A DISTRIBUTED DATA BASE VERSION OF INGRES

by

Michael Stonebraker and Eric Neuhold

Memorandum No. ERL-M612

11 September 1976

ENGINEERING RESEARCH LABORATORY

College of Engineering
University of California, Berkeley
94720

A DISTRIBUTED DATA BASE VERSION OF INGRES

by

Michael Stonebraker

Electronics Research Laboratory
University of California, Berkeley

and

Eric Neuhold

University of Stuttgart

ABSTRACT

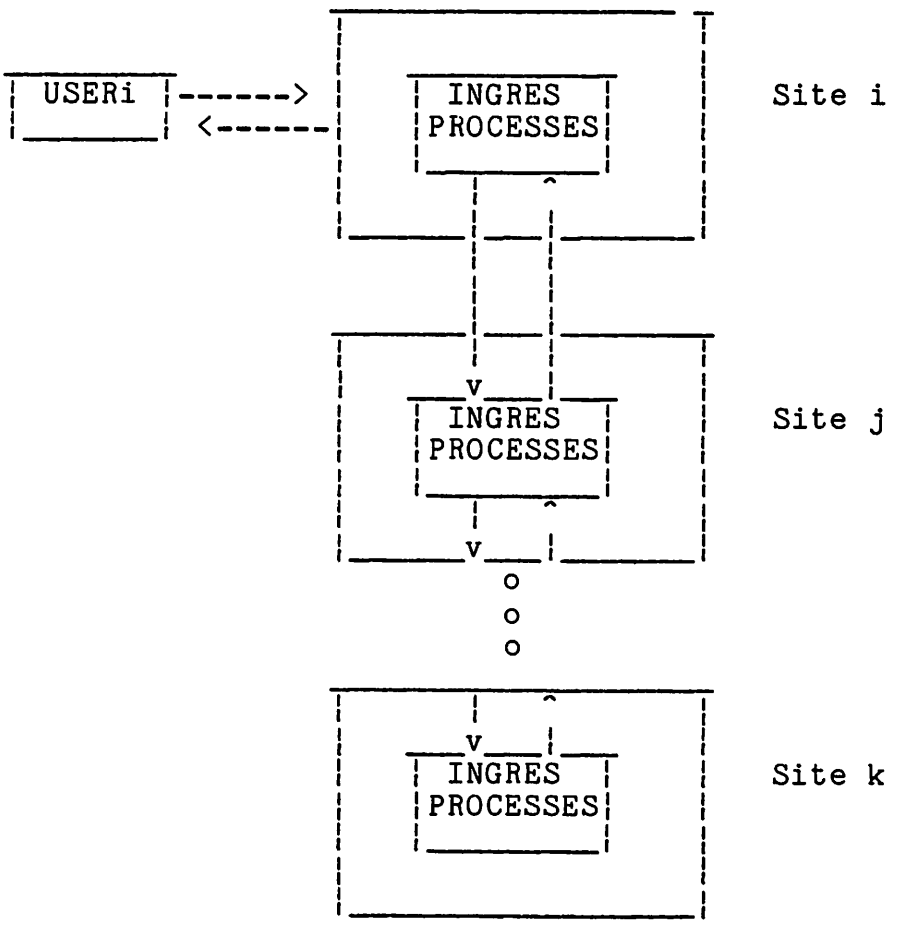
This paper sketches the extensions to the currently operational INGRES data base systems which are required for it to manage a data base distributed over multiple machines in a computer network. The machines are assumed homogeneous (or at least composed of machines each running the UNIX operating system).

Three possible user views of a distributed relational data base are presented. Each is readily seen to be a special case of the subsequent one. The difficult extensions and/or modifications to the code of the currently operational INGRES system are suggested. Lastly, the view being implemented and the reasoning behind its choice are indicated.

This research was sponsored by the Army Research Office Grant DAAG29-76-G-0245 and the National Science Foundation Grant DCR75-03839.

