INTEGRATION OF DATABASE SYSTEMS AT THE NAVIGATIONAL LEVEL BY USING PROLOG

6H−7

Motoshi KATSUMATA and Makoto TAKIZAWA
Tokyo Denki University

1. INTRODUCTION

In the distributed database system, views independent of heterogeneity of database systems have to be provided. We adopt a Prolog-like system as the common interface on the database systems. Although many researchers have tried to integrate multiple database systems at the higher level like the relational model, we try to integrate them at the lower, navigational level. In this paper, we try to get an efficient access program from the non-procedural Prolog-like query on the navigational database systems like the conventional network database systems, UNIX file systems, and even relational database systems. Our method aims at reducing not only the number of access units but also the number of redundant answers. Also, the access program is tried to be executed in parallel.

2. DISTRIBUTED NAVIGATIONAL DATABASE SYSTEM

The navigational database is a set of navigational objects. The navigational object O is composed of a totally ordered set OD_O of instances and a collection P_O of operations on OD_O. Distributed navigational database system is composed of more than one navigational database system. For example, a navigational database system is composed of three navigational database systems DB_1, DB_2, and DB_3.

DB_1: s(@s, sname) DB_2: p(@p, pname)
DB_3: sp(@sp,@p,@s), pb(@pb,@b,@p),
bp(@bp,@b,@p), b(@b, role)

Fig.2.1 Distributed Navigational Database

A view is defined on the distributed navigational database like this.

```
ssp(SNAM,PNAM):- s(S,SNAM),sp(P,S),p(P,PNAM).
```

The view ssp means that a supplier SNAM supplies parts PNAM. A query "find parts supplied by a supplier a" is written like this.

```
query(PN) :- ssp("a", PN).
```

Here, PN is a target variable and "a" is a constant.

3. NAVIGATIONAL PROGRAM

Now, we try to translate a simple query to a navigational program which accesses the database.

3.1 NAVIGATIONAL TREE

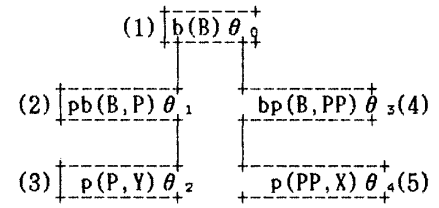
We introduce a navigational tree for a query $Q=query(X):-B_1,\ldots,B_n$. Now the query Q is written in a set $\{B_1,\ldots,B_n\}$.

[Definition] A navigational tree T = (V,B) for a query Q is defined as follows.

(1) V is a set of nodes. For each atom B in Q, there exists a node $N_B=<B, \theta_B, \sigma_B>$, where θ_B and σ_B are substitutions.

(2) B is a set of branches $N_P \rightarrow N_C$, where $N_P=<P, \theta_P, \sigma_P>$ is a parent of $N_C=<C, \theta_C, \sigma_C>$. Here, $\theta_C = \sigma_P$ and σ_C is a substitution which instantiates $C\theta_C$. □

For a query Q, a navigational tree T is constructed by the procedure SP[TAKI87]. A tree in Fig.3.1 is an example of a navigational tree for a query Q1=query(Y,PP) :- p(P,Y), pb(B,P), p(PP,X), b(B), bp(B,PP).



$\theta_0=\varepsilon$ $\theta_1=\theta_0\{B/c_B\}$ $\theta_2=\theta_1\{P/c_P\}$
$\theta_3=\theta_0\{PP/c_{PP}\}$ $\theta_4=\theta_3\{X/c_X\}$

Fig.3.1 Navigational Tree

3.2 REDUCTION OF REDUNDANT ANSWERS

Next, we try to get all the answer substitutions without redundant refutations. Here, let T be a navigational tree. X is a some nodes in T. Let parent(X) be X's parent in T. For a node A in T, suppose that there are n nodes $B_1,\ldots,B_n$ where $A = parent(B_j)$ for $j=1,\ldots,n$. Let T_j be a subtree of B_j. Suppose that a substitution θ_A is obtained by the resolution of A. Let $Ans(B_j)$ be a set of refutations obtained from $T_j\theta_A$. In the Prolog program, for each $A\theta_A$, a cartesian product $CA_A = Ans(B_1) \times \ldots \times Ans(B_n)$ is obtained. Here, let T_{hk} be a subtree which includes targets in $T_1,\ldots,T_n$ $(k=1,\ldots,p, p\leq n)$. In our method, only a projection of CA_A on $T_{h1},\ldots,T_{hp}$, i.e. $AA_A = Ans(B_{h1}) \times \ldots \times Ans(B_{hp})$ is accessed. Since it is clear that $|CA_A| \geq |AA_A|$, we can get all the answer substitutions in less accesses to the database than the Prolog program. Here, we present the navigational program in a network of objects where the objects are nodes in T and two special objects ST and OUT. ST is a start node. OUT node outputs the answer. Fig.3.2 shows an navigational program for Q1.

