Report 77-21
Stanford -- KSL

Scientific DataLink

Application of the Contract Net Protocol
to Distributed Databases.
Hector Garcia-Molina, Gio Wiederhold,
Apr 1977

HPP-77-21

APPLICATION OF THE CONTRACT NET PROTOCOL TO DISTRIBUTED DATA BASES

Hector Garcia-Molina and Gio Wiederhold[1]

Department of Computer Science
Stanford University
Stanford, California, 94305

Abstract

The contract net is a formalism which can be used to express the control of problem solving in a distributed computing environment. In this paper we apply the contract net formalism to develop a model for interaction in a distributed data base.

1 INTRODUCTION

In a distributed data base system, the data base is physically partitioned over several computing facilities while allowing integrated access to the data [2]. Each computing facility, which we will call a node, includes a process which is in charge of the management of some section of the data base. Each node is connected to the other nodes to allow the sharing of data.

We will assume that there is neither shared memory nor global data in the system. Except for this restriction, our model will not specify the interconnection architecture of the network. An important objective of our studies is in fact the development of measures of computational activities which will lead to evaluation criteria for architectural alternatives [10].

2 CONTRACTING

Using the contract net formalism[1], nodes communicate by using four different message types: contract announcement, bid, award, and report. When a node wishes to assign a task to another node, the originating node (the contract manager) will broadcast a contract announcement. This announcement describes the task and the resources needed for its execution. Nodes in the system will receive the broadcast and may make bids to the contract manager. Each bid contains

---

[1] This work was partially supported by the Advanced Research Projects Agency under contract DAHC 15-73-C-0435. Computer facilities were provided by the SUMEX-AIM facility at Stanford University under National Institutes of Health grant RR-00785. A number of useful suggestions were provided by K. G. Knutsen and R. G. Smith.

information regarding the capability the node has to perform the task. The contract manager receives and evaluates the bids. Then an award is made to one or more contractor nodes. An award contains detailed information about the task to be performed.

Each contractor node that receives an award is then responsible for the completion of the task. A contractor node in turn may generate subcontracts if the size or the data requirements of the original contract warrants it. When the assigned task has been completed, the contractor node sends a report to the contract manager. This report might contain a solution if the task required one, or a notification of an inability to obtain an answer.

Contract announcements do not necessarily have to be broadcast to all nodes. If a contract manager knows that only a certain subset of nodes have the ability to perform a task, then he may announce the contract via limited broadcast or via directed communications.

3 A CONTRACT NODE

Each node in the system is viewed as having two independent processors. The management processor is responsible for network communications, contract management, and bidding. The contract processor does the actual computation. In our case, the contract processor manages the section of the data base at the node. The contract processor might have several pending contracts at a given time. Therefore, the management processor is capable of examining the available contracts and choosing the most appropiate one for the contract processor to work on. Some of the factors for choosing a contract might be availability of resources or estimated time to completion.

A complete description of the contract net is given in [1]. Here we assume that the necessary underlying mechanisms are provided and available to our distributed data base system.

4 A DISTRIBUTED DATA BASE

The types of data bases we wish to model are based on the notion of distributed expertise [3]. We see a data base node as a collection of local knowledge, represented by data generated or utilized by local users and by procedures which include heuristics developed over time. Local use, analysis, and feedback provides incentive for the quality and comprehensiveness of the local data base which make the node a potential resource for other network participants.

For our initial evaluation of the utility of the contract mechanism to a distributed data base, we will assume that the same data model is used throughout the system. This eliminates the need for translating mechanisms between nodes. Furthermore, we will use the relational data model to simplify the presentation, but we expect the concepts can be applied to any data model. The relational model is chosen for its

simplicity and for the availability of theoretical results. For a complete description of the relational data model, see [4] [5].

Each node in the network will manage a set of relations. We will permit a single relation to be split across several nodes as long as the partition is controlled by one of the attributes of the relation. The distributed relation can then be represented by a set of non-distributed relations, each with a predicate condition.