I INTRODUCTION

INGRES is a relational data base management system [STON76, HELD75] which operates as a collection of user processes on top of the UNIX [RITC74] operating system. This DBMS is currently being extended to operate on a collection of computer systems each running UNIX. In this paper we indicate the extensions and modifications which are required for the new environment. Throughout the paper we assume the existence of the UNIX to UNIX communication facility being constructed by the UNIX designers [THOM76]. This facility allows a process on one machine to "fork" slave processes on another machine and interchange data with such processes. Consequently, a user can invoke INGRES at one machine and interact with the INGRES processes running at that site. Moreover, the INGRES initiation program can "fork" a slave collection of INGRES processes at each site in the network. These collections can communicate with each other through UNIX network support code. This capability is illustrated in Figure 1.



The Distributed INGRES Environment
Figure 1

It should be noted that each INGRES user will have such a collection of processes on each machine. The processes on a single machine share text segments so the core overhead of multiple collections of INGRES processes is only that of data segments. Moreover, much of the time the processes will be inactive and will reside on a secondary storage device.

The sketch presented can apply to any other distributed computer support facility which has the above mentioned capabilities such as the ARPANET [ROBE70, CHES75], TELENET [KARP76], etc.

This paper does not consider problems of placement of data in a

distributed computing environment [LEVI75, CASE72, WHIT70, CHU69] nor does it consider physical design problems such as choice of line speeds, etc. Only implementation issues are discussed.

II THE THREE VIEWS OF A DISTRIBUTED DATA BASE

On a single machine INGRES manages a collection of data bases each with a given data base administration (DBA) who has powers not available to normal users. Each data base consists of a collection of relations R1,...,Rn, plus system catalogs [STON76]. The computer network consists of sites S1,...,Sk interconnected by communications links.

A user at a given site Si "logs into" INGRES and indicated which data base, D, he wishes to interact with. The view of the network which he sees is the following. Suppose data base D exists at a subcollection of the sites S1,...,Sm, at the j-th site Sj are relations R1(Sj),...,Rn(Sj).

The user with view number 1 sees a collection of relations C={Ri(Sh), h = 1,...,j, i = 1,...,m} with the following restrictions:

1. Each relation is on a unique machine i.e. no two relations in C have the same name

2. A user interaction with the data base can only span relations at a single site

Note that the user presented with such a "view" need not necessarily know where a given relation is physically placed. In

fact, except for performance, he cannot tell that the collection C is not all on the machine to which he "logged into".

In view 2 restriction number 2 above is dropped while in view 3, both restrictions are absent. We illustrate each situation with an example. Suppose EMP(NAME, SALARY, MANAGER, AGE, DEPT) and DEPT(DNAME, FLOOR#, LOCATION, SALES) are two relations. First suppose EMP is on machine 1 and DEPT is on machine 2. With view 1 a user logged onto either machine sees a data base consisting of both relations; however he can interact only with the data at a single site in one interaction. Consequently the interaction "Find the names of employees on the first floor" i.e.

```
RANGE OF E IS EMP
RANGE OF D IS DEPT
RETRIEVE (E.NAME) WHERE E.DEPT
        =D.DNAME AND D.FLOOR# = 1
```

would not be allowed. However, with the second view such a query would be executed even though it spans more than one machine. With the third view portions of DEPT and EMP could be on each machine. For example machine 1 might have all tuples from DEPT where LOCATION = 1 and machine 2 would have the remainder of the relation. Moreover, an employees tuple could similarly be on the machine having data for his location. With this view each relation may be physically distributed.

Note that all three views differ fundamentally from the approach taken by the data computer [MARI75] project. There a large centralized facility is provided.

The consistency condition determining the correctness of an interaction, I, for each of the views is the following: Suppose the collection of relations C is assembled on one machine; relations with the same name being merged. The interaction, I, can be processed against this data base and yields a result defined in [HELD75]. This result must be the identical to one produced by first applying the interaction to the distributed data base then assembling the relations.

Regardless of which view is supported the need for copies of relations may arise. Two reasons are commonly given for the existence of copies:

1) reliability

If a machine fails a backup copy of a relation on another machine may be used.

2) performance

In environments where there is a large percentage of retrieve requests, higher performance may be obtained by directing such retrievals to the "closest" copy.

In either case the copies must be kept current. In a later section we will see that how copies should be updated will depend on whether they exist for reason 1 or reason 2.

Lastly, regardless of which view is supported and whether copies of relations exist, the network can be run with varying amounts of control centralized in a single machine. We shall refer to

this decision as whether "GOD" should exist and if so what powers should he have. It will be seen that the existence of "GOD" is a very fundamental question.

