S_2 for selection predicates in a subset B_2 of F_2, and an execution sequence S_3 for selection predicates in B_3, where B_3 is $(F_1 - B_1) \cup (F_2 - B_2)$, such that the response time of processing $\sigma_{S_3}(\sigma_{S_1} R_1 \bowtie_{p'} \sigma_{S_2} R_2)$ is minimized.

Assume S_1, S_2, and S_3 are $p^{u_1} \to p^{u_2} \to \ldots \to p^{u_i}$, $p^{v_1} \to p^{v_2} \to \ldots \to p^{v_j}$, and $p^{w_1} \to p^{w_2} \to \ldots \to p^{w_l}$, respectively, where $0 \le i \le m, 0 \le j \le n$, and $l = m + n - i - j$. Let C^{u_h}, C^{v_h}, and C^{w_h} be the total costs of function calls for distinct values in the associated attributes (in the base relation) of predicates p^{u_h}, p^{v_h}, and p^{w_h}, respectively. C^{u_h} is $\lambda^{u_h} \cdot d^{u_h}, 1 \le h \le i$, and C^{v_h} is $\lambda^{v_h} \cdot d^{v_h}, 1 \le h \le j$. (Note that if p^{u_h} is p_1^k, where $1 \le k \le m$, then λ^{u_h} and d^{u_h} represent λ_1^k and d_1^k, respectively. Similarly, if p^{v_h} is p_2^k, where $1 \le k \le n$, then λ^{v_h} and d^{v_h} represent λ_2^k and d_2^k, respectively.) For each predicate $p^{w_h}, 1 \le h \le l$,

$$C^{w_h} = \begin{cases} \lambda_1^a \cdot d_1^a & \text{if } p^{w_h} \text{ is } p_1^a \text{ in } F_1 - B_1 \\ \lambda_2^b \cdot d_2^b & \text{if } p^{w_h} \text{ is } p_2^b \text{ in } F_2 - B_2 \end{cases}$$

and the selectivity

$$\rho^{w_h} = \begin{cases} \rho_1^a & \text{if } p^{w_h} \text{ is } p_1^a \text{ in } F_1 - B_1 \\ \rho_2^b & \text{if } p^{w_h} \text{ is } p_2^b \text{ in } F_2 - B_2 \end{cases}$$

Let R_1' and R_2' be the resultant relations of processing $\sigma_{S_1} R_1$ and $\sigma_{S_2} R_2$, respectively, and T_1 and T_2 be the cardinalities of R_1' and R_2', respectively. T_1 is $n_1 \cdot \Pi_{h=1}^i(\rho^{u_h})$ and T_2 is $n_2 \cdot \Pi_{h=1}^j(\rho^{v_h})$. First, consider the time H_1 for processing $\sigma_{S_1} R_1$ and transmitting the reduced relation R_1' to the final site:

$$H_1 = g_1 \cdot n_1 + (C^{u_1} + n_1 \cdot t_c) + \rho^{u_1} \cdot (C^{u_2} + n_1 \cdot t_c) + \ldots + \Pi_{h=1}^{i-1}(\rho^{u_h}) \cdot (C^{u_i} + n_1 \cdot t_c) + T_1 \cdot b_1 \cdot X_1 + E_1.$$

Similarly, the time H_2 for processing $\sigma_{S_2} R_2$ and transmitting the reduced relation R_2' to the final site is

$$H_2 = g_2 \cdot n_2 + (C^{v_1} + n_2 \cdot t_c) + \rho^{v_1} \cdot (C^{v_2} + n_2 \cdot t_c) + \ldots + \Pi_{h=1}^{j-1}(\rho^{v_h}) \cdot (C^{v_j} + n_2 \cdot t_c) + T_2 \cdot b_2 \cdot X_2 + E_2.$$

Since $\sigma_{S_1}R_1$ and $\sigma_{S_2}R_2$ can be locally processed and transmitted in parallel, the time for locally processing relations and transmitting the reduced relations is $max(H_1, H_2)$. After all the reduced relations arrive at the final site, the join can be processed. The join processing time is $D \cdot T_1 \cdot T_2$, where D is a constant. Finally, we consider the final selection. Let R' be the result of processing $\sigma_{S_1}R_1 \bowtie_{p'} \sigma_{S_2}R_2$, and r_h be the cumulative reduction factor of the number of distinct values in an attribute of R_h after the join processing, where $h = 1, 2$. r_1 and r_2 are evaluated to be $\Pi_{h=1}^{i}(\rho^{u_h}) \cdot \beta_1$ and $\Pi_{h=1}^{j}(\rho^{v_h}) \cdot \beta_2$, respectively. For each predicate $p^{w_h}, 1 \le h \le l$, the cumulative reduction factor of the number of distinct values for the associated attribute is

$$r^{w_h} = \begin{cases} r_1 & \text{if } p^{w_h} \text{ is } p_1^a \text{ in } F_1 - B_1 \\ r_2 & \text{if } p^{w_h} \text{ is } p_2^b \text{ in } F_2 - B_2. \end{cases}$$

The cardinality of R' is estimated to be $C \cdot \beta_1 \cdot \beta_2 \cdot T_1 \cdot T_2$ denoted by n_3, where C is a constant, and the time for processing the final selection $\sigma_{S_3}R'$ is evaluated as

$$H_3 = g_3 \cdot n_3 + (C^{w_1} \cdot r^{w_1} + n_3 \cdot t_c) + \rho^{w_1} \cdot (C^{w_2} \cdot r^{w_2} + n_3 \cdot t_c) + \ldots + \Pi_{h=1}^{l-1}(\rho^{w_h}) \cdot (C^{w_l} \cdot r^{w_l} + n_3 \cdot t_c).$$

Summarizing the above discussion, the total response time for processing $\sigma_{S_3}(\sigma_{S_1}R_1 \bowtie_{P'} \sigma_{S_2}R_2)$ is

$$max(H_1, H_2) + D \cdot T_1 \cdot T_2 + H_3.$$

We define the problem of minimizing response time for processing the query $\sigma_{p_1 \wedge p_2 \wedge \ldots \wedge p_k}(R_1 \bowtie_{p'} R_2)$ as follows:

Definition 1 (*MRT*). *Given*

$$P = (\beta_1, \beta_2, C, D, g_3, t_c, (E_1, X_1, g_1, n_1, b_1, \{(\rho_1^h, \lambda_1^h, d_1^h) \mid 1 \le h \le m\}), (E_2, X_2, g_2, n_2, b_2, \{(\rho_2^h, \lambda_2^h, d_2^h) \mid 1 \le h \le n, \}))),$$

where all the values are positive rational numbers $0 \le \beta_1, \beta_2 \le 1$ *and* $0 \le \rho_i^h \le 1$, *find the optimal sequences* S_1, S_2, *and* S_3 *such that* $RT(S_1, S_2, S_3)$ *is minimized, where* $RT(S_1, S_2, S_3)$ *denotes* $max(H_1, H_2) + D \cdot T_1 \cdot T_2 + H_3$.

3.4 The Property of MRT

It can be seen that if the numbers of predicates in B_1, B_2, and B_3 are i, j, and $m + n - i - j$, respectively, then the number of possible execution plans for minimizing $RT(S_1, S_2, S_3)$ is $i! \times j! \times (m + n - i - j)!$. Since i ranges from 0 to m and j from 0 to n, the total number of possible execution plans for the query $\sigma_{p_1 \wedge p_2 \wedge \ldots \wedge p_k}(R_1 \bowtie_{p'} R_2)$ is

$$\sum_{i=0}^{m}\sum_{j=0}^{n}(\binom{m}{i} \times \binom{n}{j} \times i! \times j! \times (m + n - i - j)!).$$

In the following, a property is derived for reducing the search space for finding an optimal query execution plan. Assume B_1, B_2, and B_3 are $\{p^{u_1}, p^{u_2}, \ldots, p^{u_i}\}$, $\{p^{v_1}, p^{v_2}, \ldots, p^{v_j}\}$, and $\{p^{w_1}, p^{w_2}, \ldots, p^{w_l}\}$, respectively, and S_1, S_2, and S_3 are $p^{u_1} \to p^{u_2} \to \ldots \to p^{u_i}$, $p^{v_1} \to p^{v_2} \to \ldots \to p^{v_j}$, and

$$p^{w_1} \to p^{w_2} \to \ldots \to p^{w_l},$$

respectively.