To see how this is done, consider the following example. A relation EMPLOYEE:(NAME, DEPT, SALARY) is to be split up among the nodes located at each department. In this case we create the following relations:

EMPLOYEE | DEPT=shoe: (NAME, SALARY)
EMPLOYEE | DEPT=toy : (NAME, SALARY)

and finally the relation

COMPANY:(DEPT, ...)

which gives a list of all the departments. Relation COMPANY has become a catalog which defines how and into how many pieces the relation EMPLOYEE has been divided. Anyone accessing the EMPLOYEE relation must consult the catalog and then access the individual relations.

Most relations that have to be split will have some attribute (e.g. DEPT) that logically divides them. If there is no suitable attribute, then it probably is not a good idea (from the management point of view) to split the relation in the first place.

We will also allow redundant data in the system both for reliability and for performance. We will recognize two different objectives for redundant data:

1) Reliability. An entire relation may be completely duplicated in one or more backup copies. One of the copies, which will have special duties assigned, will be called the prime relation, while all others will be called the backup relations.

2) Performance. A certain group of attributes may be duplicated in several relations. For example, if employee Jones is also a manager, the component AGE = 53 might appear as Jones' EMP-AGE attribute in the EMPLOYEE relation as well as Jones' MANAGER-AGE attribute in the MANAGER relation. Among the duplicated attributes, there will be one prime attribute. The network node where the prime attribute is located will be in charge of locking or updating the secondary attributes. Notice that all backup attributes (i.e. attributes in a backup relation) are secondary attributes, but not vice versa.

These two objectives may be shared by having the data in the backup relation stored according to different access criteria than the prime relation. Thus the backup data can also provide faster access to certain types of searches.

5 DATA DESCRIPTION

Each relation will have associated with it a relation descriptor. This descriptor will contain:

1) A relation dictionary. The dictionary contains a

definition of all the keywords involved in the relation. A keyword is any word that has special meaning in the context of this relation. The dictionary will be used for understanding and parsing queries involving the relation.

The following types of keywords are included in a dictionary:

A) Relation name.

B) Attribute names.

C) Domain element names.

D) Synonyms. (Attributes and domain elements may have several names.)

For example, the dictionary for an EMPLOYEE relation might look like this:

```
EMPLOYEE --  "relation name".
NAME     --  "attribute name", principal key, domain is
             alphabetic unconstrained.
SS-NUM   --  "attribute name", secondary key, domain is
             numeric, range is 100 < SS-NUM <= 100000.
ID       --  "synonym" for SS-NUM.
SEX      --  "attribute name",domain is { male, female }.
male     --  "value" of SEX attribute.
m        --  "synonym" for male.
      etcetera.
```

2) The location. This is the address of the node where the relation is currently located. The address may change or may be unavailable at times.

3) Performance data. This might include the number of tuples in the relation or statistics for the utilization of the relation.

4) Protection information. This information describes who has which access privileges to specified attributes.

5) A set of attribute descriptors. Each attribute descriptor specifies the relationship of the attribute to other duplicate attributes. The contents of the attribute descriptor are discussed below.

6) Backup data. If the relation is a backup relation, then we must have the following:

A) The name and location of the prime relation.

B) The name and location of other backup copies (if any).

C) A copy of all the attribute descriptors of the prime copy. These are needed in case this backup relation has to become the prime copy.

Each attribute descriptor in the relation descriptor contains:

1) A flag indicating that the attribute is the prime attribute.

2) If the attribute is prime, then we need the following fields:

A) A LOCK-FLAG. This flag is set while the attribute is locked.

B) The LOCKER-NAME. This is the name of the node that has locked the attribute.

C) The EXECUTE-UPDATE-FLAG. This flag is set when an update operation has been started.

D) A list of the secondary attributes. Each entry in the list is of the form <Location, relation name, attribute name, procedure name, is-this-backup-attribute?>. ("Procedure name" is a procedure whose use will be explained later.)

3) If the attribute is not the prime attribute, then we just have a pointer to the prime attribute, i.e. <Location, relation name, attribute name>.

The collection of descriptive information in a data base node will be termed the internal schema. There is no global comprehensive global schema.

