"I wish I were over there": Distributed Execution Protocols for Data Definition in R*

PAUL F. WILMS
BRUCE G. LINDSAY
PATRICIA G. SELINGER

IBM San Jose Research Lab
5600 Cottle Road
SAN JOSE, CA95193
USA

Abstract

The design and implementation of R*, an experimental prototype of a distributed system for the management of interrelated, voluntarily cooperating, but also autonomous databases is based on several major objectives: site autonomy, transparency, ease of use and performance. This paper discusses the way data definition and control statements are executed in a distributed environment and shows how the general objectives are fulfilled. Specialized distributed execution facilities have been developed to facilitate the implementation of complex multi-site functions. This paper describes the facilities and methodology used to implement the distributed processing needed to perform multi-site data definition operations in R*.

1. Introduction

R*, an experimental prototype of a distributed database management system (DDBMS) [HAA82], differs considerably from most of the currently existing database management systems in the sense that data manipulation statements (like SELECT, INSERT, UPDATE, DELETE) are compiled rather than interpreted. This compilation approach [CHA81] has been chosen to increase performance since data manipulation statements (DML) will often be run repeatedly. Indeed, compiled statements will save a great part of the processing (e.g. query optimization) which would have to be repeated each time a DML request was run if the compiler was replaced by an interpreter. Although compilation is a little more expensive, the reader should keep in mind that a real application accesses data via programs which are more likely to be executed many times, so the time spent in compilation can be really considered as negligible in a long run.

On the other hand, data definition statements and access authorization statements are typically not intended to run repeatedly. Consequently, the time spent to compile these statements would be a burden and so it makes sense to interpret such statements. Furthermore, these statements frequently do not do very much that could be offloaded to compile time.

In R*, however, a new dimension is added: data definition and access authorization statements may operate on objects located at remote sites. The interpretation of these statements must locate the operands and perform the necessary operations at the site(s) of the operands. If an operand is itself distributed accross multiple sites, the interpretation must lead to coordinated actions at several sites. The algorithms used should, of course, seek to minimize the number of intersite interactions and should only require the participation of sites storing (part of) the object of the operation.

The main problem addressed in this paper can be summarized as follows: in building software for distributed systems where objects can move, how can procedures implementing data definition functions be written so that they work for both local and remote objects with a minimum of extra mechanism, good performance, and a high degree of autonomy. This paper presents the major protocols used by the data definition and control routines in R*, gives the mechanism used for formally describing the distributed execution patterns, and discusses the advantages as compared to other possible approaches.

2. Naming vs location

An important feature of many entities in nature is their ability to move or to be moved. This process of location switching doesn't affect however the identity of the entity to be moved; only the access path used to reach the entity is modified. A DDBMS should be able to handle, in addition to static entities, entities subject to such location changes without changing any name references to these objects after they move.

Two kinds of entities can be distinguished in R*: static entities and dynamic entities. Static entities include users, programs, or sites at which some action has to take place. In this case, the location at which the entity resides is either explicit (contained in the query issued by the user), or implicit (the entity location assigned to be the site of the user issuing the request, by default). Tables in R*, on the other hand, are dynamic entities. They are the only entities whose naming doesn't constitute a sufficient clue to the system for deducing their location.

The naming convention designed for R* meets the requirement to access movable entities through the specification of a constant name. Its mechanism can be summarized as follows:

- ***Print Name***: this is a character string embedded in a program or an ad-hoc query to reference an object. Its specification remains constant throughout the lifetime of the object, regardless of whatever location stores the object, and whatever modification such as replication or partitioning it might undergo.

System Wide Name: a name resolution facility [LIN80] converts the user's print name (PN) into a system wide name (SWN). This process involves searching for possible synonyms defined by the user to map selected names. If no synonym matches the PN, name completion rules extend the unmapped PN into a correct object reference. This process is strictly local and leads to the automatic elaboration of a SWN composed of the local object name, its birth site, the identity of the object creator, and the creator site.