Lemma 1. *Given* B_1, B_2, *and* B_3, S_1, S_2, *and* S_3 *are optimal if and only if*

$$\frac{C^{u_h} + n_1 \cdot t_c}{1 - \rho^{u_h}} \le \frac{C^{u_{h+1}} + n_1 \cdot t_c}{1 - \rho^{u_{h+1}}},\ 1 \le h \le i - 1,$$

$$\frac{C^{v_h} + n_2 \cdot t_c}{1 - \rho^{v_h}} \le \frac{C^{v_{h+1}} + n_2 \cdot t_c}{1 - \rho^{v_{h+1}}},\ 1 \le h \le j - 1,$$

and

$$\frac{C^{w_h} \cdot r^{w_h} + n_3 \cdot t_c}{1 - \rho^{w_h}} \le \frac{C^{w_{h+1}} \cdot r^{w_{h+1}} + n_3 \cdot t_c}{1 - \rho^{w_{h+1}}},\ 1 \le h \le l - 1.$$

Proof. Consider the selection sequence $S_1 : p^{u_1} \to p^{u_2} \to \ldots \to p^{u_d} \to p^{u_{d+1}} \to \ldots \to p^{u_i}$. The cost for processing $\sigma_{S_1}R_1$ is

$$Cost(S_1) = g_1 \cdot n_1 + (C^{u_1} + n_1 \cdot t_c) + \rho^{u_1} \cdot (C^{u_2} + n_1 \cdot t_c) + \ldots + \Pi_{h=1}^{d-1}(\rho^{u_h}) \cdot (C^{u_d} + n_1 \cdot t_c) + \Pi_{h=1}^{d}(\rho^{u_h}) \cdot (C^{u_{d+1}} + n_1 \cdot t_c) + \ldots + \Pi_{h=1}^{i-1}(\rho^{u_h}) \cdot (C^{u_i} + n_1 \cdot t_c).$$

For the selection sequence

$$S_1' : p^{u_1} \to p^{u_2} \to \ldots \to p^{u_{d-1}} \to p^{u_{d+1}} \to p^{u_d} \to p^{u_{d+2}} \to \ldots \to p^{u_i},$$

the cost for processing $\sigma_{S_1'}R_1$ is

$$Cost(S_1') = g_1 \cdot n_1 + (C^{u_1} + n_1 \cdot t_c) + \rho^{u_1} \cdot (C^{u_2} + n_1 \cdot t_c) + \ldots + \Pi_{h=1}^{d-1}(\rho^{u_h}) \cdot (C^{u_{d+1}} + n_1 \cdot t_c) + \Pi_{h=1}^{d-1}(\rho^{u_h}) \cdot \rho^{u_{d+1}} \cdot (C^{u_d} + n_1 \cdot t_c) + \ldots + \Pi_{h=1}^{i-1}(\rho^{u_h}) \cdot (C^{u_i} + n_1 \cdot t_c).$$

We have

$$\frac{Cost(S_1) - Cost(S_1')}{\Pi_{h=1}^{d-1}(\rho^{u_h})} = (C^{u_d} + n_1 \cdot t_c) \cdot (1 - \rho^{u_{d+1}}) - (C^{u_{d+1}} + n_1 \cdot t_c) \cdot (1 - \rho^{u_d})$$

$$Cost(S_1) - Cost(S_1') \le 0 \text{ if and only if } \frac{C^{u_d} + n_1 \cdot t_c}{1 - \rho^{u_d}} \le \frac{C^{u_{d+1}} + n_1 \cdot t_c}{1 - \rho^{u_{d+1}}}.$$

Therefore, S_1 is optimal *if and only if*

$$\frac{C^{u_h} + n_1 \cdot t_c}{1 - \rho^{u_h}} \le \frac{C^{u_{h+1}} + n_1 \cdot t_c}{1 - \rho^{u_{h+1}}},\ 1 \le h \le i - 1.$$

Similarly, we can prove that S_2 is optimal *if and only if*

$$\frac{C^{v_h} + n_2 \cdot t_c}{1 - \rho^{v_h}} \le \frac{C^{v_{h+1}} + n_2 \cdot t_c}{1 - \rho^{v_{h+1}}},\ 1 \le h \le j - 1,$$

and S_3 is optimal *if and only if*

$$\frac{C^{w_h} \cdot r^{w_h} + n_3 \cdot t_c}{1 - \rho^{w_h}} \le \frac{C^{w_{h+1}} \cdot r^{w_{h+1}} + n_3 \cdot t_c}{1 - \rho^{w_{h+1}}},\ 1 \le h \le l - 1.$$

□

Definition 2. *For each selection predicate p^{u_h} in B_1, $1 \le h \le i$, the* rank *of p^{u_h} is defined as*

$$rank_1(p^{u_h}) = \frac{C^{u_h} + n_1 \cdot t_c}{1 - \rho^{u_h}}.$$

For each selection predicate p^{v_h} in B_2, $1 \le h \le j$, the rank *of p^{v_h} is defined as*

$$rank_2(p^{v_h}) = \frac{C^{v_h} + n_2 \cdot t_c}{1 - \rho^{v_h}}.$$

And, for each selection predicate p^{w_h} in B_3, $1 \le h \le l$, the rank *of p^{w_h} is defined as*

$$rank_3(p^{w_h}) = \frac{C^{w_h} \cdot r^{w_h} + n_3 \cdot t_c}{1 - \rho^{w_h}}.$$

Since i ranges from 0 to m and j from 0 to n, there are

$$\sum_{i=0}^{m}\sum_{j=0}^{n}\left(\binom{m}{i} \times \binom{n}{j}\right)$$

*combinations for B_1, B_2, and B_3. By **Lemma 1**, we can obtain optimal sequences S_1, S_2, and S_3 by sorting ranks of predicates in B_1, B_2, and B_3, respectively, in an increasing order. Therefore, the search space for finding an optimal execution plan of the query $\sigma_{p_1 \wedge p_2 \wedge \ldots \wedge p_k}(R_1 \bowtie_{p'} R_2)$ can be reduced to*

$$\sum_{i=0}^{m}\sum_{j=0}^{n}\left(\binom{m}{i} \times \binom{n}{j}\right).$$

Theorem 1. *The MRT problem is NP-hard.*

Proof. We prove the NP-hardness of the problem via the knapsack problem which has been shown to be NP-complete in [15]. Given a finite set $A = \{a_1, a_2, \ldots, a_k\}$, a size $s(a_i) \in Z^+$ and a value $v(a_i) \in Z^+$ for each $a_i \in A$, and two positive integers B and L, the knapsack problem answers whether there is a subset $A' \subseteq A$ such that $\sum_{a \in A'} s(a) \le B$ and $\sum_{a \in A'} v(a) \ge L$. Let us consider a special case of MRT as follows: Assume there is no related selection predicate for R_2 in the query $\sigma_{p_1 \wedge p_2 \wedge \ldots \wedge p_k}(R_1 \bowtie_{p'} R_2)$ and the site where R_1 is located is considered to be the final site. Let $p_1^1, p_1^2, \ldots, p_1^k$ be $p_1, p_2, \ldots, p_k$, respectively, and F' the set $\{p_1^1, p_1^2, \ldots, p_1^k\}$. Some assumptions are made as follows:

- All the selectivities $\rho_1^1, \rho_1^2, \ldots, \rho_1^k$ are one.
- All the numbers of distinct values $d_1^1, d_1^2, \ldots, d_1^k$ are x.
- Both the paremeters r_1 and r_2 are y, which is less than 1.
- Assume the cache resides in main memory. The parameter t_c is zero.

In this case, our purpose is to find execution sequences Seq and Seq' for selection predicates in subsets B' of F' and $F' - B'$, respectively, such that $RT(Seq, Seq')$ is minimized. Assume Seq and Seq' are $p_1^{u_1} \to p_1^{u_2} \to \ldots \to p_1^{u_i}$ and $p_1^{u_{i+1}} \to p_1^{u_{i+2}} \to \ldots \to p_1^{u_k}$, respectively. $RT(Seq, Seq')$ denotes

$$\begin{aligned} max\{&g_1 \cdot n_1 + (\lambda_1^{u_1} \cdot x + \lambda_1^{u_2} \cdot x + \ldots + \lambda_1^{u_i} \cdot x), \\ &n_2 \cdot b_2 \cdot X_2 + E_2\} + D \cdot n_1 \cdot n_2 + g_3 \cdot C \cdot n_1 \cdot n_2 \\ &+ (\lambda_1^{u_{i+1}} \cdot y + \lambda_1^{u_{i+2}} \cdot y + \ldots + \lambda_1^{u_k} \cdot y). \end{aligned}$$

To simplify the expression, let

1. $n_3 = C \cdot n_1 \cdot n_2$,
2. $G = g_1 \cdot n_1$,
3. $M = n_2 \cdot b_2 \cdot X_2 + E_2$, and
4. $Q = D \cdot n_1 \cdot n_2$.

Then,

$$\begin{aligned} SRT(Seq, Seq') = max\{&G + x \cdot (\lambda_1^{u_1} + \lambda_1^{u_2} + \ldots + \lambda_1^{u_i}), M\} \\ &+ Q + g_3 \cdot n_3 + y \cdot (\lambda_1^{u_{i+1}} + \lambda_1^{u_{i+2}} + \ldots + \lambda_1^{u_k}). \end{aligned}$$

For any given instance of the knapsack problem, the transformation to the above special case is as follows:

$$\begin{aligned} s(a_j) = \lambda_1^j \text{ and } v(a_j) &= \lambda_1^j \text{ for } 1 \le j \le k \\ B = L &= (M - G)/x. \end{aligned}$$

□

It is easy to show that there exists a solution to the transformed instance of the MRT problem if and only if there is a solution to the knapsack problem. Since the transformation can be done in polynomial time, the MRT problem is NP-hard.

4 Solving a Special Case of MRT

In this section, we propose an optimal algorithm with polynomial complexity to solve a special case of MRT.

Consider a special case of MRT as follows: There is no related selection predicate for R_2 in the query

$$\sigma_{p_1 \wedge p_2 \wedge \ldots \wedge p_k}(R_1 \bowtie_{p'} R_2)$$

and the site where R_2 is located is considered to be the final site. Let $p_1^1, p_1^2, \ldots, p_1^k$ be $p_1, p_2, \ldots, p_k$, respectively, and F' the set $\{p_1^1, p_1^2, \ldots, p_1^k\}$. In this case, our purpose is to find execution sequences Seq and Seq' for selection predicates in subsets B' of F' and $F' - B'$, respectively, such that the response time of processing $\sigma_{Seq'}(\sigma_{Seq} R_1 \bowtie_{p'} R_2)$ is minimized.

