Report Number: WUCS-79-4

1979-06-01

Concurrency Coordination in a Locally Distributed Database System

Authors: Gruia-Catalin Roman

A pipelined architecture for a locally distributed database system is proposed along with a simple concurrency coordination mechanism. The approach is based on the idea of serializing transaction processing throughout the database. The scheme is shown to require few coordination messages, to be deadlock free, to preserve database consistency, and to support recovery. Several performance related issues are also discussed.

CONCURRENCY COORDINATION IN A LOCALLY DISTRIBUTED DATABASE SYSTEM

Gruia-Catalin Roman

WUCS-79-4

June 1979

Department of Computer Science
Washington University
St. Louis, Missouri 63130

As appeared in Proceedings of the 1980 National Computer Conference, May 1980, pp. 269-273.

ABSTRACT

A pipelined architecture for a locally distributed database system is proposed along with a simple concurrency coordination mechanism. The approach is based on the idea of serializing transaction processing throughout the database. The scheme is shown to require few coordination messages, to be deadlock free, to preserve database consistency, and to support recovery. Several performance related issues are also discussed.

Keywords: distributed database, consistency, transaction, schedule

I. INTRODUCTION

This paper is concerned with locally distributed databases. A pipelined architecture for a multi-processor database system is proposed along with a concurrency coordination scheme. The architecture can be modelled as a collection of single-rooted directed acyclic graphs (DAG's). The root of each DAG is associated with a column of a cross-bar switch; the rows correspond to separate user groups. Each node of the DAG represents a processor, and each arc indicates a communication link between processors. The leaves of the DAG's are called data processors and may contain single files or larger portions of the database. The other nodes, called directory processors are assumed to store, in a distributed fashion, an index structure designed to maximize throughput while maintaining a relatively constant response time. For the sake of simplifying the exposition, changes to the index structure will not be considered until the end of Section IV.

Transactions are entered serially at the root of each DAG and the concurrency coordination mechanism enforces their "proper" pipelining through the networks. Each directory processor performs a routing function. When the leaves are reached, the requested data is read and passed back to the user who later sends the updates followed by a commit message. The sending of a commit message is fundamental to the approach; the fact that the user creates the updates is not. An optimized system would have the data and/or directory processors create the updates. The decision would be determined by some data transfer volume minimization strategy.

Section II of this paper is a short review of the terminology. The formal solution is described in Section III. Some performance issues are discussed in Section IV , and a short summary is provided in Section V.

II. DEFINITION OF TERMS

For the definition of the terms "database," "transaction" and "consistency" the reader is referred to [G1]. Each transaction is assumed to have three phases:

(1) Data Selection Phase -- A selection criterion or query is passed to the root nodes of the networks (DAG's). Each root node distributes the selection process among its successors by initiating appropriate subqueries. Later, when the answers to the subqueries return, it concatenates and sends them to the user along with an end-of-selection signal. All other nodes repeat the same scenario throughout the network.

(2) Data Modification Phase -- Upon receipt of the required data, the necessary updating is carried out.

(3) Data Commitment Phase -- The updated data is sent back into the network to permanently replace the old copies. As soon as all updates are acknowledged, a final commit message is issued by the user.

For the time being, the single network case is considered. The extention to multiple networks is made in the conclusion of Section III. Some notation is introduced next.

- Xz denotes a single node (data or directory processor) in the network. XO is used to represent the root node.
- Ti represents the transaction i. Each transaction is assumed to have a unique identifier i (e.g., distinct time stamps). All messages exchanged by processors in behalf of Ti carry the identifier i.
- Qi denotes any non-null message associated with the transaction Ti during the data selection phase; it is called a <u>query message</u>. The notation $Qi \subset Tj$ means i=j.

- i is used to represent a null query message associated with Ti.
- Sk is called a schedule and is defined as an arbitrary sequence of query messages. Sk is a <u>serial schedule</u> iff no two query messages in Sk have the same identifier, i.e.,

  $(Sk)/p \subset Ti \;\&\; (Sk)/q \subset Tj \;\&\; p \neq q \Rightarrow i \neq j$

  where (Sk)/r is the r'th query message in Sk.

