COORDINATING CONCURRENT ACCESS IN A DISTRIBUTED DATABASE ARCHITECTURE

M. J. Stucki, J. R. Cox, Jr., G.-C. Roman, P. N. Turcu
Department of Computer Science
Washington University
St. Louis, Missouri 63130

ABSTRACT

A distributed architecture for an interactive information system is described, and a scheme for *coordinating concurrent access to its data is* presented. The scheme is deadlock free and is carried out without the need for centralized control. Conflicts are detected as they occur, and competing processes are given exclusive access to the data they need.

INTRODUCTION

A research group at Washington University is working on the design of an interactive system that could provide the information handling facility for a large installation such as a medical center (COX 76). A major objective of this design effort is the creation of an architecture that can expand to accomodate increased workloads without incurring the cost and inconvenience of a major system reorganization. The architecture being considered, Figure 1, is a collection of computing systems that cooperate to perform the function of the information system. Each Q unit operates interactively with a subset of the user terminals, and each D unit operates a data management system for the mass storage devices attached to it. The Q units evaluate data requests entered at the terminals and obtain (or update) data by sending messages to appropriate D units. This architecture meets the expandability objective by being totally modular. Q modules are added or deleted as the population of terminals changes, while D modules are added or deleted as dictated by storage requirements and as needed to adjust the effective bandwidth of the storage system.

Implementing the functional aspects of an information system in such an architecture could be more difficult than in traditional architectures, and this paper looks at the problem of coordinating concurrent access to shared data. In order to preserve database consistency, some transactions must be able to obtain exclusive access to data (ESWA 76). This means that each module must have an allocation mechanism which is deadlock-free with respect to its own resources and which precludes intermodule deadlocks of the following sort. Suppose that Q_1 is executing transaction T_1, that Q_2 is executing transaction T_2, and that both transactions need exclusive access to data items x and y before they can proceed any further. Data item x is accessed through unit D_1 and data item y is accessed through D_2. Both transactions send messages to D_1 and D_2 requesting the needed data. D_1 allocates x to T_1 and puts the T_2 request in a waiting queue. D_2 allocates y to T_2 and puts the T_1 request in a waiting queue. Both transactions are now waiting for data they cannot get. They are deadlocked, and none of the units involved knows it. Such situations can also occur in traditional architectures and schemes for eliminating them have been reported in the literature. These schemes generally depend on a central coordinating or allocation mechanism and consequently are not applicable here. One exception is a scheme mentioned briefly in (CHAM 74).

This work was supported by the National Institutes of Health Research Grant No. RR-00396.

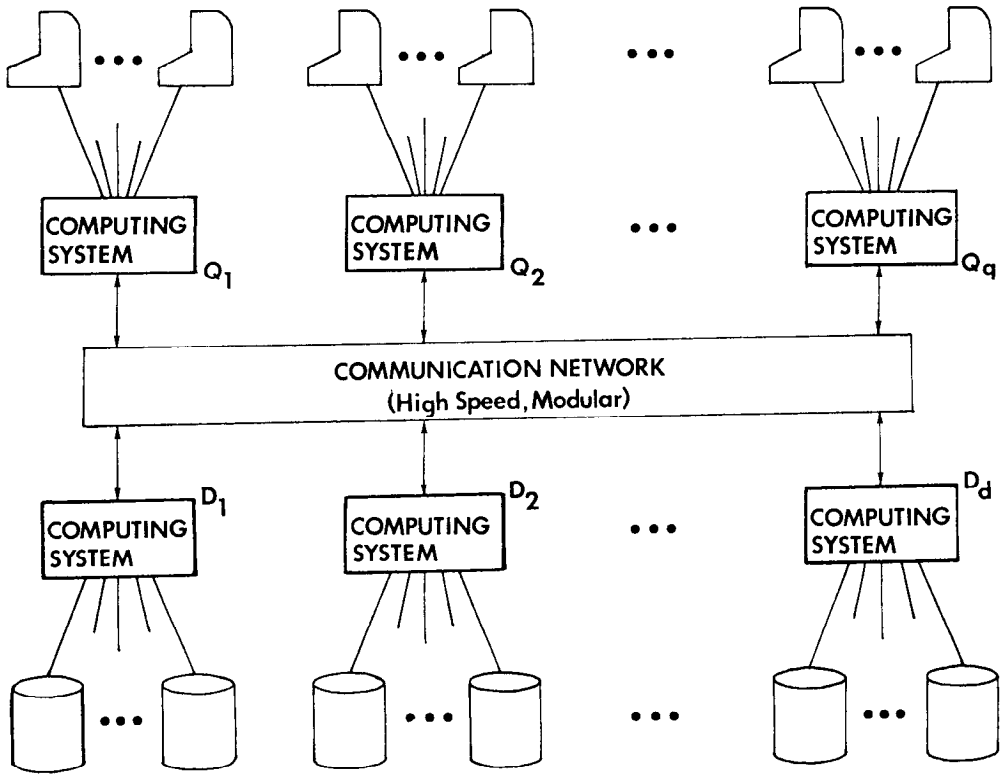

Figure 1. System Architecture

Our version of this scheme is described in the following section. It grants every transaction exclusive access to the data it needs.