6 PROCESSING IN THE DISTRIBUTED DATA BASE

We will now describe how a user interaction is handled by the contract protocol. We will first take a case where all the data in the system are static, i.e. no updates take place.

The user enters his query at a network node and it becomes a contract for the local contract processor. The query does not necessarily contain a priori information giving the relation name [6] or the node name where relevant relations reside. A query which only involves local data will be answered directly and the result will be sent to the user. Since we are mainly interested in data base networks with intensive local node activities, we expect that most of the queries will be of this type.

If the interaction cannot be completely processed locally, then it must involve data from other nodes. The first step is to obtain the necessary domain knowledge from other nodes. The internal schemas provide the knowledge required to translate the query into a set of precise machine executable commands. The local processor will extract from the query all the terms it does not understand and will broadcast a contract announcement containing these terms. Other nodes on the network will examine the terms and if they find any of them in one of their internal schemas, they will return the relevant entries, the location of the relation, and the protection information in the form of a bid.

When the contract manager receives the bids, he will use the information in them to complete the query analysis. Having all the necessary information, the manager can generate the contract awards. The nodes that receive awards may in turn generate subcontracts if necessary.

From the bidding information, the contract manager will be able to decide which nodes are best equipped to solve the query (i.e. what nodes can respond faster, etc.). The manager can even decide to try processing the query in multiple ways. He will then issue contract awards to several nodes and will accept the first result that arrives. An option available to the manager is to issue an award to the node that

has the prime copy of a relation and also to the nodes that have the backup relations. Because each relation might be stored differently or because the computing loads at the nodes vary, one node will send the answer back before the others. This way the response time of the system can be improved. The contractor may also keep the bid information around for a time, in case other similar queries follow. This "working set" strategy has been suggested in [7].

The following example will illustrate the processing mechanism in the data base. Suppose a user asks the query

"FIND SALARY WHERE EMPLOYEE-NAME = FRED"

at node X which has no data on salaries. The node will broadcast a contract announcement containing the incomprehensible words "SALARY", "EMPLOYEE-NAME", and "FRED". It might receive the following bids:

From node A : I can give you SALARY if you give me SOC-SECURITY-NUM.

From node B : I can give you SALARY if you give me JOB-RATING.

From node C : If you give me EMPLOYEE-NAME, I will give you SOC-SECURITY-NUM, POSITION, etc.

Now node X can initiate one solution: Send a contract award to node C to get the social security number and then using this number issue another contract to node A to find the requested salary. While this solution attempt is going on, node X can attempt another approach. It broadcasts a contract announcement containing the word JOB-RATING and gets the following bid:

From node D : I can give you JOB-RATING if you give me POSITION.

With this information, node X can start up a parallel solution attempt: Issue an award to node C to get the employee's position, then one to D to get the job rating and finally one to B to get the desired answer.

7 PROTECTION

The protection restrictions should be considered by contract managers before they make awards. If the bidding contractor requires an access privilege greater than the one possessed by the contract manager, then the bid can not be accepted. The access privileges of the contract manager are the access privileges of the user that originated the request plus any special "local" privileges.

For example, suppose that at some point in an interaction a node needs to find the average salary in a given department. After broadcasting a contract with the words "SALARY" and "AVERAGE", it gets the bids:

From node A : I can give you SALARY given ... (Access privilege needed is P1.)

From node B : I can give you the AVERAGE given ... (Access privilege needed is P2.)

Since the user does not have access privilege P1 (i.e. the user can not look at individual salaries), but does have privilege P2 (i.e. the user can look at statistics, as for instance average salaries), the node awards a contract to node B. Node B has the privilege allowing it to issue a subcontract to node A so that it can calculate the average and return it to the user.

8 LOCKING REQUIREMENTS

Since in reality the data base is constantly being updated, we can not assume that the data is static. Therefore we will need a locking mechanism to ensure accurate results.

There are two modes of user interaction:

1) Free interaction. In this mode a user will only read data for informational purposes. For example, the user might be interested in obtaining the average salary of a set of employees. If a free read is used, some salaries may be in the process of being updated so that the average obtained will not be accurate. Most of the results obtained in a free read will be adequate and most users will work in this mode. Simple updates for attributes that are not duplicated and are not subject to audit may also be done in a free mode.