III. CONCURRENCY COORDINATION

It is generally known ([E1],[G1]) that concurrent execution of several transactions can result in violations of database consistency due to undesirable interferences among transactions. The network organization suggested in this paper adds a new level of complexity since each transaction is implemented using concurrency, thus requiring coordination even when a single transaction exists in the system. The coordination scheme described below handles both types of problems. Furthermore, it assumes no centralization; it is simple; it involves a small synchronization overhead and, in contrast to other approaches, ([B1],[R1],[S2]), it is architecture specific.

The idea behind the scheme is the following. Given any two data processors, Xp and Xq, for any two transactions Ti and Tj which access (read or write) data from both Xp and Xq, the two processors are forced to see Ti and Tj in the same order. This can be accomplished by requiring the behavior of every node to be describable by the functions below:

(S is the set of all possible serial schedules)

MATCH: $S^n \rightarrow S$

SO = MATCH (S1,S2,...,Sn)
where
(SO)/ℓ⊂Ti iff (Sj)/ℓ⊂Ti for j = 1,2,...,n
(SO)/ℓ =Qi iff ∃k: (Sk)/ℓ =Qi
i iff (Sj)/ℓ =i for j = 1,2,...,n

TRIM: $S^n \times S \rightarrow S^n$

(S1',S2',...,Sn') = TRIM(S1,S2,...Sn; SO)
where
Sj=Sj".Sj' for j = 1,2,...,n
(Sj")/ℓ⊂Ti iff (SO)/ℓ⊂Ti for j = 1,2,...,n

FORK: $S \rightarrow S^n$

(S1,S2,...,Sn) = FORK(SO)
where
(Sj)/ℓ = Qi or i if (SO)/ℓ ⊂ Ti & (SO)/ℓ ≠ i
= i if (SO)/ℓ = i for j=1,2,...,n

For every node Xu, the MATCH function provides the mechanism by which a new schedule is generated from the messages received by Xu through combining messages associated with the same transaction. TRIM is used to remove from the input queues of Xu all messages that have been used by MATCH. The function FORK is nondeterministic and simulates the process by which Xu decides what query messages to send to its successors as a consequence of processing a query message from the schedule generated by MATCH. To assure correct propagation of the schedule received by XO, each Xu sends, for every message processed, query messages to all its successors; for those not needed in the data selection process, a null message is sent. Since each node processes and puts out messages in the same order, the initial total order over the transactions Ti is maintained throughout the network.

PROPOSITION: Given the fact that the entry node XO is presented with a serial schedule SO, every node Xu will execute an order equivalent serial schedule Su:

$$(Su)/\ell \subset Ti \text{ iff } (SO)/\ell \subset Ti.$$

Proof by induction:

(0) The proposition holds trivially for XO.

MATCH(SO) = SO

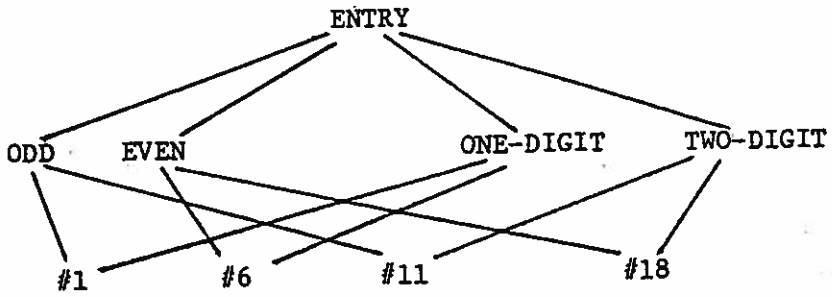
TRIM(SO:SO) = $\emptyset$

FORK(SO) = (SO1,SO2,...,SOn) --where SOk represents the schedule generated by SO for its k'th successor and is order equivalent to SO.

(N) Let us assume the proposition to be true for nodes at distance N from XO. Distance is defined as the length (number of links) of the longest path from XO to the particular node.

