PROBLEMS IN SUPPORTING DATA BASE TRANSACTIONS IN AN OPERATING SYSTEM TRANSACTION MANAGER

by

Michael Stonebraker

Memorandum No. UCB/ERL M84/87

23 October 1984

ELECTRONICS RESEARCH LABORATORY

PROBLEMS IN SUPPORTING DATA BASE TRANSACTIONS IN AN OPERATING SYSTEM TRANSACTION MANAGER

by

Michael Stonebraker

Electronics Research Laboratory
University of California
Berkeley, Ca.

and

Deborah DuBourdieux and William Edwards

Prime Computers, Inc.
500 Old Connecticut Path
Framingham, Mass.

ABSTRACT

This paper reports on the experience of the authors in attempting to support data base transactions on top of an existing operating system transaction manager. It will be seen that significant modifications to both the example data base system and the example operating system are required to support the concept. The conclusion to be drawn is that operating system transaction managers will have to be designed more generally than is now suggested and that application programs (such as data base systems) will have to participate in the transaction management process.

1. INTRODUCTION

It is widely suggested that transaction management is a function that should be provided by next generation operating systems. Moreover, prototype systems have started to appear with transaction management facilities [MITC82, POPE81, SPEC83]. One of the few commercial implementations of this concept is in the PRIMEOS (c) operating system. The file manager of PRIMEOS is a software system called ROAM (Recovery Oriented Access Method) which provided recoverable synchronized update to files [DUBO82]. In Section II we briefly discuss the structure of ROAM.

The authors spent considerable time planning how to make an existing relational data base system operate on top of ROAM. The approach was to turn off the existing transaction routines in INGRES [RTI83] and instead use the facility provided by ROAM. In Section III we indicate the design problems which we encountered and the approaches we considered in dealing with these issues. Other issues conveying a similar theme are reported in [TRAI82]. The paper then concludes in Section IV with some lessons which we learned from this design experience.

2. ROAM

The Recovery Oriented Access Method is used by all Prime data management products which require transaction management services. Concurrency control is provided by a two-phase locking algorithm. To guarantee serializability locks are held to the end of a transaction and disk pages serve as the granule which is locked. Read-only transactions do not set locks and never interfere with update transactions. Their accesses are satisfied through the use of multiple versions in a manner similar to that reported in [CHAN81]. The same versions are used to accomplish transaction rollback.

When a ROAM user (who has not predeclared himself as read only) submits a read or write request, ROAM transparently acquires a read or write lock on the covering disk page(s). If the lock cannot be granted, ROAM will queue the requestor unless it determines that a deadlock exists. To provide recovery from disk-intact system crashes and user initiated aborts, ROAM supports transaction rollback by creating a preimage of each updated disk page in the system log. In addition, a postimage of the changed record is journaled to the log. This postimage is used to roll forward from a checkpoint in the event of a crash where data is lost. When a transaction commits, a commit record is written to the log and all pending writes are completed prior to returning control to the user.

3. DESIGN PROBLEMS

The following three subsections indicate the problems which we encountered in attempting to use the above facilities.

3.1. Page Writes

INGRES attempts to do efficient file access by reading and writing file data in fixed size blocks. The intent is to have this block size be a muliple of the operating system disk block size. In this way read and write requests will cross the minimum number of disk block boundaries. Moreover, all bytes on any operating system block read from the disk will be delivered to INGRES buffers. In most environments that INGRES runs, such block oriented I/O is much more efficient than record oriented I/O.

Consequently, when INGRES delivers an operating system block to ROAM in a write request, ROAM must record both the preimage and the postimage of the block in its log. Specifically, the ROAM log will contain postimages of all operating system blocks in which INGRES changes any record. Other applications in the PRIMEOS environment had previously tried postimaging whole blocks and found it drastically less efficient than postimaging only the record being changed. Hence, the first necessary change to INGRES would be to change the low level write routines to deliver record oriented updates rather than block oriented updates.

One consequence of this proposal is the possibility that a single INGRES update will require two log entries. In particular, INGRES performs a record insert by first finding space for the record on the desired page and writing it at that location. Then, INGRES finds a position in a "line table" at the bottom of the page and inserts the byte offset of the new record in this line table position. The record-id of the new record is the pair:

record-id = (page-number, line-number)

INGRES uses this record-id in any secondary indexes which the record participates in. The purpose of the line table is to allow physical reorganization of a disk block without changing record-ids. Hence, the second update which must be logged is the write to the line table.

