PERFORMANCE ANALYSIS OF DISTRIBUTED DATA BASE SYSTEMS

by

Michael Stonebraker, John Woodfill, Jeff Ranstrom, Marguerite Murphy
Joseph Kalash, Michael Carey and Kenneth Arnold

Memorandum No UCB/ERL M82/85

10 October 1982

ELECTRONICS RESEARCH LABORATORY

Research Sponsored by the Air Force Office of Scientific Research under grant number 78-3596, by the Navy Electronics Systems Command under contract numberN00039-81-C-0569, and by the Army Research Office under grant number DAAG29-76-G-0245.

PERFORMANCE ANALYSIS OF DISTRIBUTED DATA BASE SYSTEMS

by

Michael Stonebraker, John Woodfill, Jeff Ranstrom, Marguerite Murphy
Joseph Kalash, Michael Carey and Kenneth Arnold

DEPARTMENT OF ELECTRICAL ENGINEERING AND COMPUTER SCIENCE
UNIVERSITY OF CALIFORNIA
BERKELEY, CA

ABSTRACT

In this paper we discuss the design of Distributed INGRES and the performance testing that is planned for it. We also give initial benchmark data for the system. In addition, we discuss analytic and simulation studies which are in progress and implementation difficulties we have faced.

1. INTRODUCTION

There have been a considerable number of algorithms developed to support distributed relational data bases in the areas of concurrency control, crash recovery, support of multiple copies of data, and command processing.

At present there is little concrete knowledge concerning the performance of such algorithms. Previous work has been based exclusively on simulation, *e.g.* [RIES79, GARC79, LIN81] or formal modeling, *e.g.* [GELE78, BERN79]. It is the basic objective of the Distributed INGRES project to provide empirical results concerning the performance of alternate algorithms.

In Section 2 we discuss the current state of Distributed INGRES. Then, in Section 3 we present initial benchmark data on the running system. Section 4 discusses the additional benchmarks that are planned while Section 5 and 6 describe some simulation and analytic studies that are underway. Lastly, Section 7 comments on the implementation difficulties that we have faced.

2. DISTRIBUTED INGRES

Distributed INGRES operates in a hardware environment consisting of a collection of DEC VAX 11/780s and 11/750s all running the UNIX operating system. In fact, all run 4.2BSD, a version of UNIX enhanced at Berkeley with paging and numerous program development tools. As of September 1982 there are 5 11/780s and 5 11/750s connected by a 3Mbit ETHERNET purchased from Xerox. The 4.2BSD software has been extended to support remote interprocess communication and remote execution of a process. Hence, one can spawn a process on a remote machine and then do interprocess communication with that process as if it were on the same machine.

Distributed INGRES has been described in [EPST78] and is operational with many of its features at this time. It consists of a master INGRES process which runs at the site where the command originated and slave INGRES processes at each site which have data involved in the command. The master process does parsing, view resolution and creates an action plan to solve the command using the fragment and replicate technique. The slave process is essentially one-machine INGRES [STON76] with minor extensions and the parser removed. The coordinator and slaves communicate over the 4.2BSD interprocess message system.

Distributed INGRES supports fragments of relations at different sites. For example, one can distribute a relation EMP as follows:

```
create EMP (name = c10, salary = i4, manager = c10,
                age = i4, dept = c10)

range of E is EMP
distribute E at Berkeley WHERE E.dept = "shoe"
               at Paris    WHERE E.dept = "toy"
               at Boston   WHERE E.dept != "toy" and
                                  E.dept != "shoe"
```

Berkeley, Paris and Boston are logical names of machines which are mapped to site addresses by a table lookup. The distribution criteria is assumed to partition the EMP relation and is not currently checked for this property by Distributed INGRES software. A one site relation is a special case of the above distribute command, *e.g.*

```
distribute ONE-SITE at Berkeley
```

At the current time all QUEL commands are processed correctly for distributed data with the exception of aggregates. For example, it is acceptable to perform the following update:

```
range of E is EMP
replace E(dept = "toy") where e.salary > 10000
```

This command will be processed at all three sites where fragments of the EMP relation exist. Moreover, all qualifying tuples must have an update performed and their site location may have to be changed.

A two phase commit protocol is implemented [GRAY78]. Hence, a "ready" message is sent from the slaves to the master when they are prepared to commit the update. If there are tuples which change sites, they are included with the ready message. The master can then process the tuples from all sites and redistribute them. This redistribution is accomplished by piggybacking the tuples onto the commit message when it is sent out. Optionally, a three phase commit protocol can be used [SKEE82] for added reliability. In this case the above redistribution is handled in phase two.