Assume B' is $\{p^{u_1}, p^{u_2}, \ldots, p^{u_i}\}$, and Seq and Seq' are

$$p^{u_1} \to p^{u_2} \to \ldots \to p^{u_i}$$

and

$$p^{u_{i+1}} \to p^{u_{i+2}} \to \ldots \to p^{u_k},$$

respectively. Let C^{u_h} be $\lambda^{u_h} \cdot d^{u_h}, 1 \le h \le k$, and R' be the resultant relation of processing $\sigma_{Seq} R_1$. The cardinality of R' is estimated to be $n_1 \cdot \Pi_{h=1}^{i}(\rho^{u_h})$ denoted by T'. The time H' for processing $\sigma_{Seq} R_1$ and transmitting the reduced relation R' to the final site is

$$H' = g_1 \cdot n_1 + (C^{u_1} + n_1 \cdot t_c) + \rho^{u_1} \cdot (C^{u_2} + n_1 \cdot t_c) + \ldots + \Pi_{h=1}^{i-1}(\rho^{u_h}) \cdot (C^{u_i} + n_1 \cdot t_c) + T' \cdot b_1 \cdot X_1 + E_1.$$

After R' arrives at the final site, the join is processed. The join processing time is $D' \cdot T' \cdot n_2$, where D' is a constant. Let R'' and r' be the result of $(\sigma_{Seq} R_1 \bowtie_{p'} R_2)$ and the cumulative reduction factor of the number of distinct values in an attribute of R_1 after the join processing, respectively. The cardinality of R'' is estimated to be $C' \cdot \beta_1 \cdot \beta_2 \cdot T' \cdot n_2$ denoted by n_3, where C' is a constant, and r' is $\beta_1 \cdot \Pi_{h=1}^{i}(\rho^{u_h})$. Therefore, the time for processing final selection $\sigma_{Seq'} R''$ is

$$L' = g_3 \cdot n_3 + (C^{u_{i+1}} \cdot r' + n_3 \cdot t_c) + \rho^{u_{i+1}} \cdot (C^{u_{i+2}} \cdot r' + n_3 \cdot t_c) + \ldots + \Pi_{h=i+1}^{k-1}(\rho^{u_h}) \cdot (C^{u_k} \cdot r' + n_3 \cdot t_c).$$

We define the problem of minimizing query response time for the special case as follows:

Definition 3 (SMRT). *Given*

$$P = (\beta_1, \beta_2, C', D', n_2, g_3, t_c, (E_1, X_1, g_1, n_1, b_1, \{(\rho_1^h, \lambda_1^h, d_1^h) \mid 1 \le h \le k\})),$$

where all the values are positive rational numbers, $0 \le \beta_1, \beta_2 \le 1$ *and* $0 \le \rho_1^h \le 1$, *find the optimal sequences Seq and Seq′ such that* $SRT(Seq, Seq')$ *is minimized, where* $SRT(Seq, Seq')$ *denotes* $H' + D' \cdot T' \cdot n_2 + L'$.

To simplify the expression, let

1. $G = g_1 \cdot n_1 + E_1$,
2. $B = \beta_1$,
3. $Q = C' \cdot \beta_1 \cdot \beta_2 \cdot n_1 \cdot n_2$, *and*
4. $Y = n_1 \cdot b_1 \cdot X_1 + D' \cdot n_1 \cdot n_2 + g_3 \cdot Q$. *Then,*

$$\begin{aligned} SRT(Seq, Seq') = G &+ (C^{u_1} + n_1 \cdot t_c) + \rho^{u_1} \cdot (C^{u_2} + n_1 \cdot t_c) + \ldots + \Pi_{h=1}^{i-1}(\rho^{u_h}) \cdot (C^{u_i} + n_1 \cdot t_c) \\ &+ Y \cdot \Pi_{h=1}^{i}(\rho^{u_h}) + B \cdot \Pi_{h=1}^{i}(\rho^{u_h})(C^{u_{i+1}} + \rho^{u_{i+1}} \cdot C^{u_{i+2}} + \ldots + \Pi_{h=i+1}^{k-1}(\rho^{u_h}) \cdot C^{u_k}) + Q \cdot t_c \cdot \Pi_{h=1}^{i}(\rho^{u_h}) \cdot (1 + \rho^{u_{i+1}} + \ldots + \Pi_{h=i+1}^{k-1}(\rho^{u_h})). \end{aligned}$$

Lemma 2. *Given B′, Seq, and Seq′ are optimal if and only if*

$$rank_1(p^{u_h}) \left(= \frac{C^{u_h} + n_1 \cdot t_c}{1 - \rho^{u_h}}\right) \le rank_1(p^{u_{h+1}}) \left(= \frac{C^{u_{h+1}} + n_1 \cdot t_c}{1 - \rho^{u_{h+1}}}\right), \quad 1 \le h \le i - 1$$

and

$$rank_3(p^{u_h}) \left(= \frac{C^{u_h} \cdot r' + n_3 \cdot t_c}{1 - \rho^{u_h}}\right) \le rank_3(p^{u_{h+1}}) \left(= \frac{C^{u_{h+1}} \cdot r' + n_3 \cdot t_c}{1 - \rho^{u_{h+1}}}\right), \quad i + 1 \le h \le k - 1.$$

The proof is similar to that for **Lemma 1** and is omitted.

Lemma 3. $SRT(Seq, Seq')$ *is minimal if and only if*

$$Y \cdot (1 - \rho^{u_h}) - C^{u_h} \cdot (1 - B) \ge t_c \cdot (n_1 - Q), \ 1 \le h \le i, \quad (1)$$

and

$$Y \cdot (1 - \rho^{u_h}) - C^{u_h} \cdot (1 - B) < t_c \cdot (n_1 - Q), \ i + 1 \le h \le k, \quad (2)$$

$$rank_1(p^{u_h}) \le rank_1(p^{u_{h+1}}), \ 1 \le h \le i - 1, \quad (3)$$

and

$$rank_3(p^{u_h}) \le rank_3(p^{u_{h+1}}), \ i + 1 \le h \le k - 1. \quad (4)$$

Proof. ($\Rightarrow$) By **Lemma 2**, if $SRT(Seq, Seq')$ is minimal, then conditions (3) and (4) are satisfied. Assume $SRT(Seq, Seq')$ is minimal but does not satisfy conditions (1) and (2). Consider the following cases:

- Case 1. Assume

  $$Y \cdot (1 - \rho^{u_i}) - C^{u_i} \cdot (1 - B) < t_c \cdot (n_1 - Q).$$

  Let S and S' be

  $$p^{u_1} \to p^{u_2} \to \ldots \to p^{u_{i-1}}$$

  and $p^{u_i} \to p^{u_{i+1}} \to \ldots \to p^{u_k}$, respectively. Since $SRT(Seq, Seq')$ is minimal,

  $$SRT(S, S') - SRT(Seq, Seq') \ge 0.$$

  We get

  $$Y \cdot (1 - \rho^{u_i}) - C^{u_i} \cdot (1 - B) \ge t_c \cdot (n_1 - Q) \ldots contradiction.$$

- Case 2. Assume

  $$Y \cdot (1 - \rho^{u_{i+1}}) - C^{u_{i+1}} \cdot (1 - B) > t_c \cdot (n_1 - Q).$$

  Let S and S' be

  $$p^{u_1} \to p^{u_2} \to \ldots \to p^{u_{i+1}}$$

  and $p^{u_{i+2}} \to p^{u_{i+3}} \to \ldots \to p^{u_k}$, respectively. Since $SRT(Seq, Seq')$ is minimal,

  $$SRT(S, S') - SRT(Seq, Seq') \ge 0.$$

  We get

  $$Y \cdot (1 - \rho^{u_{i+1}}) - C^{u_{i+1}} \cdot (1 - B) \le t_c \cdot (n_1 - Q) \ldots contradiction.$$

From Case 1 and Case 2, we claim that if $SRT(Seq, Seq')$ is minimal, then conditions (1) and (2) must be satisfied.

($\Leftarrow$) Assume ST and ST' are sequences which satisfy conditions (3) and (4), but do not satisfy conditions (1) and (2). From the discussion in Case 1 and Case 2, it shows that we can always find sequences Seq and Seq' satisfying conditions (1), (2), (3), and (4) such that $SRT(Seq, Seq') < SRT(ST, ST')$. Therefore, if sequences Seq and Seq' satisfy conditions (1), (2), (3), and (4), then $SRT(Seq, Seq')$ is minimal. □

In the following, based on **Lemma 3** Algorithm **S** is proposed to efficiently solve the *SMRT* problem.