Upon completion of a name, the DDBMS is in charge of finding the actual site at which the object is stored (the store site), and routing the request to that particular site. The associated protocol will be described in the next sections. The system catalogs [LIN80] record information about the objects of the distributed database in the following general matter:

- the *birth site* (BS) of a table contains in its system catalog the location identification of the site which actually stores the object;
- the catalogs at the table *store site* (SS) contain a full description of that table (the columns of the table, the indexes created on that table, the users having access rights on the table,...);
- the *other sites* may contain cached information about that table: the location of the store site will be recorded in the catalog at such a site after the table has been accessed for the first time from that site; this information can, of course become obsolete if the table is transferred to another site.

Complete replication of a central catalog is an alternative which makes it easy to find the identity of the current store site. However, the maintenance of a complete catalog at each site is costly in terms of storage, and there is a problem in keeping all these catalogs up to date. Whenever any user at any site wants to create tables, migrate or drop those tables, either all sites must be available and able to make the change, or a complex catalog replica update scheme must be invented which can deal with site unavailability and recovery. Such a scheme would behave like an expensive replacement for the catalog caching concept in R* with every site starting with caches for every object.

This concept of complete independence between object naming and object location is required in a DDBMS to avoid modifying the location part of the object name in existing programs after each object migration. A side effect of this mechanism is the data location transparency presented to the user. It must also be understood that this transparency is not aimed at hiding from the user the identity of the real object store site, and is therefore not intended as a protection mechanism. Such protection could however be provided, if necessary, by revoking from the user the right to consult the system catalog containing the appropriate information.

3. Classification of protocols

From the classification of entities given in the previous section, a distinction between data definition statements is suggested:

- SOPS: Statically Oriented Processing Statements. These statements involve only static entities. The principal location of execution is contained in the statement itself.
- DOPS: Dynamically Oriented Processing Statements. These statements contain at least one dynamic entity. As a consequence, the principal location of execution can no longer always be deduced from reading the statement.

Two protocols are needed to execute these statements, one for SOPS and one for DOPS. The protocol which executes SOPS is simple: R* determines from the statement itself where the statement should execute, and then requests that site to perform the necessary operation locally. The protocol for DOPS has the following basic form: locate the sites in charge of accessing the required entities, and then issue requests to those sites to perform the necessary operations locally. The next section will describe the protocols in more detail.

4. Execution strategy of interpreted routines

We will now examine the techniques used by the two protocols mentioned above to locate the entities referenced in the statements and to execute the distributed statements. We will concentrate our attention on data definition statements (DDS), data configuration statements (DCS), and authorization management statements (AMS), since other papers [DAN82] adequately cover the protocols for the subset of DOPS which constitutes the data manipulation language (DML).

The number of sites involved in the search for the right execution site varies depending on the class to which the statement belongs. In R*, SOPS may involve at most one remote site. Up to three remote sites may be involved in running statements in the DOPS class. The protocol for SOPS is called C1RS, and the C3RS protocol handles the DOPS class.

Before we describe the actual protocols, let us examine the mechanisms they might use for getting work done at other sites. The execution of a possibly distributed statement may essentially be decomposed into four steps:

(1) the transaction initiation at some site called the origin site, or query site,
(2) the migration of the transaction to some remote sites,
(3) the processing at these remote sites,
(4) and eventually the termination of the transaction which results in either the success or the failure of the transaction considered as an atomic action regardless of the number of sites involved in the processing.

The processing indicated by (3) at the remote sites participating in the execution of a transaction could use either (or both) of the following two mechanisms:

- calls to specific special routines at remote sites,
- distributed recursive calls.

The migration of execution from one site to another can be accomplished by using a Remote Procedure Call (RPC) like mechanism. Given an RPC facility, one can then consider how to organize the distributed computation. One straightforward approach is to identify which actions might occur at each site and to write procedures implementing each case of local processing. At execution, an umbrella procedure at the origin site activates the appropriate procedures after analysing the distributed execution requirements of the statement in question. This approach leads to a fragmentation of the logic implementing each statement type and leads to unique procedures representing each combination of origin and execution sites (typically there are 5 distinct site combinations for operations on tables).

