EVALUATION OF DISTRIBUTED CRITERIA FOR DISTRIBUTED DATA BASE SYSTEMS

by

D. Ries and R. Epstein

Memorandum No. UCB/ERL M78/22

12 May 1978

EVALUATION OF DISTRIBUTION CRITERIA FOR DISTRIBUTED DATA BASE SYSTEMS

by
Daniel Ries
Robert Epstein

Memorandum No. UCB/ERL M78/22
May 12, 1978

Electronics Research Laboratory
College of Engineering
University of California, Berkeley
94720

EVALUATION OF DISTRIBUTION CRITERIA FOR DISTRIBUTED DATA BASE SYSTEMS

by
Daniel Ries
Robert Epstein
(University of California at Berkeley)

Abstract

In this paper we examine three distribution criterion and copy mechanisms within a distributed database environment. The three mechanisms are: free control, user restrictive control, and user permissive control. Each mechanism is evaluated with respect to user convenience, performance, concurrency control, and ease of implementation.

We show that the concepts of primary fragment and primary tuple are extremely important. Moreover, we demonstrate that the user restrictive mechanism is most desirable since it provides primary fragments and primary tuples. It also supplies a separate copy mechanism which can be changed independently of the distribution criterion.

The ideas expressed in this paper are being incorporated in the distributed database version of INGRES.

Research sponsored by the U.S. Army Research Office Grant DAAG29-76-G-0245, and the Joint Services Electronics Program Contract F44620-76-C-0100.

Distribution Criteria

I. INTRODUCTION

This paper is concerned with the specification and maintenance of distribution criteria for data in a distributed database. A mechanism must be provided which is simple enough to understand and yet powerful enough to allow considerable flexibility in the data distribution. Ideally, the Data Base Management System should assume as much of the burden as possible.

This paper is motivated by the fact that distributed databases are becoming increasingly important as the volume of data continues to grow and the cost of independent computing resources continues to decrease. In a distributed database, the distribution of the data and its copies greatly effects the performance of the system. In determining the data distribution in a computer network, one should attempt to make as many query transactions as possible run locally. Furthermore, such locality must be properly balanced for efficient updates and retrievals.

In the next section a sample environment and a data manipulation language for a distributed database are discussed. In section III, three alternative mechanisms for specifying the distribution control are presented. In section IV, these methods are analyzed with respect to performance considerations, concurrency control, ease of use both by the database administrator and the user, and implementation considerations. In the last section, several conclusions are summarized.

Several simplifying assumptions are made concerning reliability and recovery. First, the maintenance of distributed copies for recovery or resiliency is not considered. This issue is discussed in [STON78]. An assumption is made that because the system is operational enough of the time, the distribution criteria mechanisms should be optimized towards normal transaction processing at the expense of recovery costs.

Considerable work has been done on the optimal solutions to the distribution of data for a given set of transactions [CHU69, CHU76]. These studies report on the cost-performance tradeoffs in distributed systems for directories

or system catalogs and optimum file allocations. However, these studies are not concerned with how a user specifies the data distribution. They also do not consider the implications of such a specification on the mapping of logical transactions to the actual physical distribution of the data.

II. DATA BASE ENVIRONMENT

In this section the facets of the the distributed database environment that affect the use and specification of the distribution criteria are discussed. Topics include distributed relations and their copies, distributed query processing, concurrency control, and aggregation. The relational model of data [CODD70] and the query language for INGRES, QUEL [HELD75] will be used to discuss these issues.

Distributed Relations

It is assumed that relations are either local or global. Local relations are known only to their own site and cannot be accessed outside that site. Global relations are known to all sites and can be stored at any number of sites. The portion of a relation at a particular site will be referred to as the fragment of the relation at the site. Fragments are subrelations, i.e. a subset of the tuples of a given relation, and not projections of a relation [ROTH77].

Some sample QUEL commands are now given.

```
create global savings (acct = i4, name = c20,
        balance = i4, branch_code = i2)
```

This command creates a global relation called "savings" with four attributes: acct, name, balance, and branch_code. This relation can be queried, for example, by asking for a list of all accounts and balances:

```
range of s is savings
retrieve (s.acct, s.balance)
```

Notice that the user need not be concerned with the details of whether "savings" is local or global, or if it is global where the fragments are stored.