When a command spans data at multiple sites, a rudimentary version of the "fragment and replicate" query processing strategy is implemented. We illustrate this module by example. Suppose a second relation

```
    DEPT (dname, floor, budget)
```

exists at two sites as follows:

```
    distribute D  at Berkeley  where D.budget > 5
                  at Paris     where D.budget <= 5
```

and suppose a user submits the following query at Boston:

```
    range of E is EMP
    range of D is DEPT
    retrieve (E.name) where E.dept = D.dname and D.floor = 1
```

First, the one variable clause "D.floor = 1" is detached from the query and run at Berkeley and Paris, *i.e.*

```
    range of D is DEPT
    retrieve into TEMP (D.dname) where D.floor = 1
```

The original query now becomes

```
    range of E is EMP
    range of D is TEMP
    retrieve (E.name) where E.dept = D.dname
```

Data movement must now take place to satisfy the query. One relation (say TEMP) is chosen to be replicated at each processing site. Hence, both Berkeley and Paris send their portion of the TEMP relation to each site which has a fragment of EMP. The needed transmissions are:

```
    TEMP(Paris) -> Boston
    TEMP(Paris) -> Berkeley
    TEMP(Berkeley) -> Paris
    TEMP(Berkeley) -> Boston
```

At this time all three sites have a complete copy of TEMP and a fragment of the EMP relation. The above query is performed at each site, yielding a portion of the answer. As a last step each site returns tuples to the master site which displays them to the user.

Since our ETHERNET has the hardware capability to support broadcast, it is possible to perform the above four transfers by broadcasting each fragment of TEMP to the other two sites. However, the 4.2BSD operating system software does not support multicast or broadcast transmissions. Consequently, the above transmissions must take place individually and our strategy of replication may perform poorly [EPST78]. The network on which we planned to run [ROWE79] supported broadcast, and the code has not been changed.

At the moment, the relation to be replicated is chosen arbitrarily, so TEMP and EMP are equally likely to be selected for movement. A more elegant strategy is being planned.

3. INITIAL EXPERIMENTAL OBSERVATIONS

In these experiments we use a data base of employees with fields as discussed in Section 2. Our data base consists of 30,000 employee tuples, each 38 bytes in width. Our benchmark consisted of 1000 random updates of the form:

```
    replace E (salary = K) WHERE E.name = L
```

For this benchmark we compare the performance of four different INGRES configurations:

a) Normal INGRES on a single site data base with a VAX 11/780 CPU
b) Distributed INGRES run on a data base that happens to reside at the site from

which the benchmark originates. This site has a VAX 11/780 CPU.
c) Distributed INGRES run on a data base spread evenly over three machines, one VAX 11/780 and two VAX 11/750s. In this way exactly 1/3 of the updates are performed at each of three sites. Moreover, the benchmark was submitted in three job streams one at each site so as to avoid forcing a single site to be "master" for every command. Consequently, the master running at each site will discover an update which is equally likely to be processed at any of the three sites. In this case we report statistics for each site individually as well as a summation.
d) A computation called 3*780. This row is obtained by multiplying the 11/780 numbers from c) by three. Since one-third of the total work is performed at each site, this is an estimate of the resources which would be consumed if the benchmark had been run on three VAX 11/780s.

Table 1 gives three measures for each system, elapsed time, CPU time spent in application code, and CPU time spent inside the operating system.

The conclusion to be drawn from Table 1 is that Distributed INGRES is about 20 percent slower than normal INGRES when run on a local data base. This time is largely the extra overhead which Distributed INGRES must spend examining the distribution criteria and ascertaining that each of the commands is a local one. This checking is performed at run time in the current implementation; however, a smarter implementation would perform most of it at compilation time. The second source of overhead cannot be diverted to compile time. Each tuple which is updated must be checked against the distribution criteria to ensure that it is not being updated in a manner that would physically change its location.