2) Locked interaction. For queries which have to stand up to audit and for updates which change attribute values subject to audit or which change duplicated attributes, it will be necessary to lock the data involved so that one process has exclusive access to it. The salary figures for the above example will require such a locked update, but a change in the employee's address (if the address is not duplicated) may be processed without such a restriction.

The desire to avoid locking is very strong in the type of distributed system we are hypothesizing. The cost of interlocks among multiple nodes is apt to be high since messages have to be sent over communication links.

8.1 Locked Reading

A free interaction is processed just as was described previously when we assumed that the data was static. For an audit read, we proceed differently since it is necessary to lock the data. As before, the contract manager node broadcasts a contract announcement with the keywords involved. This time the announcement indicates that a locked interaction is involved. The bidding nodes, in addition to returning the required parsing and protection information, also state which of the relevant attributes are prime attributes.

After decomposing the query, the contract manager must issue a series of contracts to lock all the attributes involved in the interaction. These contracts are awarded to the nodes containing the prime attributes. The contractor's report informs the contract manager whether it was possible to lock the attribute and all its duplicates. To prevent deadlocks, if any of the attributes can not be locked, then the contract manager must release all locks and try again at some later time. Once all the required attributes have been locked, the contract node can proceed with the query as before. Notice that it can award the contracts to read to any copy of an attribute since they are all locked. When the reading is completed, all the attributes are unlocked by issuing contracts to the prime attribute nodes.

When a node containing a prime attribute gets a contract to lock the

attribute, it first checks if the attribute is already locked or in the process of being locked. If that is the case, it returns a negative report. Otherwise it starts the locking process.

The first step is to look in the attribute descriptor and get a list of the backup attributes (duplicate attributes in a backup relation). Since the backup attributes might have to take the place of the prime attribute, the next step is to inform all the backup attributes that the attribute will be locked. This is accomplished by sending a contract to the backup relations to set the LOCK-FLAG and to save the name of the locking node in LOCKER-NAME in the copy they have of the prime attribute descriptor. Only when LOCK-FLAG and LOCKER-NAME have been set in all the backup relations, will the prime attribute node set them in the prime attribute descriptor. At that point the prime attribute and all its duplicates are locked. (The secondary attribute descriptors do not need to be informed that they are locked.) A positive report is then sent to the locking node.

When a request to unlock is received from the locking node, the lock is removed by following a similar procedure. There is a time-out mechanism that prevents an attribute from being locked forever.

8.2 Locked Updating

An interaction that involves locked updating requires that all referenced attributes are locked as was done in an audit read. The updating process is then done in two steps. The first step involves searching for the tuples to be modified; the second step does the actual updating. The search for the tuples to modify differs from a regular search in that the components that will be modified must be components of a prime attribute. (Recall that we only allow the prime attribute to be modified directly.) The result of the search will be a set of updates of the form <Tuple-id, attribute name,new value>, where tuple-id is a network wide address of a tuple and attribute name is the name of a prime attribute.

It is important that the first step is finished completely before the second one starts. The originating node will receive reports (possibly more than one) of the completion of the search. Once all the reports are received, the updates are directed to the nodes where the prime attributes are located.

When a node with a prime attribute receives a contract to update a set of tuples, it first checks that the attribute is actually locked by the contract manager. Then, if during the updating process it is instructed to unlock the attribute, it will delay the unlocking until all the updates have been completed.

The update contract specifies what tuples in the prime relation have to be modified. The tuple-id's have to be translated into tuple-id's for the secondary tuples that must also be modified. This translation will be done by the procedure associated with the link to the secondary attributes. For example, this procedure might produce a reference to the tuple MANAGER(SOC-SECURITY-NUM = 7752, DEPT = shoe, AGE = 55, ...) given the tuple EMPLOYEE(NAME = Smith, SOC-SECURITY-NUM = 7752,

AGE = 55, ...). If the age component in the EMPLOYEE tuple was being modified, then it would also be modified in the MANAGER tuple.