To help the database system localize queries, a duplication mechanism can also be provided. A user can specify that a copy of one or more fragments of a particular relation be created. For example:

```
range of s is savings
```

```
duplicate s from (sf, berkeley) onto headquarters
where s.balance > 50000
```

The interpretation of this command is to make a copy of those accounts at sf and berkeley with balances over 50000 and place it at the site called "headquarters". The "where" clause is optional. (In fact, the addition of the where clause may complicate the use of copies for recovery purposes.) The system will automatically update the copy and use it whenever it is advantageous to do so.

Query Processing

There are four types of queries in QUEL: retrieve, append, replace, and delete. The update commands (append, replace, and delete) are run as retrieve commands which discover what must be done, followed by low level update processing.

In order to understand the effects of a distribution criterion, the processing of queries is first discussed. (A detailed explanation of distributed query processing for INGRES can be found in [EPST78].) The way to process a simple retrieve query involving only one relation is relatively straightforward. The query is broadcasted to all relevant processing sites and the results are gathered up and returned to the originating site.

A query involving more than one relation (called a multi-variable query) can be processed by a combination of local queries and moving fragments of the relations between the various sites. An example should serve to illustrate how this can be done.

Suppose there are the following relations:

```
transact(acct, date, teller, amount, code)
savings(acct, name, balance, branch_code)
```

Assume that both relations are distributed among many different sites. Now if the query:

```
range of s is savings
range of t is transact
retrieve (t.amount, t.code, s.branch_code)
```

```
where t.acct = s.acct
and t.date = "78-01-01"
```

were given it would be potentially necessary to examine every combination of t.acct and s.acct looking for equality. Here is one possible way the processing would proceed.

(1) Each site is directed to run the query:

```
retrieve into temp(t.amount, t.code, t.acct)
        where t.date = "78-01-01"
```

In the new relation "temp", all the tuples have the correct date.

(2) Every site which has a fragment of "temp" sends a copy of its fragment to every site which has a fragment of "savings".

At the end of this step, every site which has a fragment of "savings" will have an identical, complete copy of "temp".

(3) Every site which has "savings" now runs the following query.

```
range of t is temp
retrieve (t.amount, t.code, s.branch_code)
        where t.acct = s.acct
```

(4) The results are accumulated and forwarded to the originating site.

Notice that step (2) required data movement in order to process the query. Either "temp" or "savings" could have been moved. An attempt can be made to limit the amount of data which must be moved by taking advantage of the distribution criterion. How this can be done will be explained shortly.

Concurrency Control

The concurrency control is responsible for ensuring the consistency of the database during simultaneous updates. In section IV, it is shown how the distribution criterion can limit the available alternatives for concurrency control.

Two types of consistency must be considered in a distributed database. First, each transaction should be presented with an internally consistent view of the data. This view must obey certain global consistency requirements that may be placed on the database. For example, one such consistency requirement may be that a department salary budget must equal the sum of the salaries of the employees in that department. A transaction which might not see a consistent view would be a general report which included some of a set of related updates without including all of those updates. In [ESWA76], it is shown that a sufficient condition for maintaining a consistent view is that the effects of processing multiple concurrent transactions be equivalent to the effects of processing the transactions in some sequential order.

Many of the mechanisms for enforcing this type of consistency can be divided into two areas: primary site control and decentralized control. Under primary site control, one site can be responsible for granting access to the global relations. Under decentralized control, each site grants access to data stored at that site. Several alternatives have been proposed for resolving or avoiding deadlocks in the decentralized case. These include preordering transactions and/or sites [ROSE77], transferring requests to all sites [ELLI77], or transferring conflict information to one site [STON78].

A second consistency consideration is that copies of the data should be kept mutually consistent. For example, the same transaction run at different sites may use different copies of the same data. The concurrency control may be required to ensure that the same results will be produced for the same transactions run with different copies of the same data.

The mutual consistency of copies can be ensured by locking strategies similar to those that ensure internal consistency. However, even with decentralized control, locking for copies can involve the designation of a primary copy to which all updates must first be sent. The relation fragment at that site is called the "primary" fragment. The tuples in that fragment are the "primary" tuples for all of the duplicate copies. Another alternative with decentralized control is to associate timestamps with each data field and each update. An update and the timestamp is sent to all copies of the data. An update for a given copy is only performed if the timestamp for the data field is older than the timestamp of the update. If the updates for a given field of a given relation quiesce, all copies will eventually obtain the same value.