In the next four sections we examine implementation problems in a network environment which are present for some or all of the views. These problems are grouped into four categories:

1) New user commands needed
2) Problems with storage of system catalogs
3) Decomposition and query processing problems
4) Concurrency and consistency problems

We examine each in turn.

III NEW USER COMMANDS

For all three views the following user commands must be added to the INGRES user language [ZOOK76].

1) The command language will be expanded to include a keyword "LOCATION" by which a user can indicate his knowledge (or desire) concerning the physical location of a relation.

For example, one RANGE declaration would be the following:

```
RANGE OF E IS EMPLOYEE (LOCATION = 1)
```

In this case the user is only interested in the employee relation if it is at site 1 (or the portion of it present at site 1).

Similarly,

CREATE NEW_EMPLOYEE (LOCATION =2) (NAME =C10, AGE = I2, SALARY = I4)

indicates a desire to create a relation NEW_EMPLOYEE at site 2. Other commands are similarly extended in a straightforward manner.

2) It must be possible to move a relation from one site to another. This requires a command:

MOVE RELATION_NAME(LOCATION=X) TO RELATION_NAME(LOCATION=Y)

3) If view three is adopted there are three possible ways to form a relation which spans more than one machine.

One sequence of operations might be:

```
RANGE OF R IS RELATION
RETRIEVE INTO W(R.ALL) WHERE QUALIFICATION
DELETE R WHERE QUALIFICATION
MOVE W TO RELATION (LOCATION = X)
```

This sequence of steps moves the tuples from RELATION which satisfy QUALIFICATION from the machine they are currently on to the machine at LOCATION X.

Alternately, CREATE can be applied at several sites and a bulk load (using the COPY facility) done for each machine.

Both of the above mechanisms create a relation which is distributed according to a user defined criteria. A user who wishes to utilize location information to speed processing must build it

into the transactions he writes. Moreover, he must guarantee that the distribution criteria which he is using remains valid. For example, if machine 1 has employees with salaries under 10000 and machine 2 has the remainder, the user must guarantee that this condition remains true after updates. To do so, for example, it may be necessary to move a tuple from one machine to another upon a raise or paycut.

The third mechanism is designed to allow the system to utilize (and enforce automatically) a distribution criteria. This requires a new form of the CREATE command.

```
DIST_CREATE RELATION_NAME({DOMAIN_NAME = FORMAT})

  ({QUALIFICATION = LOCATION})
```

Here, QUALIFICATION is a valid QUEL qualification involving only a single tuple variable ranging over RELATION_NAME, and LOCATION is a valid location.

An example of this command is the following:

```
RANGE OF E IS EMPLOYEE
DIST_CREATE EMPLOYEE (NAME = C10, AGE = I2, SALARY = I4)
  (E.SALARY <10000 = LOC1,
   E.SALARY >=10000 = LOC2)
```

The effect of this command would be to create a distributed relation. Moreover, the system would ensure on all updates that the distribution criteria remained in force. In addition, it would try to utilize the distribution criteria to limit any search for

information. The former is easily done by checking each modified or added tuple against the distribution criteria and inserting it in the local data base only if it meets the local distribution criteria. Otherwise a message must be sent to the appropriate machine. To achieve the latter effect requires the system to examine the qualification presented by the user and direct his interaction only to those machines whose distribution criteria do not have an empty intersection with the one of the user. Although this step requires a theorem prover in the general case, there are several simple cases that can be checked readily.

The reasoning behind suggesting that the distribution criteria only have a single tuple variable is that the above two steps appear reasonable for this special case and are much more difficult otherwise. Also, it is felt that a more general criteria is not needed.

3) To achieve backup copies, a backup command is required, for example:

```
BACKUP OF RELATION_NAME(LOCATION = X) IS
RELATION_NAME(LOCATION = Y)
```

4) In section 6 it will be seen that there is a high overhead in communication traffic assoicated with CREATE and DESTROY since network consistency must be guaranteed. To help avoid this overhead two kinds of relations will be permitted:

a) regular relations. These are shared among the network.

b) local relations. These are only visible to a user logged onto the machine where they reside.

A naming convention will be enforced to distinguish the two cases. Of course, users must be aware of the convention. Local relations involve substantially less overhead to manipulate as will be presently indicated.