The alternative approach taken in R* is to implement each statement level function as a single procedure (package) which executes the actions appropriate to its site of execution along with a facility which allows the execution of the statement to be initiated at another site. This approach, called ***distributed recursive calls***, initiates execution of the current statement at a remote site, passing only the statement and user identification (for authorization purposes) to the remote site. The procedures implementing a statement level function check to see if they should execute at another site to perform part or all of the requested function. If

remote execution is needed to complete the current function, special interfaces are available to initiate the function at another site and to return the results obtained at that site. The distributed recursive call allows the implementation of a function to examine the situation and conclude that "I wish I were executing this at site XYZ". The wish is fulfilled simply by requesting the remote execution of the current function.

The use of distributed recursive calls has several advantages for R*. The logic of an interpreted function is concentrated in a single procedure (package) which is structured according to the function in question, and not according to the possible distribution patterns of the function. More importantly, for R*, the mapping of the statement level interface to distributed execution enhances the autonomy of participating sites. The use of the statement level interface to distribute the execution allows complete checking of incoming recursive calls without implementing call specific code to confirm the coherence of each request. The original statement is checked by the parser on arrival which confirms the syntactic correctness of the request while the remote checks are uniformly embodied in the procedure(s) implementing the function.

The current design and implementation of DDS, DCS, and AMS in R* use distributed recursive calls. This provides a more modular set of procedures which preserve the autonomy of each R* database.

The mechanism of distributed recursive calls permits one common routine to manage a multi-site function. The global structure of a distributed recursive routine managing a multi-site request follows this schema:

```
distributed_execution procedure (p1,...,pn);
  begin
   - initialization and local processing;
   - classification of current site and verification that this
     site has some role to play;
   - determination of sites at which work must be done;
   - do over sites at which work must be done;
       if work is local
       then standard local processing;
       else call distributed_execution (p1,...,pn)
            AT SITE (remote_site);
   - end;      /* of do over each site */
   - termination of local processing;
  end.
```

The schema describes a sequential execution of the user's request in the sense that only one process at a time is running on behalf of this request throughout the distributed system. In the current implementation, a process which activates a remote process by issuing a remote recursive call will wait for the completion of the process at that remote site before being able to resume local execution. This prohibits any kind of parallelism among processes on behalf of a given user; however, for some applications, local processing could not resume until information is sent back from the callee, even if parallelism were permitted.

Sometimes, the condition of strict execution sequentiability can be relaxed to enhance performance. This is desirable if the same routine can be activated simultaneously at several remote sites, while the originator of the activation waits for the completion of the process at each of the remote sites. In the case of parallel processing for the same request at multiple sites, our previous schema would be modified by adding an asynchronous call and by permitting a "broadcast" call:

```
call distr_exec (p1,...,pn) AT SITES (remsite1,...,remsitem).
```

The underlying communication mechanism for R* [LIN82], [WIL82] permits both synchronous and asynchronous remote calls, and both of these are used in DML processing. For DDS, DCS and AMS, however, we did not consider performance to be of major importance and so we chose the simpler approach of using only synchronous remote procedure calls.

The processing schema given above is suitable for implementing most of the R* statements which are interpreted. In the remainder of this section, samples of the protocols used in R* for interpreted SOPS and DOPS statements will be given to illustrate the use of distributed recursive procedure calls and outline the kind of environment needed to use this mechanism.

5. Description of DDS distributed execution protocols

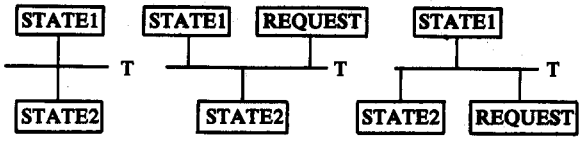
5.1. *The evaluation nets model*

Now that the mechanism has been chosen, we will employ a general model derived from evaluation nets [NUT72] to describe the protocols which use this mechanism. This model will permit us to present within one schema a global view of the request processing across all sites handling that request. A single evaluation net is capable of describing the global execution activity which is composed of the four steps mentioned earlier, and the part of the activity which is assigned to each specific site. This model can be used to describe a variety of distributed algorithms; it was very useful for describing the R* distributed execution protocols. Only a few protocols described using this model were needed to represent the mechanisms of distributed execution in R*. Wide applicability, generality, and compactness are some features giving power to this model.