Aggregation

There are, in general, two classes of aggregates. In QUEL they are referred to as regular (count, sum, avg, min, max), and unique (countu, sumu, avgu). Regular aggregates use each tuple in a relation. (In the relational model, only one occurrence of a given tuple is allowed.) Unique aggregates are performed by first removing any duplicate values in the set of domains being aggregated. The distribution mechanism affects the ability of the database system to efficiently process aggregates.

Here are two example aggregates:

```
range of t is transact
retrieve (num_accts = count(t.teller),
          num_tellers = countu(t.teller))
```

The first "count" aggregate would simply count the number of tuples in transact. The second aggregate, "countu", would count the number of unique values for t.teller, giving the number of different tellers involved in the transactions.

III. DISTRIBUTION CRITERION AND COPY MECHANISM

As already mentioned, one goal of distribution criterion and copy mechanisms is to improve the performance of the database system. It accomplishes this by making queries as local to their originating site as possible. In this section three general approaches are discussed: (1) free control, (2) user restrictive control, and (3) user permissive control.

(1) Free Control

With this method, the database system has complete control of where tuples are physically located. New tuples are appended wherever it is "cheapest" to place them. The user can optionally create a duplicate of a fragment using the "duplicate" command discussed earlier.

The implications are that the user does not care where tuples are stored. The database system will take advantage of duplicate copies whenever possible.

(2) User Restrictive Control

User restrictive control restricts any tuple to one, unique site. Tuples are partitioned onto different sites according to a user supplied criterion. The user gives an ordered list specifying where tuples belong. The tuples are then placed at the first qualifying site in the ordered list. The user can create copies by using the "duplicate" command. Here is an example distribution criterion:

```
range of s is savings
distribute s at
        headquarters where s.balance > 50000,
        berkeley where s.branch_code = 2,
        albany where s.branch_code = 3,
        sf where s.branch_code = 1
```

The meaning here is that accounts with more that 50,000 will be physically kept only at headquarters. Otherwise the branch_code will determine which site (or no site) a tuple belongs to. Notice that a tuple with a branch_code of 2 and balance of 100,000 would go to headquarters and not to

berkeley.

(3) User Permissive Control

User permissive control maps any tuple to as many sites as are permitted. The user provides an unordered list of which tuples should be associated with which sites. The database system guarantees that a tuple will be placed at every site where it is allowed. This type of control combines the concepts of distribution criterion and copies into one homogenous method. Thus the "duplicate" command is unnecessary. Here is an example:

```
range of s is savings
distribute s at
        berkeley where s.branch_code = 2
                or s.branch_code = 3
        albany where s.branch_code = 3,
        headquarters where true,
        sf where s.branch_code = 1
```

Here "savings" is allowed at four sites. Assume that berkeley is branch code 2, albany is branch code 3, and sf is branch code 1. This criterion specifies that sf and albany get their own accounts, berkeley gets its own and all of albany's, and headquarters gets everything. A tuple with branch_code = 3, for example, would thus reside at three different sites.

IV. ANALYSIS

The main differences between free control and user restrictive control are twofold. First, it is possible that 'unknown' duplicates can exist at several sites under free control. For example, the same tuple could be added at two different sites. Scanning every fragment for potential duplicates in a distributed database would be prohibitively expensive. Under user restrictive control each tuple has a unique assigned site. The second difference is that restrictive control allows the system to restrict the number of sites which need to be accessed for a given transaction. Under free control it must be assumed that all sites are needed for a query.

The user permissive control can also assist the system in restricting sites for transaction processing. The main difference between permissive control and restrictive control is that the latter designates a "primary copy" for a relation fragment. Another difference is that user restrictive control has a separate mechanism for specifying the distribution criterion and the duplication policy. The user permissive control combines these specifications into one mechanism. Both mechanisms have equal power in terms of describing distribution and duplication criteria.

In this section, each of the distribution criterion mechanisms are evaluated in terms of implementation, retrieval, updating, aggregation, concurrency control, and ease of use. A summary of our analysis is shown in table 1.

Free Control

The free control distribution criterion is the simplest to implement and is the minimum facility needed to support the distributed database environment described in section II.

Under free control, the operations required to support updates are straightforward. For a tuple originating from or modified at a given site, the relation is updated at that site. If copies are required at other sites, the originating sites also issue the copies.

Distribution Criteria

Table 1

	Free Control	Restrictive Control	Restrictive Control
implementation	easy	hard	hard
retrieval	must scan all sites	restricts sites to scan	restricts sites to scan
updating	simple	complicated	complicated
aggregation	not well defined	easy, same as single site	difficult
concurrency	cannot have strict locality	can have strict locality	can have some strict locality
user friendly	best	fair	good

For retrievals, all sites are originally potential candidates for required tuples. The number of sites required for the retrieve can then be reduced only by considering available copies. Once the candidate sites have been selected the retrieve proceeds as described in section II.