IV SYSTEM CATALOGS

Each machine should keep system catalogs for the relations which reside physically on that machine. Since this condition must be true for a network consisting of a single machine, it appears useful to require this situation to be generally true. For purposes of this paper the INGRES system catalogs contain four types of information.

1) the relation name
2) parsing information (domain names, format, etc.)
3) performance information (number of tuples, storage structure, etc.)
4) consistency information (protection, integrity constraints, etc.)

A distributed INGRES can exist with each machine maintaining only its own system catalogs (subject to some consistency constraints discussed in Section 6). However, an interaction which involves a relation not on the originating machine may necessitate a search of the complete network for appropriate catalog information before execution can take place. To potentially improve

performance, at least the following options for redundancy exist.

1) Designate one machine "GOD". Store a complete collection of system cataglogs at that site. In this case unknown catalog information can always be obtained by requesting it from "GOD".

2) Store item 1) above for each machine at every other machine.

3) Store 1) and 2) on every machine.

4) Store 1) - 4) as above.

5) Whenever items 1) - 4) are required by a machine, save them but make no effort to correctly update them. Discard such information after a predetermined length of time.

The advantage of case 1 is that there would be no need to broadcast commands to each machine, one could simply ask "GOD" where desired data is located. Of course, the network fails completely if "GOD" becomes inoperative. Improved reliability can only be obtained by having a backup copy of "GOD". The performance tradeoff concerning "GOD" is the communication traffic generated by broadcasting requests when necessary versus the traffic generated keeping "GOD" with an up to date collection of system catalogs.

Very crudely, let k be the number of sites and x the percentage of INGRES commands which involve data at a remote site. Without "GOD" there will be an average kx messages per interaction. With "GOD" per interaction there will be x requests to "GOD" for locations of data, x requests to these locations to satisfy the interaction and then y messages to update the catalogs at "GOD's"

location (where y is the percentage of commands which update the system catalogs). Because INGRES currently maintains detailed performance information y is not substantially less than 100%. (However, y could be reduced substantially by less frequent maintenance of this information.)

Hence, the "GOD" solution has lower traffic if

$$x(k-1) > 2x + y$$

If $k = 8$, $x = 0.1$ and $y = 0.5$, then both methods generate the same traffic.

Case 2 assures that any machine knows where all relations in the network are. Hence, a request for catalog information can be sent to the correct machine and a broadcast of the request avoided. The overhead of case 2) is that of keeping the information current. That may or may not be greater than the overhead of supplying "GOD" with much more detailed information.

Case 3 will allow an interaction to be completely parsed at the site from which it originates with no requests for additional information. However, for execution to begin items c) and d) must be requested from remote sites.

Case 4 generates no network traffic in parsing and deciding on an execution strategy for an interaction. However, the cost of keeping such detailed information current may be very high except in "RETRIEVE almost always" environments.

Case 5 appeals to a working set management strategy for system

catalogs. This technique is widely used in memory management for operating systems [SHAW75]. Here, a machine might be required to assemble catalog information about a relation residing at a remote site. However, once assembled, the information would be entered into local system catalogs. Then, if a particular relation is referenced again, catalog information is at the local site and need not be requested again. After a certain period of time catalog information so obtained would be declared "out of date" and discarded. In this case one may obtain execution time errors and optimization mistakes because of inaccurate information.

V DECOMPOSITION

Since INGRES decomposes a multivariable interaction into a sequence of one variable interactions [WONG76, STON76], additional steps must be taken when the relations involved are not on a single machine. Also, the optimization problem discussed in [WONG76] must be restated.

If view 1) is used there is no problem. The interaction need only be sent to the correct machine for processing according to the currently implemented algorithms.

In view 2) or 3), one always has the option of reducing the problem to the case of view 1) by a sequence of MOVE commands to assemble all needed relations on a single machine. The other option is to extend the decomposition algorithm.

The machine on which an interaction originates can parse the interaction and start the decomposition process. It can perform

the "one-variable overlap algorithm" to split the interaction into components. Any of these components which involve only a single tuple variable can be immediately sent to the appropriate machine(s) and processed. The result is a temporary relation formed on the appropriate machine(s) which must return a descriptor (summary of catalog information) to the originating machine.

The originating machine must now decide on a tuple variable on which to perform tuple substitution. The required performance information concerning each of the relations involved already exists on the machine either because it was assembled before parsing or returned as a result of a one variable clause.