The used formalism is derived from the NUTT's evaluation nets; this formalism is in turn based on Petri nets, and seems to be particularly well adapted to representing the distributed execution procedures.

An evaluation net is composed of 3 kinds of elements:
- S: set of places, represented by a rectangle, denoting possible states of the system,
- R: set of requests, represented by a rectangle: these requests are mainly initialization requests (SQL), communication requests (COMM, respXXX), or responses after catalog lookup (FOUND, NOTFOUND, ...),
- T: set of transitions, represented by an horizontal line.

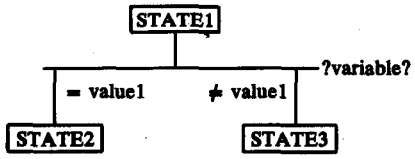
These elements are interconnected according one of the following schemes:



A marker called a token is placed in the current state; a diagram will contain one token per site currently running the procedure represented by the diagram. The state preceding the transition is called the input state; the state following the transition is the output state. A transition will be fired if there are tokens on all the states following the transition, unless the transition is a condition.

The actions described beside the transition are performed when the transition is fired. The transition can also express a condition; depending on the result indicated along the output branches, only the appropriate state will receive the token; this

facility will be used to control the progress of the computation, and the routing among the involved sites.



5.2. *C1RS protocol*

The C1RS protocol handles the statically oriented processing statements, those statements which involve only static entities whose location is known from the statement. C1RS stands for communication with at most one remote site. This case handles all the SQL statements related to dbspaces (acquisition and locking of dbspace, change in dbspace lock size or in percentage of free space) and special privileges (granting and revoking a special privilege such as DBA or resource authority to a user at a specific site). For the C1RS protocol, the site where the action (modification or insertion) on the catalogs should take place is either explicitly given in the SQL statement, or it is the query site (default option).

The formal protocol is presented in Fig.1 as an evaluation net. To explain how this protocol works, we will give an example of how it might be used for a user UGIVER at USERSITE who wants to grant the DBA privilege at site REMOTESITE to a user URECEIVER.

UGIVER tells R*: "GRANT DBA AT REMOTESITE TO URECEIVER".
R* calls the local routine grantdba (REMOTESITE,UGIVER, URECEIVER); this routine is thus activated at USERSITE.

```
procedure grantdba(siteparm,fromparm,toparm)
   fill in site defaults to this site;                    ** //
   if siteparm = this site                                ** //
      then do;                                              //
         check if fromparm can make the grant;              //
         if so, then record grant in catalog;               //
         else error;                                        //
      end;                                                  //
      else do;                                          **
         invoke grantdba(siteparm,fromparm,toparm)      **
               at siteparm;                             **
         check result;                                  **
      end;                                              **
   return to caller;                                    **
   end.
```

USERSITE executes the ** lines and REMOTESITE executes the // lines.

Note that this same procedure handles:

1. *locally* issued grants for DBA privileges at *this* site;
2. *locally* issued grants for DBA privileges at a *remote* site;
3. *remotely* issued grants for DBA privileges at *this* site;

Fig. 1 is an evaluation net for the C1RS protocol. In that figure, the asynchronous communication is represented by the label QWAIT, and the activation of the routine at the remote site by COMM(X→Y), where X is the sender and Y the name of the site at which the routine will be activated. The local variable cs verifies or indicates at which site the routine is currently being executed. A routine activated at a remote site terminates either with an error message (respERR), or is completed correctly (respOK). The routine at the QS can also end both ways: successfully with END, and unsuccessfully with ERRMSG sent to the user.

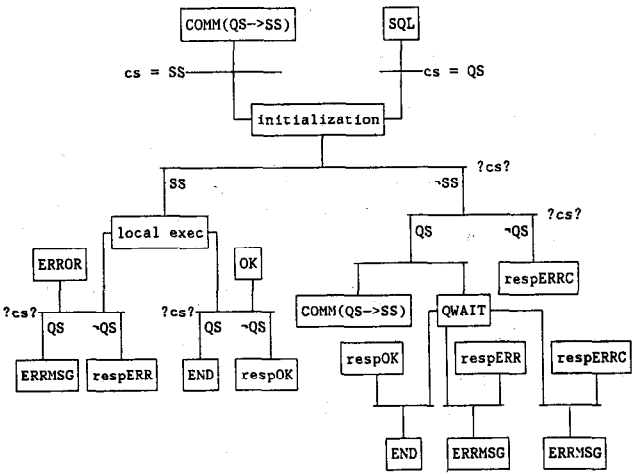


Fig.1: C1RS protocol.