(N+1) Given any node Xu at distance N+1, due to assumption (N) all its input schedules are order equivalent to SO. By applying again the definitions of MATCH, TRIM, and FORK, one establishes the propositions to be true for the level N+1.

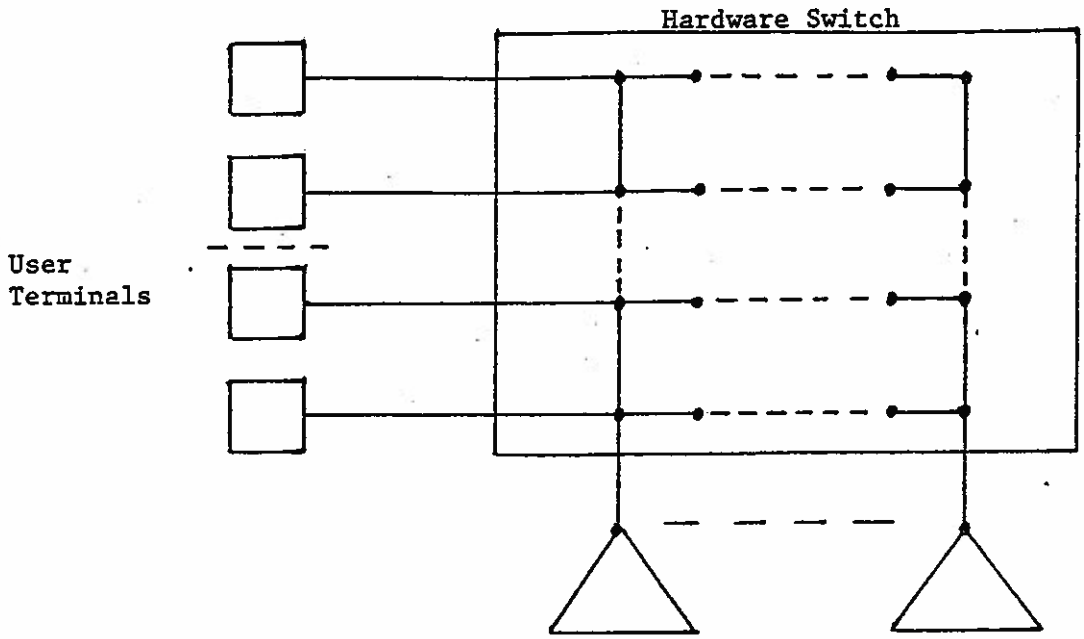
Example. For exemplification purposes, a database is described and the sample schedules resulting from processing two transactions, T1 and T2, are listed.



ENTRY:	(T1 : add 2 mod 6 to one-digit numbers).(T2 : list even numbers)		Q1.Q2
ODD:	(T1 : nil).(T2 : nil)		1.1
EVEN:	(T1 : nil).(T2 : list even numbers)		1.Q2
ONE-DIGIT:	(T1 : add 2 mod 6 to one-digit numbers).(T2 : nil)		Q1.2
TWO-DIGIT:	(T1 : nil).(T2 : nil)		1.2
#1:	(T1 : nil).(T2 : nil) (T1 : add 2 mod 6).(T2 : nil)	= (T1 : add 2 mod 6).(T2 : nil)	Q1.2
#6:	(T1 : nil).(T2 : list) (T1 : add 2 mod 6).(T2 : nil)	= (T1 : add 2 mod 6).(T2 : list)	Q1.Q2
#11:	(T1 : nil).(T2 : nil) (T1 : nil).(T2 : nil)	= (T1 : nil).(T2 : nil)	1.2
#18:	(T1 : nil).(T2 : list) (T1 : nil).(T2 : nil)	= (T1 : nil).(T2 ; list)	1.Q2

NOTE: In this example, the updates are carried out at the nodes. However, if node #6 sends the data to the entry point to be updated by the user processor that initiated the particular transactions, it will not process the next message until the result comes back and is committed throughout the network.