The corresponding set of translated update tuples is sent to each node having a secondary attribute. However, the nodes do not perform any updating until they receive an execute updates contract. Once the translated tuples have been sent and received at the nodes, the node with the prime attribute will set the EXECUTE-UPDATE-FLAG in the prime attribute descriptor copies in all the backup relations. Next it will set the EXECUTE-UPDATE-FLAG in its own prime attribute descriptor and will issue contracts to all the secondary attributes to actually execute the updates. Finally, it will update the components of the prime attribute itself.

When all reports of successful updates are received from the nodes with secondary attributes (including backup attributes), the prime copy node will report to its contract manager that the updating is completed and will reset all the EXECUTE-UPDATE-FLAGs.

9 RELIABILITY

A multinode network is attractive from a viewpoint of reliability [8], but this means that the interconnected processes cannot be permitted to affect each other in a disastrous maner. Let us now discuss what has to be done when a node in the network "dies".

Periodically, every network node will go through all its relation descriptors. For every backup relation it finds, it will issue a contract to the prime copy to check if that copy is still alive. If the prime copy does not respond, then the backup copy will try to take its place.

Since other backup copies may at the time be trying to become the prime copy, it is important to insure that only one copy does so. This can be achieved by having an independent lock mechanism by which a backup copy locks out all other backup copies. To prevent deadlocks, if a copy can not get all the locks it requests, it must give up all the locks it already holds and must start the process at a later time. (An alternative is to let all backup copies have predefined priorities for taking over as is done in the TYMNET control protocol [9].)

Once a backup copy succeeds in locking all other copies, it is ready to become the prime relation. It informs all copies that it is the new prime relation. It also makes sure that any operation that was in progress is finished. (Recall that the backup copy had an up-to-date version of the LOCK-FLAG, LOCKER-NAME, and EXECUTE-UPDATE-FLAG fields.) A specification for a detailed recovery protocol is in preparation.

10 CONCLUSION

In this paper we have presented a model for a distributed data base using the concept of contracts. We have only sketched some of the main features. Many of the problems we presented require additional work to

be solved completely; other problems yet remain to be addressed. For example, we must still face the problem of creating relations and copying them according to the load on the nodes. We must investigate further the locking and protection issues.

The contract net formalism seems to provide a suitable underlying framework for the distributed data base model in a loosely coupled network. When the model achieves a satisfactory state we will use the existing contract net simulator [1] to simulate some data base problems in order to evaluate architectural alternatives.

References

1. R. G. Smith, "The Contract Net: A Formalism for the Control of Distributed Problem Solving", Stanford Heuristic Programing Project, February 1977, submitted to IJCAI 1977.

2. M. E. Deppe and J. P. Fry, "Distributed Data Bases: A Summary of Research", in Computer Networks, Vol. 1, num. 1, March 1976, pp. 130-138.

3. D. B. Lenant, "Beings: Knowledge as Interacting Experts", IJCAI Proceedings, Tbilisi, USSR, September 1975, pp. 126-133.

4. C. J. Date, "An Introduction to Database Systems", Addison-Wesley Publishing Co., 1976

5. G. Wiederhold, "Database Design", McGraw Hill Book Co., 1977.

6. S. Chang and J. S. Ke, "Database Skeleton and its Application to Fuzzy Query Translation", Department of Information Engineering, University of Illinois, November 1976.

7. M. Stonebraker and E. Neuhold, "A Distributed Data Base Version of INGRES", Electronics Research Laboratory, University of California, Berkeley, September 1976.

8. F. Heart, S. Ornstein, W. Crowther, and W. Baker, "A New Minicomputer/Multiprocessor for the ARPA Network", AFIPS NCC 1973, pp. 529-537.

9. L. R. Tymes, "TYMNET Control Techniques", Presented at the Second Annual Communications Conference at California State University at San Jose, Jan 24-25, 1973.

10. K. Thurber and L. Wald, "Associative and Parallel Processors", Computing Surveys, Vol. 7, num. 4, December 1975, pp. 215-255.