After this decision, the entire interaction must be sent to the machine(s) on which reside the relation to be substituted for. A descriptor for all other relations involved must also be sent. These machines then perform tuple substitution, forming in the process, a sequence of interactions each with one less tuple variable. Each such interaction is treated as if it were an incoming interaction. The process terminates when only a single tuple variable remains as noted in [WONG76].

Two notes will be made, then we will do an example.

1) If the command is a RETRIEVE INTO NEW_RELATION, the resulting location of NEW_RELATION is the machine which has the only relation for which tuple substitution is not performed. Moreover, if that relation is distributed, NEW_RELATION will be distributed also. This location(s) are not predetermined, since the processing strategy for interactions makes incremental decisions. If

the user wishes NEW_RELATION elsewhere, it must subsequently be moved with a MOVE command. Alternately, the user can specify:

RETRIEVE INTO NEW_RELATION (LOCATION = X)

which will guarantee a specific location. In this case the decomposition strategy can be invoked and a MOVE command generated subsequently. Alternately, the algorithm in [WONG76] can be modified to avoid substituting for a relation on the appropriate machine (if that is possible). This will ensure that the relation ends up on the correct machine.

2) The algorithm in [WONG76] does not consider a network environment. In order to do optimaization in this case the following parameters could potentially be used in the optimization equations of [WONG76].

a) speed of the secondary storage device on each machine

b) speed of each CPU

c) communication cost between each pair of machines

d) the location desired for the result relation (or the location of the relation to be updated)

We now indicate an example of distributed decomposition at work.

Suppose EMP (NAME, SALARY, MANAGER, AGE, DEPT) is a relation distributed on machines 1 and 2. Suppose on machine 1) the following interaction is received:

Find the employees under 35 who earn more than their managers, i.e.

```
RANGE OF E IS EMP
RANGE OF M IS EMP
RETRIEVE (E.NAME)  WHERE E.SALARY> M.SALARY
      AND E.MANAGER = M.NAME
      AND E.AGE < 35
```

The following steps are performed:

1) Machine 1) assembles (if it does not have already) a descriptor for the two pieces of the EMP relation and parses the interaction.

2) Machine 1) detaches the one variable clause (E.AGE < 35) and issues the following interaction for processing on both machines 1) and 2).

```
RANGE OF E IS EMP
RETRIEVE INTO W(E.NAME, E. SALARY, E.MANAGER)
   WHERE E.AGE< 35
```

As a result W will be created on both machines and a descriptor returned to machine 1) containing catalog information on W.

3) The query which remains is

```
RANGE OF E IS W
RANGE OF M IS EMP
RETRIEVE (E.NAME)  WHERE E.SALARY > M.SALARY
```

```
        AND E.MANAGER = M.NAME
```

Machine 1) now decides whether to substitute for E or M. Since, W is smaller than EMP, E is likely to be a good choice. Hence it transmits the above interaction two both machines 1 and 2 with a descriptor for W and EMP and an indication that W is to be substituted for.

4) Each machine now substitues for W the first or next tuple in W producing the query:

```
    RANGE OF M IS EMP
    RETRIEVE (name)  WHERE salary > M.SALARY
         AND name = M.NAME
```

This is a one variable query which must be processed by both machines.

Hence, for each substituted tuple on either machine two queries are generated, one for each machine.

5) After finishinging tuple substitution both machines report back completion information to the calling process who reports back to the user.

In this case qualifying tuples are sent directly to the users terminal as they are discovered. However, if the query had called for the construction of this information in a new relation, it would have been distributed on both of the machines.

Note that a non local query is generated for each tuple which exists in W, a sizeable traffic load on the network.

It should also be noted that a new invocation of the INGRES processes must be invoked at each level of substitution. Hence, a tree of processes is used to perform an interaction. This is required or else deadlock could easily result. Note however, that the overhead of these invocations is NOT trivial. As a result, it may be more efficient to assemble W and EMP on a single machine which would then generate no traffic during processing at all. This option becomes more attractive as the interactions become more complex.