The extension of this coordination scheme from the one to several networks requires that all root nodes receive transactions spanning across nets in the same order. This can be achieved through the use of a hardware cross-bar switch.



Separate Database Networks

When a user terminal issues a transaction which concerns only one of the database networks, the query message transmission takes place as soon as the the vertical path becomes available. However, if the query involves more than one network, the user must request allocation of all needed paths before sending query messages to the various networks. This strategy will guarantee that the separate serial schedules are all consistent with each other, i.e., any two transactions occur in the same order in any serial schedule in which they appear together. Furthermore, if switching paths are viewed as resources and always allocated in the same order, the

possibility of deadlock is eliminated. At the same time, blocking within the switch can be minimized by allowing single network users to use paths which have been allocated to a multiple network user but are not yet in use.

IV. PERFORMANCE ISSUES

The architectural solution described in Section II has its justification in the application domain for which it is intended--medical information systems. Such systems tend to be confined to a single geographical location, grow relatively fast, require quick response, and exhibit a processing pattern dominated by data retrieval and creation rather than updates. As such, modifications of the directories, other than additions of new entries, can be assumed to be few. Therefore, the user should be willing, in those rare occassions, to pay an additional waiting penalty for coordinating the concurrent update of several directories.

The distribution of the index structure over several directory processors is meant to reduce the searching time through the use of concurrency in a pipelined-like fashion. The goal is to assure a constant average time through the addition of new directory and data processors when faced with transaction volume increases. However, the system's ability to handle the higher throughput relates not only to the number of processors being used but also to the "appropriateness" of the data and index distribution. Ideally, all data processors should be equally utilized. Furthermore, the searching load, within each net should be equally distributed among directory processors at equal distance from the root since the coordination scheme forces each processor to work at the rate of the slowest predecessor.

With respect to transaction recovery and roll-back, a transaction failure in some node could be signaled by passing a failure message to the user processor, which, in turn would send a "forget about my updates" message in place of the commit. Subsequently, the transactions would be started again with a new identifier. A node failure would have the effect of cancelling any transaction that requires its use.

V. SUMMARY

An architecture for a locally distributed database system was suggested. A simple solution to the problem of coordinating concurrent transactions within the database was presented. The solution requires no centralized control, is deadlock free, uses no locks, is fair, and involves little overhead.

REFERENCE LIST

[B1] Bernstein, P.A., Rothnie, J.B., Goodman, N., and Papadimitriou, C.A. "The Concurrency Control Mechanism of SDD-1: A System for Distributed Databases (The Fully Redundant Case)." IEEE Transactions on Software Engineering SE-4, No.3, pp. 154-167, May 1978.

[C1] Chamberlin, D.D., Boyce, R.F., and Traiger, I.L. "A Deadlock-Free Scheme for Resource Locking in a Data-Base Environment." Information Processing 74, North-Holland Publishing Company, pp. 340-343, 1974.

[E1] Eswaran, K.P., Gray, J.N., Lorie, R.A., and Traiger, I.L. "The Notions of Consistency and Predicate Locks in a Database System." CACM 19, No. 11, pp. 624-633, November 1976.

[G1] Gray, J.N. "Notes on Data Base Operating Systems." Research Report RJ2188(30001) IBM Research Laboratory, San Jose, California 95193, 1978.

[R1] Rosenkrantz, D.J., Sterns, R.E., and Lewis II, P.M. "System Level Concurrency Control for Distributed Database Systems. "ACM Transactions on Database Systems 3, No. 2, pp. 178-198, June 1978.

[S1] Stearns, R.E., Lewis II, P.M., and Rosenkrantz, D.J. "Concurrency Control for Database Systems." Proceedings of the 17th Annual Symposium on Foundations of Computer Science, pp. 19-32, 1976.

[S2] Stucki, M.J., Cox, J.R., Roman, G.-C., and Turcu, P.N. "Coordinating Concurrent Access in a Distributed Database Architecture." Proceedings of the Fourth Workshop on Computer Architecture for Non-Numeric Processing, August 1978.