Algorithm S

Input($C', B, \beta_2, Q, Y, n_1, n_2, t_c, \{(\rho_1^h, \lambda_1^h, d_1^h) \mid 1 \le h \le k\}$)
(We use p_1^h to denote the predicate with the associated values $(\rho_1^h, \lambda_1^h, d_1^h)$)
Step 1. For each predicate p_1^h, compute the value $V_h = Y \cdot (1 - \rho_1^h) - \lambda_1^h \cdot d_1^h \cdot (1 - B)$;
$U_1 \leftarrow \emptyset$; /* the set of selection predicates in Seq */
$U_2 \leftarrow \emptyset$; /* the set of selection predicates in Seq' */
$M \leftarrow t_c \cdot (n_1 - Q)$;
$i_1 \leftarrow 0$; /* the number of selection predicates in U_1 */
$i_2 \leftarrow 0$; /* the number of selection predicates in U_2 */
Step 2. **for** $h \leftarrow 1$ **to** k
begin
if $V_h \ge M$ **then**
$U_1 \leftarrow U_1 \cup p_1^h$;
$i_1 \leftarrow i_1 + 1$;
else
$U_2 \leftarrow U_2 \cup p_1^h$;
$i_2 \leftarrow i_2 + 1$;
end {**if**}
end {**for**}
Compute the rank for each predicate in $U_1\Big(rank_1(p_1^h) = \frac{\lambda_1^h \cdot d_1^h + n_1 \cdot t_c}{1 - \rho_1^h}\Big)$;

Derive the sorting function $f()$ such that the predicates in U_1 satisfy

$rank_1(p_1^{f(1)}) \le rank_1(p_1^{f(2)}) \le \ldots \le rank_1(p_1^{f(i_1)})$;

Let Seq be $p_1^{f(1)} \rightarrow p_1^{f(2)} \rightarrow \ldots \rightarrow p_1^{f(i_1)}$;

$r' \leftarrow B \cdot \Pi_{h=1}^{i_1}(\rho_1^{f(h)})$;
$n_3 \leftarrow C' \cdot B \cdot \beta_2 \cdot n_1 \cdot n_2 \cdot \Pi_{h=1}^{i_1}(\rho_1^{f(h)})$;
Compute the rank for each predicate

in $U_2\Big(rank_3(p_1^h) = \frac{\lambda_1^h \cdot d_1^h \cdot r' + n_3 \cdot t_c}{1 - \rho_1^h}\Big)$;

Derive the sorting function $g()$ such that the predicates in U_2 satisfy

$rank_3(p_1^{g(1)}) \le rank_3(p_1^{g(2)}) \le \ldots \le rank_3(p_1^{g(i_2)})$;

Let Seq' be $p_1^{g(1)} \rightarrow p_1^{g(2)} \rightarrow \ldots \rightarrow p_1^{g(i_2)}$;
Step 3. Output(Seq, Seq')
END

Let $f^{-1}()$ be the inverse function of $f()$. $f^{-1}(x) = y$ means $p_1^{f(y)} = p_1^x$. In Step 1, the execution time is $O(k)$. The time needed for Step 2 is dominated by sorting the ranks. In the worst case, it takes $O(k \log k)$ time. Therefore, the overall time complexity of Algorithm **S** is $O(k \log k)$.

5 Heuristic Algorithms

5.1 A Heuristic Algorithm for MRT

Since the MRT problem is NP-hard, we introduce a heuristic algorithm for it in this section.

There is a trade-off between reducing the cost of function calls and reducing the cost of data transmission and join processing. For a selection predicate with a foreign function, if it is evaluated after join processing, the cost of data transmission and join processing will increase since the size of the relation to be joined is not fully reduced before data transmission, but the cost of function calls needed for the selection evaluation can be reduced since the number of distinct values in the associated attribute decreases after join processing. Therefore, if the selection predicate with high C_i^h and ρ_i^h is delayed and evaluated after join processing with a great possibility, the benefit obtained by reducing the cost of function calls may outweigh the extra overhead needed for data transmission and join processing.

Observe that the rank of the predicate $p_i^h \cdot rank_1(p_1^h)$ is $\frac{C_1^h + n_1 \cdot t_c}{1 - \rho_1^h}$ and $rank_2(p_2^h)$ is $\frac{C_2^h + n_2 \cdot t_c}{1 - \rho_2^h}$. If both C_i^h and ρ_i^h are high, then the rank of p_i^h is high. The higher C_i^h is, the higher the cost of function calls is; moreover, the higher the selectivity ρ_i^h is, the lower the reduction of the relation size by the selection is. Intuitively, evaluating selection predicates with higher ranks after join processing is prone to reduce the query processing cost. Based on the thinking and **Lemma 1**, a heuristic algorithm is developed for the MRT problem to obtain a suboptimal solution.

Reconsider the query

$$\sigma_{p_1 \wedge p_2 \wedge \ldots \wedge p_k}(R_1 \bowtie_{p'} R_2).$$

Let $C(S_1, S_2, S_3)$ be the cost of processing

$$\sigma_{S_3}(\sigma_{S_1} R_1 \bowtie_{p'} \sigma_{S_2} R_2).$$

$C(S_1, S_2, S_3)$ is computed by the response time model described in Section 3. First, the ranks of predicates are computed and sorted in an increasing order. S_1 and S_2 are initialized to be the sequences of related selection predicates for R_1 and R_2, respectively, which satisfy the property of **Lemma 1**. That is, no selection predicates are delayed and executed after the join processing and S_3 is empty. Let *MIN* denote the minimal query processing cost, which has $C(S_1, S_2, S_3)$ as the initial value. Then, we select the predicate with the maximal rank value from S_1 and S_2. The selected predicate is added to B (which is the set of selection predicates in S_3) and deleted from the corresponding sequence S_1 or S_2. S_3 is set to be the sequence of predicates in B, which satisfies the property of **Lemma 1**. Next, the value of $C(S_1, S_2, S_3)$ is updated and compared with *MIN*. If $C(S_1, S_2, S_3) < MIN$, then *MIN* is updated to $C(S_1, S_2, S_3)$ and the present optimal execution sequences S_1, S_2, and S_3 are stored. The procedure repeats until all the selection predicates are considered to be evaluated after join processing.

The heuristic algorithm is described as follows:

Algorithm H

Input($\beta_1, \beta_2, C, D, g_3, t_c, (E_1, X_1, g_1, n_1, b_1, \{(\rho_1^h, \lambda_1^h, d_1^h) \mid 1 \le h \le m\})$,

$(E_2, X_2, g_2, n_2, b_2, \{(\rho_2^h, \lambda_2^h, d_2^h) \mid 1 \le h \le n\}))$
(We use p_i^h to denote the predicate with the associated values $(\rho_i^h, \lambda_i^h, d_i^h)$)
Step 1. Compute the rank for each predicate

$(rank_1(p_1^h) = \frac{\lambda_1^h \cdot d_1^h + n_1 \cdot t_c}{1-\rho_1^h}$ and $rank_2(p_2^h) = \frac{\lambda_2^h \cdot d_2^h + n_2 \cdot t_c}{1-\rho_2^h})$;

Derive sorting functions $k_1()$ and $k_2()$ such that
$rank_1(p_1^{k_1(1)}) \le rank_1(p_1^{k_1(2)}) \le \ldots$

$\le rank_1(p_1^{k_1(m)})$ and

$rank_2(p_2^{k_2(1)}) \le rank_2(p_2^{k_2(2)}) \le \ldots \le rank_2(p_2^{k_2(n)})$;

Let S_1, S_2, and S_3 be $p_1^{k_1(1)} \rightarrow p_1^{k_1(2)} \rightarrow \ldots \rightarrow p_1^{k_1(m)}$,

$p_2^{k_2(1)} \rightarrow p_2^{k_2(2)} \rightarrow \ldots \rightarrow p_2^{k_2(n)}$
and empty, respectively;
Step 2. $B \leftarrow \emptyset$; /* the set of selection predicates in S_3 */
$b \leftarrow 0$; /* the number of selection predicates in set B */
$MIN \leftarrow C(S_1, S_2, S_3)$; /* the minimal query processing cost */
$(O_1, O_2, O_3) \leftarrow (S_1, S_2, S_3)$; /* the present optimal execution sequences */
$t_1 \leftarrow m$; /* the number of selection predicates in S_1 */
$t_2 \leftarrow n$; /* the number of selection predicates in S_2 */
Step 3. **for** $h \leftarrow 1$ **to** $m+n$
begin
if $((t_1 > 0)$ and $(t_2 = 0))$ or $((t_1 > 0)$ and $(t_2 > 0)$ and $(rank_1(p_1^{k_1(t_1)}) \ge rank_2(p_2^{k_2(t_2)}))$ **then**
$B \leftarrow B \cup \{p_1^{k_1(t_1)}\}$;

Delete $p_1^{k_1(t_1)}$ from sequence S_1;
$t_1 \leftarrow t_1 - 1$;
else
$B \leftarrow B \cup \{p_2^{k_2(t_2)}\}$;

Delete $p_2^{k_2(t_2)}$ from sequence S_2;
$t_2 \leftarrow t_2 - 1$;
end {**if**}
$b \leftarrow b + 1$;
$n_3 \leftarrow C \cdot \beta_1 \cdot \beta_2 \cdot n_1 \cdot n_1 \cdot \Pi_{h=1}^{t_1}(\rho_1^{k_1(h)})$

$\cdot \Pi_{h=1}^{t_2}(\rho_2^{k_2(h)})$;
Derive the sorting function $(k_3(), l())$ such that the predicates in B satisfy

$\left(rank_3\left(p_{l(1)}^{k_3(1)}\right) = \frac{\lambda_{l(1)}^{k_3(1)} \cdot d_{l(1)}^{k_3(1)} \cdot r_{l(1)} + n_3 \cdot t_c}{1-\rho_{l(1)}^{k_3(1)}}\right) \le$