THE LOCKING SCHEME

In this scheme, we assume that a transaction has three execution phases. During the first phase, it acquires exclusive locks for all the data that it needs. It works on the data during the second phase, and during the third phase, it releases all locks that it holds. In order to keep deadlock from occurring during the locking phase, we assign each transaction a service number at the time that it enters the system. Each service number is larger than the one last assigned, and the numbers may therefore be used to determine the relative "ages" of transactions. For example, if the service numbers of two transactions are compared, the transaction with the smaller number received its number first and is therefore "older" than the other transaction. In situations where a transaction desires to lock an element that is already locked, we automatically pre-empt the element if the lock belongs to a younger transaction that is still in its locking phase. Otherwise, the transaction is "blocked" by the transaction holding the lock and must wait for the element to become available. To see that this pre-emption policy prevents deadlock, we note that for deadlock to exist, there would have to be a set of transactions wherein each member is blocked by another member. If such a set could exist, it would have a unique oldest member, and the member blocking it would be younger. Since the blocking member is itself blocked, it must still be in its locking phase. The oldest member can therefore pre-empt the element it needs and is not actually blocked, a contradiction. This means that the set cannot exist and neither can deadlock.

In adapting this scheme to our architecture, the first step is to observe which units carry out the various tasks involved in the execution of a transaction. A transaction originates in a Q module. That module determines which D modules manage the data to be worked on and sends them messages specifying the data to be locked. Each D module performs its task and returns a "locking-completed" message. When all requests have been acknowledged, the Q module activates phase 2. The nature of this phase depends on how the data are to be worked on, and consequently we cannot say much about it. However, it probably requires more messages between the Q and D modules since messages are used for all intermodule communication, whether data or directives. When phase 2 is finished, the Q module sends a message to the D units directing them to release the locks held for this transaction, and the transaction then terminates.

Since locking is a D module function, a D module also takes care of the pre-empting of locks. The mechanism for this is reasonably straightforward. If a transaction R requests a lock that is already held by transaction H and the lock is pre-emptable, then H is put on a waiting list and the lock is given to R. If it is not pre-emptable, then R is put on the waiting list. When a lock is voluntarily released because the holding transaction no longer needs it, the lock is given to the oldest transaction on the waiting list.

In order to facilitate the making of pre-emption decisions, each D module keeps a status table in which it records its knowledge about the phase of a transaction. This knowledge is represented by a "state", and Figure 2 shows the states and state transitions that are possible. The arrival of a lock-request message causes the name of the requesting transaction and the state LOCKING to be entered in the status table. When the requested locks have been acquired, the state is changed to UNKNOWN and a locking-completed message is sent. The new state reflects the fact that the transaction could still be acquiring locks in other D modules, or, it could now have all the locks it needs and so be able to access data. When the D module receives a message requesting data access, the phase of the transaction becomes known and the state is changed to WORKING. Finally, when a lock-release message is received and processed, the transaction is deleted from the table.

A pre-emption decision is made in the following manner. The ages of transaction R and transaction H are compared. If H is older, the contested lock is not pre-emptable. If H is younger, the status table is consulted to determine its phase. The lock is pre-emptable if the state shown for H is LOCKING, it is not pre-emptable if the state is WORKING. If the state is UNKNOWN, the inquiry changes the state to CHECKING and then causes a message to be sent to the Q module that is the origin of H. The message reads:

> If transaction H is still in its locking phase, disregard the locking-completed message sent by this module and return the message "LOCKING". Otherwise, return the message "WORKING".

When a reply is received, the state is changed from CHECKING to that specified in the message and the decision is then made. The purpose of the CHECKING state is to indicate that a message has been sent and thereby keep other inquiries from sending the same message. Thus, if an inquiry sees the state CHECKING, it waits until the state changes and then makes its decision.

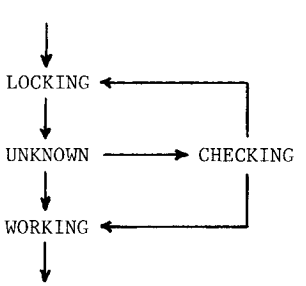

Figure 2. State Transition Diagram

This completes the description of the locking scheme except for one final consideration, the source of the service numbers. An obvious solution would be to have a unique process somewhere in the system that issues numbers on request. This, however, has several serious shortcomings: it would increase traffic in the communication network; it would become a performance bottleneck as system workload increased; and it would be a weak link whose failure could bring down the entire system. We avoid such problems by having a Q module generate service numbers for the transactions that it originates. Each Q module has a unique identifying number (see Figure 1) and this is used as the value of the low order portion of the service number. The high order portion is a time stamp that is read from a clock internal to the Q module. Service numbers are unique because the high order portion distinguishes those that originate in the same module and the low order portion distinguishes those that originate in different modules. The use of the time stamp assures that concurrent transactions are about the same age and that later transactions are younger than earlier ones.