5.3. *C3RS protocol*

The C3RS protocol handles the interpreted statements in the DOPS class. For these statements, the store sites of all dynamic entities must be found. In R*, each non DML DOPS statement has only one dynamic entity. The C3RS schema describes the communication protocol used by routines handling SQL statements which involve at most three remote sites. This includes most of the SQL data definition and control statements which have exactly one table as a parameter. For these statements (grant or revoke a privilege on a table, comment on a table or a column of a table, expand or lock a table, create or drop an index on a table), catalog modifications are made at only one site, even though the request may be processed at up to three sites. The catalog that will be changed is the one at the table store site (this is the site where the table is physically stored, which can be different from the site at which the table was born). The processing at the other sites consists of name mapping (discussed in section 2), checking of parameter validity and other checks, and determining the current store site of the table. The algorithm which determines the current store site involves a lookup in the local catalog (local FETCH), and if the supposed store site recorded in the local catalog is not here, a lookup in the catalog of that remote site. If no such record was found in the catalog at the QS (NOTFOUND), or if the lookup did not succeed at the supposed SS (respNF), the search for the current location of that table goes on (NEWSEARCH): a lookup in the remote catalog at the table birth site is performed (remote FETCH). The birth site either will be the actual store site or will provide the name of the current store site (and since a read lock is held on the birth site catalog, the actual store site location is guaranteed to be correct during this transaction). If the birth site was not the store site, a lookup in the remote catalog of the real store site is performed. After the actual store site has been found, a distributed recursive call is made to the actual store site and the catalog modifications are performed there.

The C3RS protocol first runs at the site (QS) where the query was issued by the user. If the catalog at the query site has no information about that table, then the first lookup to the presumed store site (SS) does of course not take place, and the search is directly performed at the table birth site (BS). Making a distributed recursive call to the supposed store site found in the QS catalog before asking the table birth site to tell us the real store site is justified by the assumption that most of the time the

tables will not have migrated. So the information contained at the query site about the table (if any) will not be obsolete. If this assumption is correct, communication will be reduced to only one remote site, the table store site. Lack of information about the table at the query site will result in messages to two remote sites (the birth site, and the store site). If these sites are identical, there will still be two messages to the birth site, one for remote catalog lookup, and one for the routine actually processing the statement. Finally, obsolete information about the table at the query site will be the only case requiring three messages to remote sites and when the actual store site location is retrieved from the catalog at the birth site and sent back to the QS, the obsolete cache information is replaced by the identification of the current store site (CLEARCACHE).

Local counters, (RF: remote fetch, FSS: fetch at store site), incremented at the QS guarantee that the process of determining table location terminates and hence that the request is eventually completed.