In processing aggregates, the 'unknown' duplicate tuple condition makes the semantics of sums, averages, and counts unclear. For example, an identical tuple with Jone's salary could be added to two different sites and be included twice in the calculation of an average salary. If unique aggregates are requested, the domain(s) in question can all be assembled at one site, duplicates removed and then aggregated.

Another problem caused by having all sites as potential candidates for the required tuples is that remote sites must be included in the concurrency control synchronization in order to ensure the serializability discussed in section II. If one primary site is designated to handle all concurrency

control, transactions at that site may not require network communications. Transactions at other sites must check with the primary site. With decentralized control, the concurrency control requires that all transactions communicate with other sites. Thus, the primary advantage of free control, locality of updates, may have to be sacrificed.

Restrictive Control

Restricted control is more difficult to implement than free control due to the movement of tuples on updates and more complex site selection algorithms for retrievals.

Under restrictive control, updates may be more expensive than with free control since the placement of the new tuples is predetermined by the distribution criterion. In fact, a simple replace on a domain referenced in a distribution criterion might require the corresponding tuple to be moved.

For retrieval, both duplicate copies and the distribution criterion can be used to eliminate sites from consideration. This reduction of applicable sites would require a propositional calculus theorem prover in order to determine from an arbitrary distribution criterion, that an arbitrary query does not require certain sites. However, it is frequently possible to do simple checks such as those proposed for predicate locking [ESWA76, STON75]. One starts using all sites and eliminates sites which do not contain relevant data.

Similarly, only the required sites need be involved in the concurrency control decisions if decentralized locking is used. The responsibility for maintaining mutually consistent copies is given to the site with the primary fragment.

Some transactions can be made to run completely locally. These transactions are the ones which use only primary relation fragments stored at one site. This is extremely important since it is reasonable to expect completely local transactions to run faster than transactions which must communicate with other sites.

The processing of aggregates is well defined under restricted control and rather straightforward to implement since it is easy to ensure that only one copy of a tuple is used.

Permissive Control

Under the permissive control distribution criterion, updates and retrieves can be implemented in much the same way as in restricted control. The distribution control can be used to determine the applicable sites for a given query. No separate duplicate mechanism for copies of portions of the database is available (or needed) to further reduce the applicable sites.

Note that with permissive control, one looks for the minimum set of sites which covers the solution to the query. There may be many different sets of sites which cover the solution and a mechanism must be developed for determining which set of sites to use.

The implementation of aggregates is much more difficult. The number of copies of each tuple must be checked by mapping the tuple against the distribution criterion and making the appropriate arithmetic adjustments. One possible algorithm would be to have each site check every tuple against the distribution criterion. If the tuple belongs only to that site, then use it. Otherwise, assume there exists some global site ordering and use the tuple only at the first site where the tuple occurs. While we are not necessarily recommending this algorithm, we point out that any method of processing aggregates must be both complex and expensive.

For concurrency control, the permissive distribution criterion resembles the free control case in that a given primary site for a tuple is not known. All copies of a needed fragment must be locked either through communications with all sites or with a primary site determined by some means other that the distribution criterion (e.g. a fixed ordering of the sites). The flexibility of the concurrency control is limited in that one cannot necessarily have different primary sites for different relation fragments.

Distribution Criteria

A primary site designation for each relation could be added to the syntax of the permissive control. This would most likely make the syntax dense and confusing. The primary tuple for multiple copies of tuples would then be at the primary site. However, the distribution and number of copies could not be changed independently.

V. CONCLUSIONS

Free control is useful for temporary relations created by the system, for example the "temp" relation in section II. It requires no decisions on the part of the user in exchange for potentially poor performance. This is because it provides no help for limiting the scope of a query. This inability, together with the poor aggregation capabilities makes free control a poor choice for permanent user relations.

Restrictive control provides better performance for retrieval than does free control. Since updates are processed as a retrieve followed by the actual update, any improvement in retrieval will help updates as well. We can provide much more locality because fragments can be eliminated using the distribution criterion.

Restrictive control yields the most flexibility in choosing from concurrency control alternatives. Since fragments are associated with the values taken on by their tuples, it is possible to have a query which is completely local and requires no network locking.