The situation is more complex for updates. Consider the following example whereby the salary of all employees who earn more than their managers is to be changed to equal that of their managers.

```
RANGE OF E IS EMP
RANGE OF M IS EMP
REPLACE E(SALARY = M.SALARY)
  WHERE E.MANAGER = M.NAME AND E.SALARY > M.SALARY
```

As noted in [STON76] this command is turned into the following RETRIEVE to isolate tuples to be charged

```
RANGE OF E IS EMP
RANGE OF M IS EMP
RETRIEVE (E.TID, M.SALARY) WHERE E.MANAGER = M.NAME
  AND E.SALARY > M.SALARY
```

The result of this command is a "deferred update file" which is then processed to alter the salary for the tuple with a tuple identifier of (TID) to the value of M.SALARY.

Several problems arise:

1) The deferred update file will, in general, be distributed. Moreover, the decomposition process can be carried out for this example to show that the deferred update file for each machine may end up with updates to be done by both machines.

Clearly TID must be expanded to be a unique network identifier by inclusion of a machine location.

2) A synchronization problem exists. On a single machine the deferred update file can only be processed after the RETRIEVE which created it has been completed. Otherwise, an inconsistent data base could be generated as noted in [STON76]. Since a deferred update file will be processed by perhaps more than one machine in a network, all RETRIEVE's must be finished before ANY deferred update file can be processed.

The approach needed appears to be the following: Each deferred update file must be processed redirecting updates which are for different machines. This may be done as the file is created broadcasting one tuple at a time or more economically by processing the deferred update file as whole redirecting all changes to each machine as a unit. Upon acknowledgement from all machines of receipt, it can issue "DONE" to the process controlling the interaction on the originating machine. When the originating machine receives "DONE" from all machines involved in the interaction, it can then issue a message "do deferred update". Upon a second "done" from each machine, it can send "done" to the user and the interaction is finished.

VI CONCURRENCY CONTROL

The concurrency control algorithm given in [STON74, STON76] will no longer work without extension to a network. There appear to be two choices:

1) Avoid deadlock by requesting all needed resources in advance. For a single INGRES command, these are known and safety [STON74] can be guaranteed by requesting an appropriate collection of locks. If they cannot all be obtained from the appropriate lock subsystem on the various machines, all locks held must be released and a retry after some delay must be done. Note that the overhead of obtaining such locks is equal to that of a non distributed INGRES system, in the special case where all relations reside on the originating machine. Also, note that implementing this scheme is a straightforward generalization to the current INGRES concurrency scheme.

2) Deteck deadlock if it occurs and back out a chosen "victim". This scheme is used in SYSTEM-R [GRAY75, ASTR76] for a single machine. Detecting deadlock requires assembling the equivalent of a "lock out graph" [MACR76] for the whole network and then detecting cycles in it. Whatever machine performs this function effectively must assemble ALL concurrency information for the network and maintain its currency. It is more reasonable to simply route all lock requests to this machine and have it resolve conflicts. Hence, this machine effectively becomes "GOD".

Without such a "GOD" deadlock detection appears difficult and

costly.

Picking a victim and performing backout involves applying the recovery scheme to the INGRES process which originated the transaction and all its slaves. This involves little additional complexity to the current recovery algorithm.

Note that the backout scheme just mentioned is required if one of the machines which is involved in processing a transaction crashes.

In addition to the concurrency problem, there are several consistency problems.

1) INGRES must guarantee that a data base D has a unique data base administrator (i.e. D cannot exist at two sites with different DBA's). Moreover, it must resolve the concurrency problem which arises if two DBA's try to simultaneously create a data base with the same name.

To solve this problem, CREATDB (CREATE DATA BASE) must time stamp [LAMP76] its intention of creating a data base and send such a time-stamped message to each other site. Then, it must wait for an acknowledgement from all sites before actually executing the command. If it receives a message from another site with the same intention with an earlier time-stamp, it must issue an error message to its user. Identical mesages with identical time stamps are "ties" and can be broken by an arbitrary ordering of the machines in the network. The time stamping mechanism is assumed to follow the rules in [LAMP76].

Again, the extension to the curent code is not difficult.

2) In views 1) and 2) INGRES must guarantee that a relation exists at only one site; in view 3) it must guarantee that all relations of the same name have identical column names.

The same solution as above can be used. Note that each time a CREATE is executed, all other machines must be informed. This overhead must be tolerated for relations which are "non local" as discussed in Section 3.