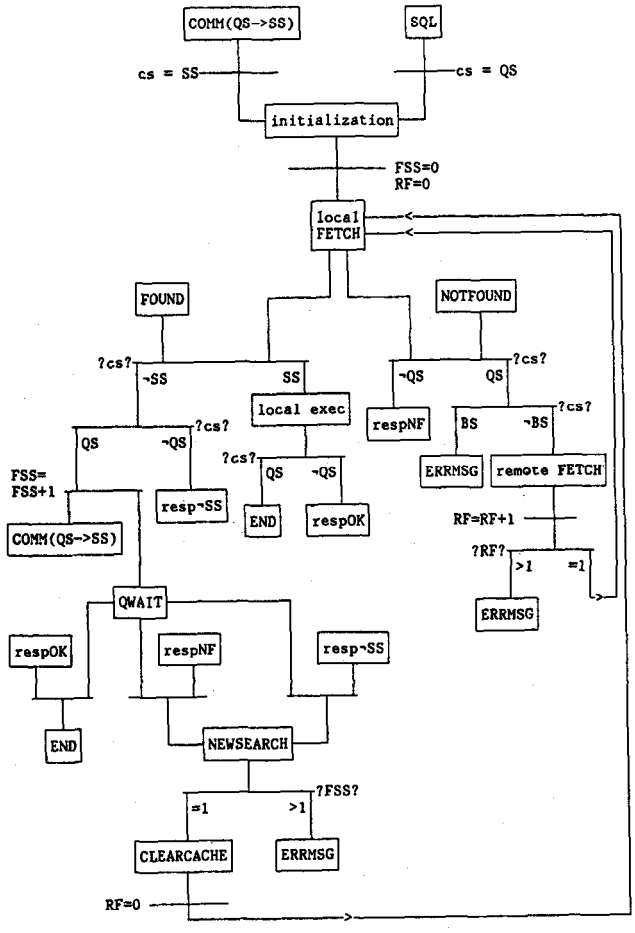


Fig.2: C3RS protocol.

6. Correctness

The formalization of the protocols by means of evaluation nets has been quite helpful in enabling us to visualize the complete behavior of the interpreted routines from the same class by means of one single evaluation net. This common diagram describes the checks and actions to perform whatever site is involved.

Tests have been written afterwards to be run against the designed protocols. All the different configurations of sites were tested for each interpreted routine. It has been verified that the expected route was followed for each case.

7. Conclusion

The provision of interfaces allowing the execution of the current statement level function to be transferred to another site have lead to an interesting and effective progressing methodology for implementing distributed functions in R*. Partitioning and fragmentation of the execution algorithm have been avoided. In addition, the statement level intersite interface has lead to a uniform mechanism for intersite interface checking. The use of the Nutt's nets graphical representation for the logic of multi-site execution has been useful for documenting and abstracting the intersite interactions and for classifying the algorithms used for different statement types.

The C1RS and C3RS protocols described here cover about 90% of the local DDS, DCS, and AMS statements. The evaluation nets models, plus the "I wish I were there" construct implemented by the distributed recursive call allowed us to implement DDS, DCS, and AMS for distributed objects in only a few months of one person's time. The implementation of statements which do not exactly fit the described protocols is easily designed using the evaluation nets and the "I wish I were over there" philosophy. Some more elaborate protocols are now being designed to handle complex distributed statements, like table migration, table replication and table partitioning.

8. References

[CHA81] D. Chamberlin, M. Astrahan, W. King, R. Lorie, J. Mehl, T. Price, M. Schkolnick, P. Selinger, R. Slutz, B. Wade, R. Yost, "Support for Repetitive Transactions and Ad-Hoc Queries in System R", *ACM Transactions on Database Systems*, March 81.

[DAN82] D. Daniels, P. Selinger, L. Haas, B. Lindsay, C. Mohan, A. Walker, P. Wilms, "An introduction to Distributed Query Compilation in R*", *Proc. of the Second Int. Symp. on Distributed Data Bases*, Berlin, Sept.82. Published in *Distributed Data Bases*, Schneider (editor), North-Holland.

[HAA82] L.M. Haas, P. Selinger, E. Bertino, D. Daniels, B. Lindsay, G. Lohman, Y. Masunaga, C. Mohan, P. Ng, P. Wilms, R. Yost, "R*: a research project on distributed relational DBMS", IBM Research Lab., San Jose, RJ3653, 10/21/82.

[LIN80] B.G. Lindsay, "Object naming and catalog management for a distributed database manager", IBM Research Lab., San Jose, RJ2914, 08/29/80.

[LIN82] B.G. Lindsay, L.M. Haas, C. Mohan, P.F. Wilms, R.A. Yost, "Computation & Communication in R*: A Distributed Database Manager", IBM Research Lab., San Jose, 12/08/82.

[NUT72] G.J. Nutt, "The formulation and application of evaluation nets", PhD dissertation, Univ. of Washington, 1972.

[WIL82] R. Williams, D. Daniels, L. Haas, G. Lapis, B. Lindsay, P. Ng, R. Obermarck, P. Selinger, A. Walker, P. Wilms, R. Yost, "R*: An Overview of the Architecture", *Proceedings of the Int. Conf. on Database Systems*, Jerusalem, June 1982.