Permissive control is the most elegant. The copy mechanism is the same as the distribution control mechanism. If there is no good partitioning, permissive control is probably easier to use than restrictive control. For example, a decision to store accounting data both at a headquarters and the appropriate branch can be specified without deciding on which site should have the primary fragment.

However its elegance is not without cost. It never provides more efficient processing than restrictive control and there are many situations where it is substantially worse. In addition, since there is no notion of a copy vs. the real tuples, aggregation is very complex and messy.

Both restrictive and permissive control have the anomaly that a tuple might not be accepted on any site. For example, with the distribution criterion specified in section III under user restrictive control, a tuple with branch_code = 4 and a balance of less than 50,000 could not reside at any site. Ideally this shouldn't happen since if the user desired to restrict certain values, an integrity

constraint mechanism would have been used. It is not clear what should be done if a tuple cannot reside at any site given in the distribution criterion.

Both the restrictive and permissive methods require that the site selection be inferred from the tuples. This may require the database designer to force the addition of another domain to the tuple, or encode site information into the distribution criterion. For example, the "savings" relation defined in section II is distributed at four sites: headquarters, berkeley, albany, and sf. The association that berkeley is "branch_code" number two, albany is "branch_code" number three, etc. is encoded in the distribution criterion.

The principle reason why restrictive control gives the best performance is that a good partition was chosen. If there is no good partitioning (i.e. one cannot say what sites will query what data) then the user may be forced to place the data at a single site, and duplicate the data at other sites.

The fact that the restrictive control is the only way to efficiently provide a "duplicate free" environment should not be overlooked. That is, one can efficiently guarantee that no tuple is duplicated on any other site.

Upon reviewing the ramifications of the three distribution criteria approaches, a restricted control where tuples are partitioned according to a user supplied criterion is preferred for several reasons. First, it provides the concept of primary tuples, which is extremely important for processing aggregates and for concurrency control. Second, a copy mechanism separate from the distribution criterion allows the concept of a primary fragment and thus the highest possible locality. In addition, the separate copy mechanism allows one to be changed without redefining the other.

One area for future research is how the database management system can choose the distribution criterion based on access patterns. This could be part of an automatic restructuring process which could also create (or destroy) copies. The difficulty of knowing how to do such restructuring is, of course, enormous.

REFERENCES

[CHU69] Chu, W.W.; "Optimal File Allocation in a Multiple Computer System", IEEE Transactions on Computers, vol. C-18, no. 10 October 1969.

[CHU76] Chu, W.W.; "Performance of File Directory Systems for Data Bases in Star and Distributed Networks", NCC Conference Proceedings, vol. 45, 1976.

[CODD70] Codd, E.F.; "A Relational Model of Data for Large Shared Data Banks", CACM vol. 13, no. 6, June 1970.

[ELLI77] Ellis, C.A.; "A Robust Algorithm for Updating Duplicate Databases", 1977 Berkeley Workshop on Distributed Data Management and Computer Networks, Lawrence Berkeley Laboratory, May 1977.

[EPST78] Epstein, R.; Stonebraker, M.; Wong, E; "Distributed Query Processing in a Relational Data Base System", SIGMOD Conference Proceedings, May, 1978.

[ESWA76] Eswaran, K.P., Gray, J.N., Lorie, R.A., Traiger, L.I.; "On the Notions of Consistency and Predicate Locks in a Database System", CACM vol. 19, no. 11, November, 1976.

[HELD75] Held, G.D.; Stonebraker, M.R.; Wong, E.; "INGRES - A Relational Data Base System", Proc. NCC vol. 44, 1975.

[ROSE77] Rosenkrantz, D.J., Sterns, R.E., Lewis, P.M.; "A system Level Concurrency Control for Distributed Database Systems", 1977 Berkeley Workshop on Distributed Data Management and Commputer Networks, Lawrence Berkeley Laboratory, May 1977.

[ROTH77] Rothnie, J.B.; Goodman, N.; "An Overview of the Preliminary Design of SDD-1: A System for Distributed Databases," 1977 Berkeley Workshop on Distributed Data Management and Computer Networks,

Lawrence Berkeley Laboratory, May 1977.

[STON75] Stonebraker, M.R.; "Implementation of Integrity Constraints and Views by Query Modification", University of California, Electronics Research Laboratory, Memorandum ERL-M514, March 1975.

[STON78] Stonebraker, M.R.; "Concurrency Control and Consistency of Multiple Copies of Data in Distributed INGRES", University of California, Electronics Research Laboratory, (To appear.)