3) Crash recovery software must restore a consistent network.
On a single machine INGRES provides the facility that each INGRES interaction is run to completion or its effects are undone (i.e. the data base appears as if the command has never been run) the recovery software is especially tedious when updates to the system catalogs are involved (such as for CREATE which creates a relation).

In a network recovery becomes much harder. At any point in processing an interaction one of the involved machines may fail. We assume that such a failure is always signaled by failure to acknowlege a message or failure to respond "DONE". Within a system alloted time any INGRES command is processed by a tree structure of operations on various machines. If any operation fails the entire tree must be backed out. This can be done by percolating a "back-out" back to the root of the tree (i.e. the process communicating with the user) who can then percolate "back out" to all processes it calls. Recursively, every node will be notified to reset. A similar process holds for nodes below the crashed

node.

Other problems exist if redundant system catalog information is kept. When the down machine is restored, this information must be updated to be consistent with the current state of the network. It appears feasible to do this by using the log on any other machine. Special problems exist with back-up copies which are discussed next.

4) Back-up copies must be kept consistent.

If "GOD" exists he can know where any redundant copies exist and direct updates to the prime copy and any back up copies. However, if "GOD" does not exist, then someone else must know where copies are. The only reasonable choice is to have such information both on the machine where the prime copy resides (so updates can be redirected to other copies) and on each of the backup machines so one can take over if the prime machine fails.

The problem arises concerning how to switch to a backup copy if the prime machine fails, how to bring the prime machine "up to date" after it resumes service and how to switch back to it as the prime machine.

Suppose the backup machines are ordered and each periodically sends "are you alive" to the prime machine. With no acknowledgement, the first backup machine can become the prime machine. During the switch and the interval between the crash and the "are you there" some updates may be lost. However, each backup machine can obtain a relation consistent as of some point in the

past be running the recovery software (i.e. each INGRES interactions will be processed to completion or not run at all).

When the down machine resumes, the recovery software will restore a data base consistent with that of the backup machine at the time it took control. It must then request a copy of all updates which occured while it was down. Finally it must send an "I am alive" to all backup machines. Upon acknowledgement by the first backup machine, it can resume normal operation.

In this manner, one copy is designated the prime copy and it must be guaranteed to have current and consistent data. Hence, any update must be directed first to this copy. Subsequently, it can redirect updates to other copies. If this is done, only one copy of the relation is guaranteed to be current; hence, user retrieval requests can only be directed to this copy. Other copies serve only to augment system reliability.

The other option concerning updates of copies is to direct an update to any of the copies and then have it redirect the update to all other copies. In this case no copy is designated the "current" copy. As a result, an update can only be committed when a positive acknowledgement of update completion is received from all other copies. As a result, each copy is synchronized on each update. This entails substantially greater overhead than the previous scheme. However, it allows a retrieval request to be directed to the "closest" copy of a relation. This second option would be desirable for augmented performance in "retrieve mostly" situations.

VII INGRES IMPLEMENTATION

INGRES will be extended to present view 3) mentioned in Section 2). There are several reasons for this choice.

1) There appears only a marginal increase in complexity over implementing view 2). Decomposition control and concurrency are the same complexity in either case. The optimization of decomposition is harder but not impossibly so. Hence, there appears to be little cost in selecting view 3) over view 2). On the other hand, view 1) appears to emasculate the network and is considered unacceptable.

2) The system catalogs obey view 3) regardless of whether data relations do. It appears useful not to distinguish the two types of relations (i.e. it is useful to be able to query the system catalogs using view 3)).

3) View 3) appears useful for many data relations.

Moreover, INGRES will not have a "GOD" even though concurrency problems are eased by his presence. There are several reasons for this choice:

1) "GOD" may be a performance bottleneck since all traffic must be routed through him.

2) The network will have good performance only if network traffic is moderate. If most requests are non-local, there will be huge traffic and a better solution might be to create a large central-

ized system. Suppose then that ALMOST ALL REQUESTS are local.

Hence with a "GOD" each local interaction must involve communication with him concerning concurrency control and updates to system catalogs. Moreover, a backup copy of "GOD's" system catalogs and concurrency information must exist on some other machine in case of a crash. Hence, all such communication will go to at least two machines. Without "GOD", local interactions generate no traffic. Hence, the "no GOD" solution should generate lower traffic.

The system catalogs will be treated according to strategy 5) discussed previously. This "working set" philosophy should cut network traffic if "locality of reference" exists.