Second, note that 3*780 Distributed INGRES uses 20 percent more CPU time than Distributed INGRES run on a local data base and 47 percent more CPU time that Normal INGRES. This appears to be the overhead of communication with a non-local data base. However, it could not possibly run slower than the 13:34 time reported for three site Distributed INGRES. Hence, it cuts elapsed time by at least 50 percent compared to Distributed INGRES on a one-site data base and 40 percent compared to Normal INGRES.

	user time	system time	elapsed time
Normal INGRES	7:34	3:04	22:34
Distributed INGRES - local data base	9:06	3:53	26:57
Distributed INGRES - three sites			
11/780	3:43	1:30	12:43
11/750	5:28	2:16	13:34
11/750	5:48	2:13	13:22
total	14:59	5:59	-
3*780	11:09	4:30	-

Table 1

Benchmark 1 results in 522,880 bytes being transferred across the network in case c) and uses less than two percent of the available bandwidth. It appears that a large number of machines could be added to ETHERNET before there were any bandwidth limitations. Also, as long as the workload partitioned evenly, total CPU time should remain a constant and be divided among a larger and larger collection of machines, resulting in a throughput essentially linear in the number of machines.

4. FURTHER EXPERIMENTATION

We propose to run a variety of benchmarks in our environment. We propose to vary the number of sites from which transactions originate, how many sites have the data required for individual commands and how much data is required to be moved between sites. The basic objectives are the following:

a) Network limitations

We speculate that it will be impossible to saturate our 3Mbit network. The reason is that CPU overhead to manage the network and do local data base processing is likely to saturate all computers on a reasonable size network before this bandwidth is achieved. We propose to measure the maximum bandwidth which our benchmark consumes. The result of this test will give insight into whether network delay or bandwidth is ever a significant issue in our environment. Moreover, we propose to explore under what circumstances a distributed DBMS can use more than 50Kbits of bandwidth. This will test whether our software could saturate a long haul network such as the ARPANET. This test will shed light on whether semi-join tactics which minimize data transmissions are desirable in distributed environments.

b) Message Limitations

We speculate that the operating system cost of sending and receiving messages may be a significant factor in distributed data base performance, and propose to test this hypothesis by direct measurement. If so, a distributed DBMS should attempt to package large messages.

On the other hand, if the operating system cost for messages is not significant, then we will have discovered that the entire network subsystem is not a bottleneck in a distributed DBMS. This has great impact on the criteria to be optimized by the query processing algorithm.

c) CPU Saturation

We expect that many benchmarks will saturate all CPUs which are involved in command processing and this will be the fundamental limitation in a distributed DBMS. If so, a query processing algorithm should schedule the work over as many machines as possible.

d) Uneven Work Distribution

Our simulation of a similar environment [MCCO81] showed that an uneven workload distribution among the machines caused substantial performance degradation. We noted that statistical fluctuations in a uniformly distributed workload could easily cause the command processing loads at the various sites to become unbalanced. In this case response time for a distributed transaction became the response time of the processing site with the heaviest load. This site was slowest to respond and the transaction could not be completed until this site finished.

Also, we found that an uneven work distribution, once created, tended to persist for a long period of time. Hence, poor response time also tended to persist. A similar phenomenon has been observed in the locking subsystem of System R [BLAS79].

We plan to measure to what extent this uneven workload phenomenon surfaces in a benchmark of uniformly distributed work. If it is sizeable, then a query processing algorithm should make rebalancing the workload its optimization criteria.

5. CONCURRENCY CONTROL

The experiments sketched above should shed light on query processing and crash recovery algorithms. In addition, we expect to experiment with a variety of concurrency control schemes. Unfortunately, there are twenty or more schemes which have been proposed. Instead of attempting to implement all twenty in Distributed INGRES (which was never designed with schemes other than locking in mind), we are proceeding by a combination of theoretical analysis and simulation.

We have proposed an abstract model of concurrency control algorithms within which we can address the performance tradeoffs of various popular schemes. The model facilitates comparisons of the CPU overhead, storage overhead, concurrency characteristics, and message overhead of alternative schemes. A report on this analysis is nearing completion [CARE82].

In order to validate the conclusions of the model and to offer further insight we have also written a simulator of distributed concurrency control schemes. Experimentation with this simulator will commence shortly. We intend to validate the simulator by comparing its results for the Distributed INGRES locking scheme with actual experimental data.

6. DISTRIBUTED ARCHITECTURES

An important aspect of any distributed data base system is sizing considerations. I/O subsystems, CPUs and networks must be balanced to achieve maximum throughtput. Moreover, the topology of the network may be a consideration. We are constructing a second simulation model which can evaluate alternate distributed architectures. Using this model we hope to experiment with environments which are not easily tested in our VAX/ETHERNET environment.

7. IMPLEMENTATION PROBLEMS

In this section we mention a few of the difficulties that we have faced in the implementation.