$\left(rank_3\left(p_{l(2)}^{k_3(2)}\right) = \frac{\lambda_{l(2)}^{k_3(2)} \cdot d_{l(2)}^{k_3(2)} \cdot r_{l(2)} + n_3 \cdot t_c}{1-\rho_{l(2)}^{k_3(2)}}\right) \le \ldots$

$\le \left(rank_3\left(p_{l(b)}^{k_3(b)}\right) = \frac{\lambda_{l(b)}^{k_3(b)} \cdot d_{l(b)}^{k_3(b)} \cdot r_{l(b)} + n_3 \cdot t_c}{1-\rho_{l(b)}^{k_3(b)}}\right)$;

/* $r_{l(i)}$ is computed by the definition in Section 3.3, where $l(i) = 1$ or 2 */

Let S_3 be $p_{l(1)}^{k_3(1)} \rightarrow p_{l(2)}^{k_3(2)} \rightarrow \ldots \rightarrow p_{l(b)}^{k_3(b)}$;

if $C(S_1, S_2, S_3) < MIN$ **then**
$MIN \leftarrow C(S_1, S_2, S_3)$;
$(O_1, O_2, O_3) \leftarrow (S_1, S_2, S_3)$;
end {**if**}
end {**for**}
Output (O_1, O_2, O_3, MIN)
END

In Step 1, the execution time is dominated by sorting the ranks of predicates. If $m > n$, then Step 1 takes $O(m \log m)$ time; otherwise, it takes $O(n \log n)$ time. The time needed for Step 2 is $O(1)$. In Step 3, the "for" loop iterates $m+n$ times and, for each iteration, the execution time is dominated by sorting the ranks of predicates in B. Since there are i elements in B for the ith iteration, Step 3 takes $O((m+n)^2 \log(m+n))$ time. The overall time complexity of Algorithm **H** is $O((m+n)^2 \log(m+n))$.

5.2 A Heuristic Algorithm for Processing Queries involving More Than One Join and Selections with Foreign Functions

In the following, we consider the case where a query involves more than one join and selections with foreign functions. The response time model described in Section 3.2 is extended to evaluate query processing cost in this case. The extension is straightforward. Assume there are n relations $R_1, R_2, \ldots, R_n$, and the sets of related selection predicates for $R_1, R_2, \ldots,$ and R_n are $\{p_1^1, p_1^2, \ldots, p_1^{x_1}\}$, $\{p_2^1, p_2^2, \ldots, p_2^{x_2}\}, \ldots,$ and $\{p_n^1, p_n^2, \ldots, p_n^{x_n}\}$, respectively. $R_1, R_2, \ldots,$ and R_n are assumed located at different sites. Let $C(S_1, S_2, \ldots, S_n, S_{n+1})$ be the cost of processing

$$\sigma_{S_{n+1}}(\sigma_{S_1} R_1 \bowtie \sigma_{S_2} R_2 \bowtie \ldots \bowtie \sigma_{S_n} R_n).$$

First, the ranks of predicates are computed and sorted in an increasing order. $S_1, S_2, \ldots,$ and S_n are initialized to be the sequences of related selection predicates for $R_1, R_2, \ldots,$ and R_n, respectively, which satisfy the property of the extension of **Lemma 1** for n relations. That is, no selection predicates are delayed to be executed after join processing and S_{n+1} is empty. Let $B_1, B_2, \ldots,$ and B_{n+1} be the sets of predicates in $S_1, S_2, \ldots,$ and S_{n+1}, respectively, and *MIN* be the minimal query processing cost, which has $C(S_1, S_2, \ldots, S_n, S_{n+1})$ as the initial value. Then, we select the predicate with the maximal rank value in $\bigcup_{1 \le i \le n} B_i$. The selected predicate is added to B_{n+1} and deleted from the corresponding sequence and set. S_{n+1} is set to be the sequence of predicates in B_{n+1}, which satisfies the property of the extension of **Lemma 1**. Next, the value of $C(S_1, S_2, \ldots, S_n, S_{n+1})$ is updated and compared with *MIN*. If

$$C(S_1, S_2, \ldots, S_n, S_{n+1}) < MIN,$$

then MIN is updated to $C(S_1, S_2, \ldots, S_n, S_{n+1})$ and the present optimal execution sequences S_1, $S_2, \ldots,$ and S_{n+1} are stored. The procedure repeats until all the selection predicates are considered to be evaluated after join processing.

The heuristic algorithm is described as follows:

Algorithm GH

Step 1. Compute the rank for each selection predicate

$\left(rank_i\left(p_i^j\right) = \frac{\lambda_i^j \cdot d_i^j + n_i \cdot t_c}{1-\rho_i^j}\right)$.

for $i \leftarrow 1$ **to** n

begin

$B_i \leftarrow \{p_i^1, p_i^2, \ldots, p_i^{x_i}\}$;

Derive sorting function $k_i()$ such that

$$rank_i(p_i^{k_i(1)}) \leq rank_i(p_i^{k_i(2)}) \leq \ldots$$

$$\leq rank_i(p_i^{k_i(x_i)});$$

Let S_i be $p_i^{k_i(1)} \rightarrow p_i^{k_i(2)} \rightarrow \ldots \rightarrow p_i^{k_i(x_i)}$;

/* S_i records the sequence of selection predicates to be evaluated before data transmission */

end {for}

Step 2. $B_{n+1} \leftarrow \emptyset$;

$S_{n+1} \leftarrow \emptyset$;

/* S_{n+1} records the sequence of selection predicates to be evaluated after join processing */

$MIN \leftarrow C(S_1, S_2, ..., S_n, S_{n+1})$; /* the minimal query processing cost */

$(O_1, O_2, \ldots, O_n, O_{n+1}) \leftarrow (S_1, S_2, \ldots, S_n, S_{n+1})$;

/* the present optimal execution sequences */

Step 3. Find a selection predicate p in $\bigcup_{1 \leq i \leq n} B_i$, which has the maximal rank value;

$B_{n+1} \leftarrow B_{n+1} \cup \{p\}$;

Delete p from the corresponding sequence S_i and set B_i;

Step 4. Derive the sort function $(k_{n+1}(), l())$ such that the predicates in B_{n+1} satisfy

$$\frac{\lambda_{l(1)}^{k_{n+1}(1)} \cdot d_{l(1)}^{k_{n+1}(1)} \cdot r_{l(1)} + n_3 \cdot t_c}{1-\rho_{l(1)}^{k_{n+1}(1)}} \leq \frac{\lambda_{l(2)}^{k_{n+1}(2)} \cdot d_{l(2)}^{k_{n+1}(2)} \cdot r_{l(2)} + n_3 \cdot t_c}{1-\rho_{l(2)}^{k_{n+1}(2)}}$$

$$\leq \ldots \leq \frac{\lambda_{l(b)}^{k_{n+1}(b)} \cdot d_{l(b)}^{k_{n+1}(b)} \cdot r_{l(b)} + n_3 \cdot t_c}{1-\rho_{l(b)}^{k_{n+1}(b)}};$$

/* Assume $\mid B_{n+1} \mid = b$. */

/* $r_{l(i)}$ is the cumulative reduction factor */

/* n_3 is the number of tuples in the result of join processing */

Let S_{n+1} be $p_{l(1)}^{k_{n+1}(1)} \rightarrow p_{l(2)}^{k_{n+1}(2)} \rightarrow \ldots \rightarrow p_{l(b)}^{k_{n+1}(b)}$;

if $C(S_1, S_2, \ldots, S_n, S_{n+1}) < MIN$ **then**

$MIN \leftarrow C(S_1, S_2, \ldots, S_n, S_{n+1})$;

$(O_1, O_2, \ldots, O_n, O_{n+1}) \leftarrow (S_1, S_2, \ldots, S_n, S_{n+1})$;

end {if}

TABLE 1
Parameter Values Used in the Simulation

parameter	*value*
n	5
m	5
n_1	1000
n_2	2000
b_1	1000 (bytes)
b_2	800 (bytes)
$E_1 = E_2$	2500 (μsec)
C	1
D	20 (μsec)
$g_1 = g_2 = g_3$	10 (μsec)
t_c	60 (μsec)

Steps (3) and (4) are repeated until all the selection predicates are considered to be evaluated after join processing.

END

Let k be $\sum_{i=1}^{n} x_i$. The complexity of Algorithm **GH** is $O(k^2 \log k)$.

6 Simulation

6.1 Simulation Results for Heuristic Algorithm H

Two programs named **HEU** and **OPT** are designed to implement the heuristic Algorithm **H** and find the optimal solution of *MRT*, respectively. By applying the property of **Lemma 1**, the search space for **OPT** is reduced from

$$\sum_{i=0}^{m}\sum_{j=0}^{n}\left(\binom{m}{i} \times \binom{n}{j} \times i! \times j! \times (m+n-i-j)!\right)$$

to

$$\sum_{i=0}^{m}\sum_{j=0}^{n}\left(\binom{m}{i} \times \binom{n}{j}\right).$$

Both the programs are coded in C and run on the SUN sparc-10 workstation. For each simulation, parameter values for ρ_i^h and d_i^h, $i = 1, 2$, are randomly generated, where ρ_i^h varies from 0 to 1, d_1^h from 200 to 1,000, and d_2^h from 400 to 2,000. Other parameter values fixed in the following **Experiments 1**, **2**, and **3** are listed in Table 1. Let p be the join selectivity of the query. p is considered to be the product of β_1 and β_2. We assume $\beta_1 = \beta_2$ in the simulation. Each experimental result is obtained by averaging the results of 500 different inputs. Let *heu_cost* and *opt_cost* be the costs for the execution plans generated by **HEU** and **OPT**, respectively, and *perc* represent

$$(heu\_cost - opt\_cost) \times 100/opt\_cost.$$

Experiment 1. To observe the relationship between the cost of the function call and *perc*. In the experiment, X_1 and X_2 are set to 60 μsec/byte.