Moreover, strategy 2) will also be used. This is required for reliability considerations. If a machine crashes and the network continues with one less machine, then interactions can be processed which do not impact the down machine or which involve a relation on the down machine which has a copy elsewhere. Unless each machine knows what relations are on the down machine, it cannot know whether or not its interaction can be processed. Hence, unless strategy 2) is used in view 3), the crash of a machine will effectively crash the whole network.

Copies will be allowed for reliability. Updating copies will be accomplished by updating the prime copy first and then redirecting updates to any copies.

Safety for concurrent transactions will be guaranteed a-priori

for a transaction consisting of a single INGRES command. The decision to support this unit for a transaction is discussed in [STON76]. Safety will be accomplished by requesting from each lock subsystem on the appropriate machines all needed resources. If all locks can be granted, the transaction can proceed; otherwise, it must release all its locks and try again at a later time.

View 3) will be supported and distributed relations allowed in two categories. Either the user can decide and enforce the distribution criteria and inform the system of any knowledge he has concerning the location of desired information. Alternately, the user can declare a relation using DIST_CREATE and the system will enforce the distribution criteria.

REFERENCES

ASTR76 Astrahan, M. et. al "System-R, A Relational Data Base System" ACM Transactions on Data Base Systems Vol 1 No 2 June 1976

CASE72 Casey, R., "Allocation of Copies of a File in an Information Network", Proc 1972 SJCC, AFIPS Press, 1972.

CHES75 Chesson,G., "A Network UNIX", Proc 1975 ACM SIGOPS Conference, Austin, Texas, November, 1975.

CHU 69 Chu, W., "Optimal File Allocation in a Multiple Computer System", IEEE Transactions on Computers, October, 1969.

GRAY75 Gray, J.N.,Lorie, R.A., and Putzolu, G.R. "Granularity of Locks in a Shared Data Base", Proc. 1975 VLDB Conference, Framingham, Mass., Sept., 1975.

HELD75 Held, G., Stonebraker, M., and Wong, E., "INGRES - A Relational Data Base System," Proc. 1975 National Computer Conference, Anaheim, Ca., June, 1975.

KARP76 Karp, P., "TELENET," Proc. Berkeley Workshop on Distributed Data Management and Computer Networks, Berkeley, Ca., May 1976.

LAMP76 Lamport, L., "Time Clocks and the Ordering of Events in a Distributed System," Massachusetts Computer Associates, May, 1976.

LEVI75 Levin, K and Morgan, H., "Optimizing Distributed Data Bases - A Framework for Research" Proc 1975 National Computer Conference, AFIPS Press, 1975.

MACR76 Macri, P., "Deadlock Detection and Resolution in a CODASYL Based Data Management System," Proc. 1976 ACM-SIGMOD Conference on Management of Data.

MARI75 Marill, T and Stern, D. "The Datacomputer- A Network Data Utility", Proc 1975 National Computer Conference, AFIPS Press, 1975.

RITC74 Ritchie, D.M. and Thompson, K. "The UNIX Time-Sharing System," CACM, Vol. 17, No. 3., March, 1974.

Robe70 Roberts, L. and Wessler, B., "Computer Network Development to Achieve Resource Sharing," Proc. SJCC, 1970, AFIPS Press.

SHAW75 Shaw, R., "The Logical Design of Operating Systems," Adison Wesley, 1975.

STON74 Stonebraker, M., "High Level Integrity Assurance in Relational Data Base Systems", University of California, Electronics Research Laboratory, Memo. ERL-M473, August, 1974.

STON76 Stonebraker, M. et al "The Design and Implementation of INGRES", ACM Transactions on Data Base Systems, Sept. 1976.

THOM76 Thompson, K. (private communication)

WHIT70 Whitney, V., "A Study of Optimal File Assignment and Communication Network Configuration", Ph.D dissertation, University of Michigan, 1970.

WONG76 Wong, E., and Youssefi, K., "Decomposition- A Strategy for Query Processing", ACM Transactions on Data Base Systems, Sept. 1976.

ZOOK76 Zook, W., Youseffi, K., Kreps, P., Held, G. and Ford, J., "INGRES- Reference Manual", University of California, Electronics Research Laboratory, Memo. No. ERL-M585, April, 1976.