1) Distributed Debugging

Attempting to remove the bugs from distributed programs has proved to be a frustrating and slow process. Programs which run on one machine pretending to be several do not usually run on several machines. Debugging tools for distributed environments are very primitive.

2) Machine Time

Attempting to obtain stand-alone time on a substantial collection of machines in order to perform experiments has been difficult. The social problem of obtaining cooperation from multiple independent system administrators has proved taxing.

3) Limitations on Operating System Parameters

Distributed INGRES requires a large number of open files and connections to numerous cooperating processes. Most machines on which we try to run are not configured with sufficient maximum numbers of these objects. Moreover, most system administrators refuse to reconfigure their systems to rectify the situation. As a result we must treat file descriptors and connections as a scarce resource and allocate them to tasks carefully.

4) Code Complexity

Distributed INGRES is about 1.7 times a large as normal INGRES. It has been substantially harder to design and code than any of us realized at the outset.

5) Connection Topology

A distributed data base system has a "master" and "slaves" as noted above. However, when data movement is required, a "receptor" must be activated at the receiving site. Additionally, when tuples change sites, they must be sent from a slave to the master who sorts them and redistributes them. The slave must be prepared to accept both commands and data from a master. Lastly, a user can interrupt the master which must reset all slaves and kill all receptors. Ensuring that each process is "listening" to the correct connection under all circumstances has been difficult. We have had considerable difficulty managing a complex connection topology.

6) Boredom

Distributed INGRES has been in development since 1979. Much of that time has been spent in "wait state" awaiting operating system support for networking. We have always been in the position of either "waiting a few months for the promised arrival of general facilities" or "spending a few months on an ad-hoc implementation of special facilities which would hopefully be thrown away". We have always chosen the former; and consequently, wait state has been frustrating. Moreover, a project which shows little noticeable progress results in boredom for the implementation team.

REFERENCES

[BERN79] Bernstein, P. and Chiu, D., "Using Semi-joins to Solve Relational Queries", Computer Corp. of America, Cambridge, Mass., Jan. 1979.

[BLAS79] Blasgen, M. et. al., "The Convoy Phenomena", Operating Systems Review, April, 1979.

[CARE82] Carey, M., "An Abstract Model of Data Base Concurrency Control Algorithms" (in preparation).

[EPST78] Epstein, R., et. al., "Distributed Query Processing in a Relational Data Base System," Proc. 1978 ACM-SIGMOD Conference on Management of Data, Austin, Texas, May, 1978.

[CARE82] Carey, M., "A Formal Model of Concurrency Control Systems" (in preparation).

[GARC79] Garcia-Molina, H., "Performance of Update Algorithms for Replicated Data in a Distributed Data Base," PhD Thesis, Stanford University, Computer Science Dept, June 1979.

[GELE78] Gelenbe, E. and Sevcik, K., "Analysis of Update Synchronization for Multiple Copy Data Bases," Proc. 3rd Berkeley Workshop on Distributed Data Bases and Computer Networks, San Francisco, Ca., February 1978.

[GRAY78] Gray, J., "Notes on Data Base Operating Systems," in Operating Systems: An Advanced Course, Springer-Verlag, 1978, pp393-481.

[LIN81] Lin, W., "Performance Evaluation of Two Concurrency Control Mechanisms in a Distributed Data Base System," Proc. 1981 ACM-SIGMOD Conference on Management of Data, Ann Arbor, Mich., May 1981.

[MCCO81] McCord, R., "Sizing and Data Distribution for a Distributed Data Base Machine," Proc. 1981 ACM-SIGMOD Conference on Management of Data, Ann Arbor, Mich., April 1981.

[RIES79] Ries, D., "The Effects of Concurrency Control on Data Base Management System Performance," Electronics Research Laboratory, Univ. of California, Memo ERL M79/20, April 1979.

[ROWE79] Rowe, L. and Birman, K., "Network Support for a Distributed Data Base System", Proceedings of the Fourth Berkeley Workshop on Distributed Data Management and Computer Networks, August, 1979, San Francisco, California.

[SELI80] Selinger, P. and Adiba, M., "Access Path Selection for a Distributed Relational DBMS," Proc. International Conference on Data Base management, Aberdeen, Scotland, July 1980.

[SKEE82] Skeen, D., "A Quorum-Based Commit Protocol," Proc. 6th Berkeley Workshop on Distributed Data Bases and Computer Networks, Pacific Grove, Ca., Feb 1982.

[STON76] Stonebraker, M. et. al., "The Design and Implementation of INGRES," TODS 2, 3, September 1976.