INTEGRATION OF DATABASE SYSTEMS
AT THE NAVIGATIONAL LEVEL BY USING PROLOG
Motoshi KATSUMATA, Makoto TAKIZAWA
Tokyo Denki University

[Proposition] For a given navigational tree T, less instances are accessed and less redundant answers are obtained by the navigational program than the Prolog one.■

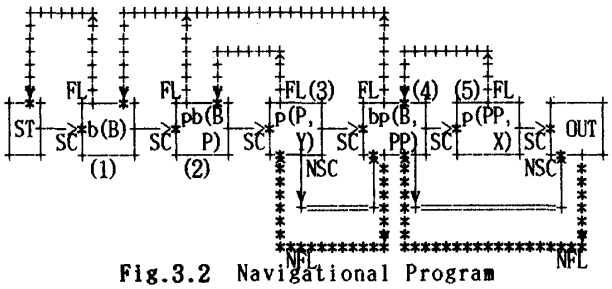


Fig.3.2 Navigational Program

4. NAVIGATIONAL PROGRAM FOR MULTIPLE INPUT RULES

Let us consider the following query.

```
query(T,U) :- A(X,Y), B(X,T), C(T), V(Y,U).
V(P,Q) :- D(P,Q,Z), E(Z).
V(P,Q) :- F(P,V,Z), G(V,Q), H(Z).
```

The views in the query are replaced by the right hand side of the views. For example, the following two queries are obtained.

```
query1(T,U) :- A(X,Y), B(X,T), C(T),
               D(Y,U,Z), E(Z).
query2(T,U) :- A(X,Y), B(X,T), C(T),
               F(Y,V,Z), G(V,U), H(Z).
```

One method to get the answers of query is to take the union of two results obtained by query1 and query2. However, in both queries, A, B, C are commonly evaluated. We try to reduce this redundant processing by introducing a new BR(branch) node O. When constructing a navigational program for a given query, if a rule atom A is selected for some parent node P, an BR node is created as a child of P. Navigational tree for a given query is shown in Fig.4.1.

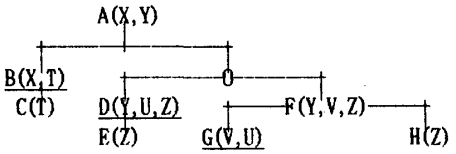


Fig.4.1 Navigational tree with BR node O

By our method, A,B,C are evaluated only once in Fig.4.1. Thus, redundant evaluations can be prevented.

5. OR PARALLELISM

Next, we try to execute concurrently a program for a navigational tree as shown in Fig.4.1. One method to get the answers of query is to execute two subtrees of O in parallel and take the union of two results obtained by them. So we introduce an object named OR for controlling this case to the navigational program. When constructing a navigational program for a given query, if a rule atom A is selected for some parent node P, an OR node is created as a child of P. By introdusing a new OR node in Fig4.1, node D,E and F,G,H are executed in parallel. There are two kinds of OR objects. i.e. all-wait(AWO) and one-wait(OWO) OR objects. Suppose that the subtrees at the right hand side of an OR node O do not include any target node. If one of the subtree S_j is found to success, we do not have to wait for the completion of all the other subtrees at the OR node O. This type of OR node is named a OWO node. On the other hand, let us consider that the subtrees of the OR node O include some target node. It is clear that of some subtree includes target node, all of the subtrees of O includes target node. In this case, even if all the answer substitutions are obtained from one subtree of O, we can't stop to wait for the other subtrees. When all the refutations for all the subtrees complete, we can backtrack to the ancestor node of O. This type of the OR node is named AWO node.

6. IMPROVED OR PARALLELISM BY PREFETCHING

Using the AWO node, we can get all the answers in subtrees including target node. Here, let rtarg(A) be a target node B leftmost to A in navigational tree. AWO node is waiting for the message from all the subtrees T_j (j=1,...,n). Here, if we can get the next answers of rtarg(T_j) unless we wait for the token from all T_j, it is much better from the point of efficient processing. From this considerations, we introduce the PAWO(Prefetch AWO) node.

7. CONCLUDING REMARKS

In this paper, we have presented a Prolog-like query language interface on the multiple various navigational database systems like the conventional network database systems and Unix file systems. In our system, derivation of the redundant answers are prevented by accessing navigationally the database without making intermediate files. Also, we have shown the method to realize the OR parallelism so as to decrease the redundant answer substitutions.

REFERENCES

[TAKI87] Takizawa, M., et al., "Logic Interface System on Navigational Database System," Lecture Notes in Computer Science, Springer-Verlag, No.264, 1987, pp.70-80.

[TAKI89a] Takizawa, M. and Katsumata, M., "Access Procedure to Minimize Redundant Refutations on the Network Database System," Lecture Notes in Artificial Intelligence, Springer-Verlag, No.383, 1989, pp.156-171.

[TAKI89b] Takizawa, M. and Katsumata, M., "Prolog Interface System on Distributed Navigational Database Systems," A Working Conference on Data and Knowledge Engineering, 1989.

[WARR]Warren, D. H. D., "Efficient Processing of Interactive Relational Database Queries Expressed in Logic," Proc. of the VLDB, 1981, pp.272-281.