From Fig. 2, it can be observed that, when the cost of the function call is low, the difference between *heu_cost* and *opt_cost* is small. This is because the transmission cost is

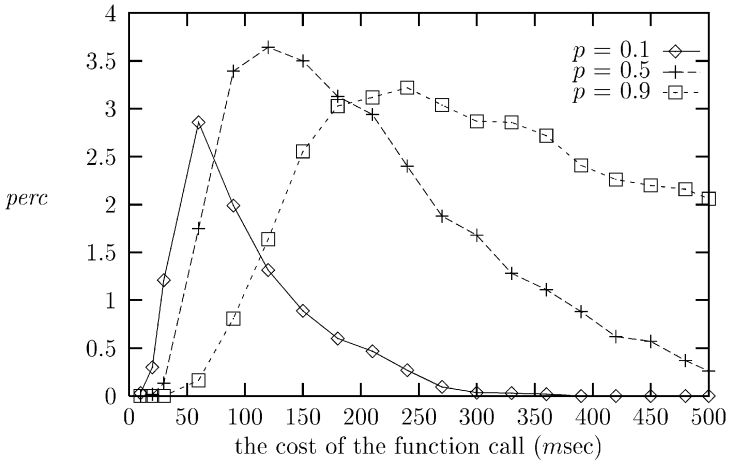

Fig. 2. The result of Experiment 1.

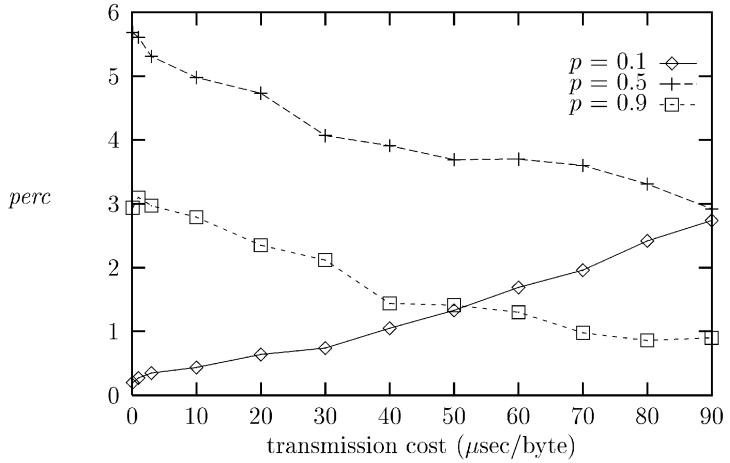

Fig. 3. The result of Experiment 2.

high and most of the selection predicates are evaluated before data transmission for both execution plans generated by **HEU** and **OPT**. As the cost of the function call increases, the difference between heu_cost and opt_cost increases gradually. When the cost of the function call exceeds a peak value, the difference between heu_cost and opt_cost lessens gradually since increasing a number of selection predicates is evaluated after join processing for both the execution plans generated by **HEU** and **OPT**. It can be seen that the higher the join selectivity is, the higher the peak value will be.

Experiment 2. To observe the relationship between the transmission cost and *perc*. In the experiment, the cost of the function call is set to 100 $m$sec.

From Fig. 3, it can be observed that when the transmission cost is low, the difference between heu_cost and opt_cost is small for $p = 0.1$ since most of the selection predicates will be evaluated after join processing for both the execution plans generated by **HEU** and **OPT**, while those for $p = 0.5$ and $p = 0.9$ are greater since the benefit obtained by evaluating selection predicates after join processing is lower. As the transmission cost increases, the difference between heu_cost and opt_cost increases gradually for $p = 0.1$, while that for $p = 0.5$ and $p = 0.9$ decreases gradually. Especially for $p = 0.9$ in the environment of high transmission cost, *perc* is less than one since most of the selection predicates will be evaluated before data transmission for both execution plans generated by **HEU** and **OPT**.

Experiment 3. To observe the relationship between the join selectivity and *perc*. Let X_{LAN}, X_{MAN}, and X_{WAN} be the transmission costs in LAN, MAN, and WAN environments, respectively. In the experiment, the cost of the function call is set to 100 $m$sec.

From Fig. 4, it can be observed that when the join selectivity is less than 0.2, the difference between heu_cost and opt_cost is small in the cases of X_{LAN} and X_{MAN} since most of the selection predicates will be evaluated after the join processing for both the execution plans generated by **HEU** and **OPT**, while that in the case of X_{WAN} is greater. As the join selectivity increases, the difference between heu_cost and opt_cost increases gradually for X_{LAN} and X_{MAN}. When the join selectivity exceeds a peak value, the difference between heu_cost and opt_cost lessens gradually. This is because the cost of data transmission will increasingly outweigh the benefit obtained by evaluating selection predicates after join processing.

Experiment 4. To observe the relationship between relation size and *perc* in a WAN environment. In the experiment,

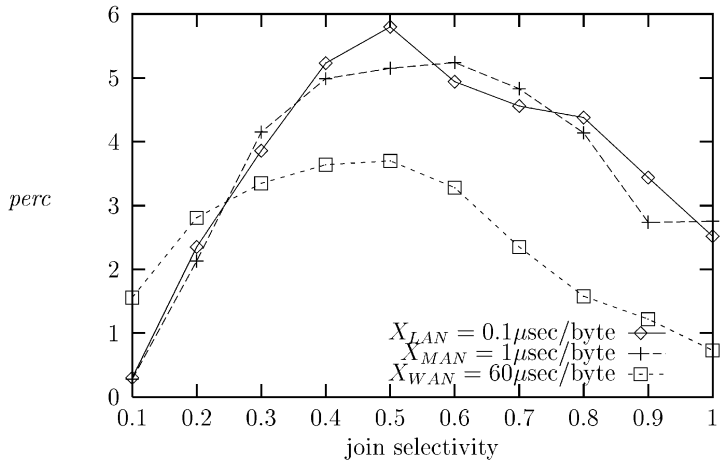

Fig. 4. The result of Experiment 3.

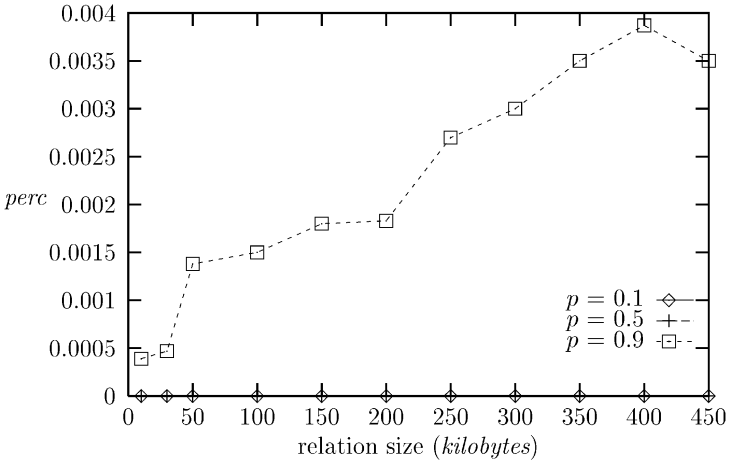

Fig. 5. The result of Experiment 4.

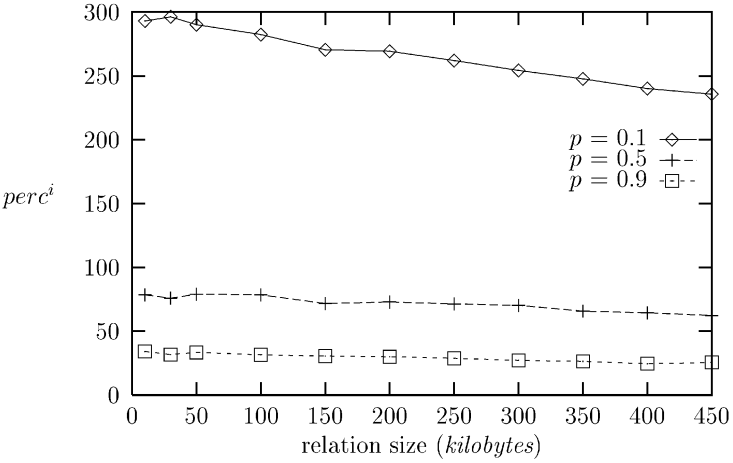

Fig. 6. The relationship between relation size and $perc^i$.

the cost of the function call is set to 100 $m$sec, the size of each tuple is 1 kilobytes, and the cache is assumed to be kept in the main memory. Thus, the time for accessing the cache is negligible.