It should be clearly noted that an operating system transaction manager must log both the data changes (such as the record insert above) and the structure changes (such as the insert to the line table). Thus, the OS transaction manager can recover physical images without being aware of their semantic content, and no computation is required during the recovery process. On the other hand, the current INGRES transaction manager logs only the data changes because it can reconstruct structure changes at recovery time as necessary. This logical recovery in INGRES minimizes the amount of data which must be logged at the expense of additional computation during recovery. Hence, moving transaction management into ROAM will tend to increase the amount of work performed at run time and decrease the amount performed at recovery time. The impact of such a change on system performance would need to be empirically studied.

3.2. System Catalog Problems

INGRES maintains a collection of system catalogs which contain meta-data about the data base. We focus the discussion in this paper on a single relation, the RELATION relation. This contains one tuple for each relation which exists in the data base with the following information:

relation name
relation owner
number of fields
storage structure (e.g. hash, heap, isam, etc.)
number of tuples
tuple width in bytes

For better or worse, INGRES currently maintains the number of tuples in a relation and uses this information to assist in generating optimized query execution plans. This number is updated as tuples are inserted into or deleted from a relation. Hence, consider the following transaction:

```
begin transaction
append to EMPLOYEE (name = "mike", salary = 1000, age = 19)
delete DEPT where dname = "toy"
end transaction
```

This transaction causes a write to the file containing the EMPLOYEE relation and a second write to the file containing the DEPT relation. However, it also causes two writes to the RELATION relation to update the tuple counts of the EMPLOYEE and DEPT relations. The ROAM lock manager will provide appropriate synchronization of this request by setting four write locks and holding them to end of transaction.

The net effect of this action is to lock the entire EMPLOYEE and DEPT relations from the time the tuple count is updated until the end of the transaction. This results from ROAM setting write locks in the system catalogs when the tuple counts are updated. All INGRES commands which impact the EMPLOYEE or DEPT relation must currently access the RELATION relation to validate that the proposed QUEL action is syntactically valid. All such reads to the RELATION relation will be blocked by the write locks held by the above mentioned tuple count update.

The conclusion is that deletes and inserts end up setting relation level locks inadvertently. In fact, since ROAM holds page level locks, all relations with a tuple on the same page in the system catalogs as the EMPLOYEE and DEPT relation are also locked. The performance consequences of such a locking policy are devastating.

To alleviate this problem, INGRES must either be modified so it does not update the tuple-count dynamically or INGRES must be able to tell ROAM to turn off concurrency control for the system catalogs. The first alternative is followed in System R [ASTR76]; however, it does not solve the problem. Consider a transaction which includes creating a relation as part of its work, e.g.:

```
begin transaction
create new-relation (field-1 = i2, ..., field-n = c20)
-other commands-
end transaction
```

The create command will set a write lock in the RELATION relation and hold it until the end of the transaction. All concurrent transactions must therefore wait if they access a relation on the same page in the system catalogs as the newly created one. Again, the loss of concurrency is considerable.

Consequently, the authors recommended the second alternative: modifying ROAM to provide a special access mode for INGRES system catalogs. In this mode locks are only held during actual updates of the catalogs and not until end of transaction. Unfortunately, there are recovery implications to this choice as discussed in the next subsection.

3.3. Relation Creation

Inside a transaction a user can ask that a relation be created or destroyed. Consider the following two example transactions:

```
T1: begin transaction
    destroy EMPLOYEE
    <other commands>
    abort transaction

T2: begin transaction
    create NEWREL
    end transaction
```

Suppose that T1 begins first and executes the destroy command. This will cause a tuple deletion from the RELATION relation. Now suppose that T2 is allowed to run and executes the create command. This will cause an insert to the RELATION relation. Suppose in the worst case that the insert is placed in the exact disk location vacated by the delete command. Since the system catalogs have only short-term locking as discussed in Section 3.2, this is a possible (although hopefully rare) event. Later when T1 aborts his transaction, ROAM will replace anything T1 wrote with the preimage which existed at the start of the transaction. This will cause the insert made by T2 to disappear. Obviously, this scenario must be avoided.

Unfortunately there are a whole collection of these kinds of events. In this paper we discuss one additional example. Suppose two transactions perform inserts to the same relation as follows:

```
T1: begin transaction
       append to EMPLOYEE ( ... ) where ....
       <other code>
       abort transaction

T2: begin transaction
       append to EMPLOYEE ( ... ) where ....
       end transaction
```

Consider the tuple count field in the RELATION relation. The transaction T1 will update the count when it finishes its insert. Then, T2 will further update the count and finish. Later, when T1 aborts and restores the previous value of the tuple count, he undoes the tuple count change made by T2. Over time, such inaccuracies in the tuple count field can become troublesome.