The deadlock prevention aspect of the locking scheme does not depend on the clocks in the Q modules being synchronized. However, the clocks should be in reasonably close agreement in order to keep the competition for locks fair. Service numbers derived from a clock that is "fast" will be younger than concurrent transactions derived from a clock that is "slow" and this will bias the allocation of locks in favor of the slow clock. Fortunately, this effect is not significant unless the difference in clocks is large, small differences are swamped by other system delays. Finally, it is worth noting that, given the proper authorization, a Q module can give a transaction a high priority by assigning it an old, currently unused, service number.

REMARKS

Although we have yet to implement the locking scheme, we can estimate the overhead associated with the pre-emption aspect. This is an important consideration because we can expect pre-emption to occur much oftener than is actually necessary for preventing deadlock. If we consider the process of competing for a single lock, we note that there is very little computation involved. Furthermore, in every case but one, the information that is used and manipulated is in primary memory and thus quickly accessible. The sole exception is the case in which a message must be sent to a Q module. This case is of major concern because of the time it takes and because it increases the traffic in the communication network. Our estimate of pre-emption overhead is therefore given in terms of the expected number of such messages per transaction. To get this estimate, we consider the transactions serviced by a single D module and calculate an upper bound for the average number of messages that inquire about the phase of a transaction. We then multiply this by the average number of D modules that are consulted by a transaction.

Figure 3 shows the state-transition diagram for a transaction. The arc labels denote the probability of making the transitions. Some of the probabilities change with time and the subscripts are used to distinguish the values associated with different occurrences of a transition. The arc from UNKNOWN to CHECKING is particularly important because each occurrence of this transition causes a message to be sent. Consequently, the average number of messages sent per transaction is given by the average number of times the state CHECKING is reached. Let m_i denote the probability of occurrence of exactly i messages, let $\bar{n}$ denote the average number of messages, and let $p_i = s_i r_{i+1}$. Then

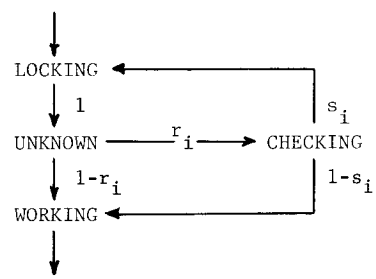

Figure 3. Transition Probabilities

$$m_0 = (1-r_1)$$

$$m_1 = r_1(1-p_1)$$

$$m_2 = r_1p_1(1-p_2)$$

$$m_3 = r_1p_1p_2(1-p_3)$$

$$\vdots$$

$$m_n = r_1p_1p_2p_3\cdots p_{n-1}(1-p_n)$$

$$\bar{n} = r_1(1+p_1+p_1p_2+p_1p_2p_3+\ldots)$$

The value of r_i depends on the probability that an older transaction wants one of the locks held by this transaction. Since the number of transactions that are older decreases with time, and since each pass around the CHECKING-LOCKING-UNKNOWN loop takes time, we have

$$r_{i+1} < r_i$$

The value of r_i actually becomes 0 before i gets very large. The value of s_i is probably independent of i which means that

$$p_{i+1} < p_i$$

Consequently, an upper bound for $\bar{n}$ is given by

$$\bar{n} < r_1(1+p_1+p_1^2+p_1^3+\ldots)$$

$$\bar{n} < r_1/(1-p_1)$$

The expected values of r_1 and p_1 will vary from installation to installation but we expect that they are substantially less than 1. This means that $\bar{n}$ is also substantially less than 1.

The average number of D modules that a transaction consults, denoted $\bar{d}$, is not known. It depends on many things, including the number of D modules in the system. Since $\bar{n}$ is small, we expect that

$$\bar{n}\,\bar{d} < 1$$

It thus appears that the pre-emption overhead of the locking scheme is not a cause for concern.

SUMMARY

A distributed architecture for large interactive information systems has been described. It is modular and can be expanded to compensate for increases in system workload. It has been shown that the concurrent access of shared data can be properly coordinated in such an architecture. The scheme presented for this is deadlock-free and does not depend on a centralized resource allocator.

REFERENCES

CHAM 74 Donald D. Chamberlin, Raymond F. Boyce, Irving L. Traiger. A Deadlock-Free Scheme for Resource Locking in a Database Environment. I.F.I.P. Proceedings 1974.

COX 76 J. R. Cox, Jr., "A Crosspoint Technology for High-Performance Information Systems," Information Systems Group Working Paper No. 1, Washington University, January 1976.

ESWA 76 K. P. Eswaran, J. N. Gray, R. A. Lorie, I. L. Traiger. "The Notions of Consistency and Predicate Locks in a Database System," Commun. of ACM, Nov. 76, Vol. 19, No. 11, 624-633.