From Fig. 5, it can be observed that when the relation size is small, the difference between heu_cost and opt_cost is small. (Note that here heu_cost and opt_cost are the costs for the execution plans generated by the variations of **HEU** and **OPT**, respectively, which do not consider the cache access time.) It does not mean that the small relation size and the zero-time cache access will make the consideration of optimization unnecessary. Let ini_cost and max_cost be the costs for the execution plan which evaluates all the selection predicates before data transmission and for that which has the maximal execution cost, respectively. $perc^i$ and $perc^m$ represent $(ini\_cost - opt\_cost) \times 100\ opt\_cost$ and

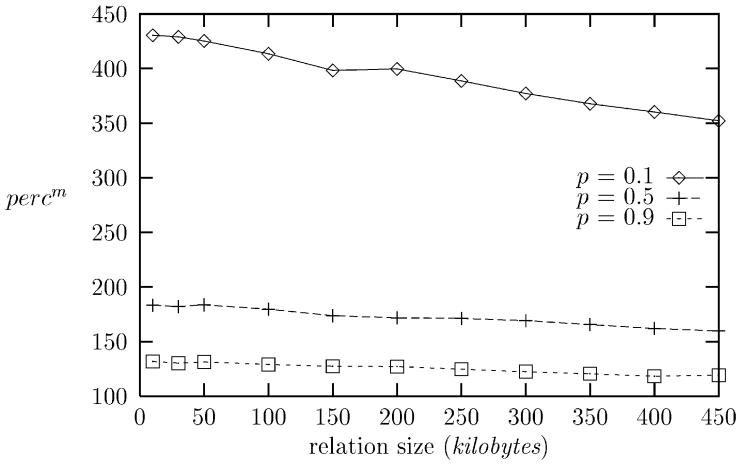

Fig. 7. The realtionship between relation size and $perc^m$.

$(max\_cost - opt\_cost) \times 100/opt\_cost$, respectively. The relationship between relation size and $perc^i$ is shown in Fig. 6. It can be seen that $perc^i$ is large, especially for $p = 0.1$ in the case of the small relation size. This is because in the case of the small join selectivity and the low data transmission cost for the small relation size, most of the selection predicates will be evaluated after join processing for both the execution plans generated by the variations of **HEU** and **OPT**. Moreover, $perc^m$ in Fig. 7 shows that max_cost is greater than double opt_cost. In a LAN environment, the result is similar. Therefore, the work to consider the placement of selection predicates is significant even in the case of small relation size and zero-time cache access.

From the simulation results, the average cost of the execution plan by **HEU** is less than 1.03 times of the optimal execution plan, which demonstrates the good quality of heuristic algorithm **H**.

6.2 Simulation Results for Heuristic Algorithm GH

Let n be the number of relations to be joined in a query. In the simulation, we assume there are 12 selection predicates in a query and each relation involves $12/n$ selection predicates. The parameter settings which are different from that in Table 1 are described in Table 2. In Table 2, the expression $[x, y]$ denotes that a value in the range of x to y is randomly generated. Let heu_cost and opt_cost be the costs for the execution plan generated by heuristic Algorithm **GH** and the optimal execution plan, and $perc$ represent $(heu\_cost - opt\_cost) \times 100/opt\_cost$. The simulation result in Table 3 is obtained by averaging the results of 30,000 different inputs. It can be seen that the more the number of relations in a query is, the less the difference between opt_cost and heu_cost is. This is because when the number of relations in a query increases, more of the selection predicates are prone to be evaluated before data transmission, which can be executed in parallel at different sites. (Recall that relations are assumed to be located at different sites and the numbers of related selection predicates for relations are equal.) In other words, if the degree of parallelism for evaluating selection predicates before join processing increases, the benefit obtained by evaluating selection predicates after join processing reduces since those selection predicates delayed and evaluated after join processing are confined to be executed at a final site in a sequential order.

It can be seen that the average cost of the execution plan generated by heuristic Algorithm **GH** is also less than 1.03 times that of the optimal execution plan, which demonstrates the good quality of heuristic Algorithm **GH**.

TABLE 2
Parameters Description

the cost of a function call	[10000,450000] (μsec)
the cardinality of a relation	[100,2000]
the size of a tuple	200 (bytes)
join selectivity	[0,1]
selection selectivity	[0,1]
data transmission cost	0.1 or 60 (μsec/byte)

6.3 The Effect of Inaccurate Join Selectivity

In the cost model, we make the assumptions of accurate selectivity estimation, attribute independence, and uniformity of attribute values, which are adopted by most query optimization algorithms. The impacts of these assumptions in a practical environment are discussed in [11], [23]. The estimation of the selectivity for a join is more difficult than that for a selection predicate. In the following, we consider the effect of inaccurate join selectivities on the execution result of the plan generated by **HEU**.

Experiment 5. To observe the effect of inaccurate join selectivities on the execution result of the plan generated by **HEU**. The parameter setting is the same as that for **Experiment 2** in a WAN environment.

Assume p is the accurate join selectivity. Let $cost1$ and $cost2$ be the costs for the execution plans generated by **HEU** with $p + 0.1$ and $p - 0.1$ as the join selectivity inputs, respectively, and $cost3$ be the cost for the initial execution

TABLE 3
The Simulation Result for Heuristic Algorithm **GH**

number of relations	2	4	6
$perc$	2.86	2.34	1.05

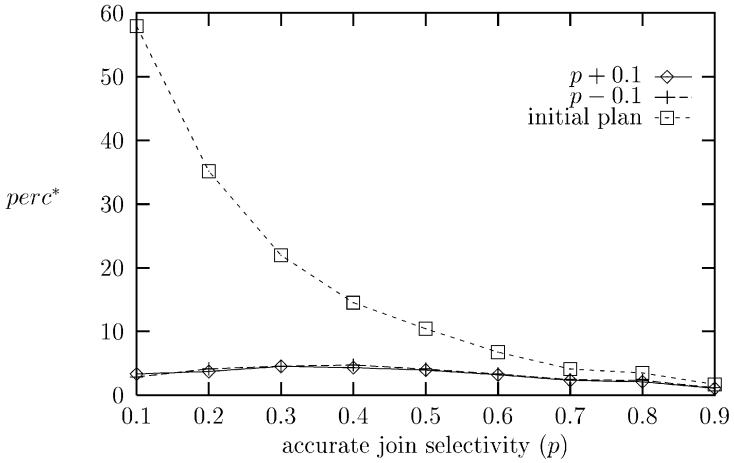

Fig. 8. The result of Experiment 5.

plan (i.e., all the selection predicates are evaluated before data transmission) with join selectivity p. $perc^*$ represents $(c - opt\_cost) \times 100/opt\_cost$, where opt_cost is the cost for the execution plan generated by **OPT** with p as the join selectivity input, and c is $cost1$, $cost2$, or $cost3$. From Fig. 8, it can be observed that $perc^*$ for the cases of $p + 0.1$ and $p - 0.1$ is less than 5, and $perc^*$ for the initial plan is greater than that in the cases of $p + 0.1$ and $p - 0.1$, especially when p is small, the difference between $cost3$ and $cost1$ (or $cost2$) is large. This is because most of the selection predicates will be delayed and evaluated after join processing in the cases of small p values. In general, when the difference of the estimated join selectivity and the accurate join selectivity is less than 0.1, the cost of the execution plan generated by **HEU** is still close to that of the optimal plan.

7 Conclusions and Future Work

In this paper, we consider the problem of optimizing the query with foreign functions in a distributed environment. An efficient heuristic algorithm is proposed to solve the general problem and the simulation result shows its good quality compared to the optimal execution plan. Moreover, from the simulation result, the efficiency of the execution plan generated by the proposed heuristic algorithm greatly outperforms that by the initial strategy, which demonstrates the significance of our work.

In this paper, we assume the values of one attribute are independent of those in another attribute (Assumption A1), and the number of distinct values in an attribute is proportional to the cardinality of the relation (Assumption A3). Based on these two assumptions, we have **Lemma 1**. **Lemma 1** implies that if the selection predicate has a lower computation cost for the corresponding foreign function and a lower selectivity, it should be evaluated as early as possible. Algorithms **H** and **GH** are developed based on the implication. In Assumption A2, the cache is assumed to reside in disk. If we assume the cache resides in main memory, the time for accessing the cache (i.e., t_c) can be ignored. In this case, $rank_1(p^{u_h})$, $rank_2(p^{v_h})$, and $rank_3(p^{w_h})$ in Definition 2 become $\frac{C^{u_h}}{1-\rho^{u_h}}$, $\frac{C^{v_h}}{1-\rho^{v_h}}$, and $\frac{C^{w_h} \cdot r^{w_h}}{1-\rho^{w_h}}$, respectively. However, it does not change the property of **Lemma 1** except that t_c is zero, and the MRT problem is still NP-hard.

Our approach can complement the one-shot semijoin approach [33] which also considers minimizing the response time for distributed query processing. The combined approach is described as follows:

Step 1. Apply Algorithm **GH** to determine the set of selection predicates, say S, to be evaluated before data transmission and the set of selection predicates, say S', to be evaluated after join processing.

Step 2. Evaluate selection predicates in S at local sites.

Step 3. Use the one-shot semijoin algorithm to determine the set of semijoins to be executed and execute them.

Step 4. Transmit all the participant relations to the final site and execute joins.

Step 5. Evaluate selection predicates in S' at the final site.