These are two examples of actions which cannot be blindly rolled backward to their previous value. The current INGRES transaction manager deals with the problem by "reversing" such actions rather than by physically rolling them backwards. For example, the reverse of incrementing the tuple count by a value X is to decrement it by X. Furthermore, the reverse of destroying a relation is recreating it. Consequently, the INGRES log is a collection of "events" some of which are undone by physically rolling backward to the previous data values and some of which are reversed by calling a routine specific to the event.

The design of ROAM as a general purpose operating system facility requires that its interface contain the minimum of information pertaining to any specific data management product. Its rollback facility requires no semantic knowledge of the data base being recovered, and for this reason the ability to reverse events was not supported. The special problems of handling system catalogs have caused us to revisit this architecture which we would now modify to allow two kinds of actions.

1) normal actions

These are updates for which roll backwards is the appropriate mechanism to use on transaction aborts.

2) user defined events (where "user" refers to the client product using ROAM)

An application program must be able to put a user defined event in the log. If the transaction is aborted, the ROAM transaction manager must agree to call a user written routine which will be able to reverse the event appearing in the log.

Notice that such user routines execute with all kernal privileges, and a malicious reversal program can do extreme damage. Thus, reversal programs must be treated just like any other kernal routines with respect to auditing their code and certifying it as trustworthy.

Besides a routine to reverse an event when a transaction aborts, the ROAM transaction manager must also agree to call a second user written procedure in another circumstance. After a "hard crash", i.e. one where disk data is permanently lost, a checkpoint of the data base must be restored and the log processed forward installing postimages of changes. Whenever a user defined event is encountered in the log, a user written "redo" routine must be called to redo the event. Hence, any user defined event must consist of a specific code to identify it in the log and two routines, a reverse routine and a redo routine.

INGRES would have to be changed to define about 6-8 such user events. The reverse and undo routines for these events comprise a substantial portion of the INGRES utilities (about 50-75K of code) which would have to be trusted kernal code.

4. CONCLUSIONS

It appears that a conventional operating system transaction manager which provides preimaging and postimaging of disk changes is not sufficient to support the needs of data base system transactions. There appear to be five solutions to this shortcoming:

1) Operating system transaction managers can be generalized to provide user events in their logs. Complexity, protection and efficiency issues arise in this solution. However, this approach provides the most complete solution and merits serious consideration.

2) Applications such as data base systems can be designed to have only events which can be physically rolled backwards. Whether such applications can be efficiently constructed is an open question.

3) Applications can rely on an operating system to roll back events and provide their own log and recovery procedures for events which must be reversed. This appears to require synchronization of multiple log managers and is probably the worst of the alternatives.

4) Applications can continue to write their own recovery and concurrency control facilities, as is commonly done today. Any operating system facilities would be ignored.

5) All applications can use data management facilities provided by the data base system. Hence, the DBMS is the only application which must provide recovery and concurrency control routines. In this scenario there is no duplication of effort. Unfortunately, telecommunications monitors want to have recoverable sessions with terminals and are automatically a non data base client with transaction needs. Hence, this option seems non viable.

It remains to be seen which of the remaining alternatives (1, 2 and 4) will turn out to be attractive.

REFERENCES

[ASTR76] Astrahan, M. et. al., "System R: A Relational Approach to Data," ACM Transactions on Database Systems, June 1976.

[CHAN81] Chan, A. et. al., "The Implementation of an Integrated Concurrency Control and Recovery Scheme," Proc. 1981 ACM-SIGMOD Conference on Management of Data, Orlando, Fla., June 1981

[DUBO82] DuBourdieux, D., "Implementation of Distributed Transactions," Proc. 6th Berkeley Workshop on Distributed Data Bases and Computer Networks, Asilomar, Ca., Feb. 1982.

[MITC82] Mitchell, J. and Dion, J., "A Comparison of Two Network-Based File Servers," CACM, April 1982.

[POPE81] Popek, G., et. al., "LOCUS: A Network Transparent, High Reliability Distributed System," Proc. Eighth Symposium on Operating System Principles, Pacific Grove, Ca., Dec. 1981.

[REDE81] Redell, D. et. al., "Pilot: An Operating System for a Personal Computer," CACM, February 1981.

[RTI83] Relational Technology, Inc., "INGRES Reference Manual," 1982

[SPEC83] Spector, A. and Schwartz, P., "Transactions: A Construct for Reliable Distributed Computing," Operating Systems Review, Vol 17, No 2, April 1983.

[TRAI82] Traiger, I., "Virtual Memory Management for Data Base Systems," Operating Systems Review, October 1982.