On the other hand, our approach can fit in the Epoq extensible query optimizer architecture. The extensible optimizer consists of several region modules constructed in a hierarchical manner, each of which implements a strategy for the query optimization. The root module receives a query from the query processing system and plans the execution of the query with the assistance of its subordinate regions. According to the Epoq architecture, the rewrite rule approach (named "RWR") [7] and our approach (named "MRT") can be considered to be stand-alone region modules. Therefore, if a query has joins and selection predicates with foreign functions, it can be optimized by the region "RWR" first and then the region "MRT." The join predicate considered in this paper does not involve foreign functions. Chaudhuri et al. [8] investigated a class of queries that include "foreign join" between relations and documents. Extending our approach on foreign joins is under investigation. From the simulation result shown in Section 6.2, we find that parallel processing is a good strategy for optimizing queries with foreign functions. Another interesting future work would be the consideration of processing queries with foreign functions on parallel machines.

REFERENCES

[1] R. Ahmed, P.D. Smedt, W. Du, W. Kent, M.A. Ketabchi, W.A. Litwin, A. Rafii, and M.C. Shan, "The Pegasus Heterogeneous Multidatabase System," *Computer,* pp. 19-27, Dec. 1991.

[2] P.M.G. Apers, A.R. Hevner, and S.B. Yao, "Optimization Algorithms for Distributed Queries," *IEEE Trans. Software Eng.,* vol. 9, no. 1, pp. 57-68, 1983.

[3] M.P. Atkinson and O.P. Buneman, "Types and Persistence in Database Programming Languages," *ACM Computing Surveys,* vol. 19, pp. 105-190, 1987.

[4] P.A. Bernstein et al., "Query Processing in a System for Distributed Databases (SDD-1)," *ACM Trans. Database Systems,* vol. 6, no. 4, pp. 602-625, 1981.

[5] Y. Breitbart, P.L. Olson, and G.R. Thompson, "Database Integration in a Distributed Heterogeneous Database System," *Proc. Int'l Conf. Data Eng.,* pp. 301-310, 1986.

[6] M. Carey, D. DeWitt, J. Richardson, and E. Shekita, "Object and File Management in the EXODUS Extensible Database System," *Proc. Int'l Conf. Very Large Data Bases,* pp. 91-100, 1986.

[7] S. Chaudhuri and K. Shim, "Query Optimization in the Presence of Foreign Functions," *Proc. Int'l Conf. Very Large Data Bases,* pp. 529-542, 1993.

[8] S. Chaudhuri, U. Dayal, and T. W. Yan, "Join Queries with External Text Sources: Execution and Optimization Techniques," *Proc. ACM Int'l Conf. Management of Data,* pp. 410-422, 1995.

[9] S. Chaudhuri and K. Shim, "Optimization of Queries with User-Defined Predicates," *Proc. Int'l Conf. Very Large Data Bases,* pp. 87-98, 1996.

[10] H. Chen, X. Yu, K. Yamaguchi, H. Kitagawa, N. Ohbo, and Y. Fujiwara, "Decomposition—An Approach for Optimizing Queries Including ADT Functions," *Information Processing Letters,* vol. 43, pp. 327-333, 1992.

[11] S. Christodoulakis, "Implications of Certain Assumptions in Database Performance Evaluation," *ACM Trans. Database Systems,* vol. 9, no. 2, pp. 163-186 1984.

[12] U. Dayal and H.Y. Hwang, "View Definition and Generalization for Database Integration in a Multidatabase System," *IEEE Trans. Software Eng.,* vol. 10, no. 6, pp. 628-644, 1984.

[13] O. Deux et al., "The Story of O_2," *IEEE Trans. Knowledge and Data Eng.,* vol. 2, no. 1, pp. 91-108, 1990.

[14] D. Fishman et al., "Iris: An Object-Oriented Database Management System," *ACM Trans. Office Information Systems,* vol. 5, pp. 48-69, 1987.

[15] M.R. Garey and D.S. Johnson, *Computers and Intractability: A Guide to the Theory of NP Completeness,* San Fransisco: W.H., 1979.

[16] G. Graefe and W.J. McKenna, "The Volcano Optimizer Generator: Extensibility and Efficient Search," *Proc. Int'l Conf. Data Eng.,* pp. 209-218, 1993.

[17] L. M. Hass et al., "Extensible Query Processing in Starburst," *Proc. ACM Int'l Conf. Management of Data,* pp. 377-388, 1989.

[18] L.M. Haas et al., "Starburst Mid-Flight: As the Dust Clears," *IEEE Trans. Knowledge and Data Eng.,* vol. 2, no. 1, pp. 143-160, 1990.

[19] J.M. Hellerstein and M. Stonebraker, "Predicate Migration: Optimizing Queries with Expensive Predicates," *Proc. ACM Int'l Conf. Management of Data,* pp. 267-276, 1993.

[20] J.M. Hellerstein, "Practical Predicate Placement," *Proc. ACM Int'l Conf. Management of Data,* pp. 325-335, 1994.

[21] J.M. Hellerstein and J.F. Naughton, "Query Execution Techniques for Caching Expensive Methods," *Proc. ACM Int'l Conf. Management of Data,* pp. 423-434, 1996.

[22] R. Hull and R. King, "Semantic Database Modeling: Survey, Applications, and Research Issues," *ACM Computing Surveys,* vol. 19, pp. 201-260, 1987.

[23] Y.E. Ioannidis and S. Christodoulakis, "On the Propagation of Errors in the Size of Join Results," *Proc. ACM SIGMOD,* pp. 268-277, 1991.

[24] A. Kemper, G. Moerkotte, H.D. Walter, and A. Zachmann, "GOM: A Strongly Typed, Persistent Object Model with Polymorphism," *Proc. Datenbanksysteme in Buro, Technik und Wissenschaft,* pp. 198-217, 1991.

[25] A. Kemper, C. Kilger, and G. Moerkotte, "Function Materialization in Object Bases," *Proc. ACM Int'l Conf. Management of Data,* pp. 258-267, 1991.

[26] A. Kemper, G. Moerkotte, K. Peithner, and M. Steinbrunn, "Optimizing Disjunctive Queries with Expensive Predicates," *Proc. ACM Int'l Conf. Management of Data,* pp. 336-347, 1994.

[27] A.Y. Levy, I.S. Mumick, and Y. Sagiv, "Query Optimization by Predicate Move-Around," *Proc. Int'l Conf. Very Large Data Bases,* pp. 96-107, 1994.

[28] G. Lohman et al., "Query Processing in R*," *Query Processing in Database Systems,* W. Kim, D. Reiner, and D.S. Batory, eds., Springer Verlag, 1985.

[29] G. Mitchell, U. Dayal, and S.B. Zdonik, "Control of an Extensible Query Optimizer: A Planning-Based Approach," *Proc. Int'l Conf. Very Large Data Bases,* pp. 517-528, 1993.

[30] H. Pirahesh, J.M. Hellerstein, and W. Hasan, "Extensible/Rule Based Query Rewrite Optimization in Starburst," *Proc. ACM Int'l Conf. Management of Data,* pp. 39-48, 1992.

[31] M. Stonebraker, "Inclusion of New Types in Relational Data Base Systems," *Proc. Int'l Conf. Data Eng.,* pp. 262-269, 1986.

[32] M. Stonebraker and L.A. Rowe, "The Design of POSTGRES," *Proc. ACM Int'l Conf. Management of Data,* pp. 340-355, 1986.

[33] C. Wang, A.L.P. Chen, S.C. Shyu, and A Parallel, "Execution Method to Minimizing Distributed Query Response Time," *IEEE Trans. Parallel and Distributed Systems,* vol. 3, no. 3, pp. 325-333, 1992.

[34] K. Wilkinson, P. Lyngbaek, and W. Hasan, "The Iris Architecture and Implementation," *IEEE Trans. Knowledge and Data Eng.,* vol. 2, no. 1, pp. 63-75, 1990.

[35] S.B. Yoo and P.C.Y. Sheu, "Evaluation and Optimization of Query Programs in an Object-Oriented and Symbolic Information System," *IEEE Trans. Knowledge and Data Eng.,* vol. 5, no. 3, pp. 479-495, 1993.

Pauray S.M. Tsai received the BS and MS degrees in computer science and information engineering from National Chiao Tung University, Taiwan, in 1990 and 1992, respectively, and the PhD degree in computer science from National Tsing Hua University, Taiwan, in 1996. Currently, she is an associate professor in the Department of Information Management, Ming Hsin Institute of Technology, Taiwan. Her research interests include multidatabase systems, object-oriented database systems, and data mining.

Arbee L.P. Chen received the BS degree in computer science from National Chiao-Tung University, Taiwan, Republic of China, in 1977, and the PhD degree in computer engineering from the University of Southern California in 1984. He joined National Tsing Hua University, Taiwan, as a National Science Council (NSC) sponsored visiting specialist in August 1990 and became a professor in the Department of Computer Science in 1991. He was a member of the technical staff at Bell Communications Research, New Jersey, from 1987 to 1990, an adjunct associate professor in the Department of Electrical Engineering and Computer Science, Polytechnic University, New York, and a research scientist at Unisys, California, from 1985 to 1986. His current research interests include multimedia databases, data mining, and mobile computing. Dr. Chen has organized (and served as a program co-chair) the 1995 IEEE Data Engineering Conference and the 1999 International Conference on Database Systems for Advanced Applications (DASFAA) in Taiwan. He is a recipient of the NSC Distinguished Research Award. He is member of the IEEE and the IEEE Computer Society.