TOWARDS A SELF-ADAPTING CENTRALIZED CONCURRENCY CONTROL ALGORITHM

Haran Boral
Microeletronics & Computer Technology Corporation
9430 Research Blvd., Echelon Bldg. #1, Suite 200
Austin, TX 78759

Israel Gold
Computer Science Department
Technion -- Israel Institute of Technology
Haifa 32000 Israel

ABSTRACT

We introduce the notion of self-adapting concurrency control algorithms -- concurrency control algorithms that consist of several rw and several ww synchronization techniques, and employ combinations of the techniques in a manner that attains a performance objective. We Consider synchronization techniques that use locking and certification. A general proof method for such algorithms is outlined and applied.

1. INTRODUCTION

Several centralized concurrency control algorithms have been proposed during the past several years. The majority of the algorithms are based, to one degree or another, on the Two Phased Locking method (2PL) [ESWA76] in which waiting is used to synchronize conflicting transactions; and on methods that allow conflicting transactions to run concurrently but use rollback in cases where inconsistent updates to the database could result [KUNG81] (known as Certification Methods because they certify a transaction for additional processing or commit, or cause it to abort).

At system design time a concurrency control algorithm is picked for the system (typically a 2PL variant) and incorporated into it. This design decision may be made based on some a priori knowledge of the expected use of the system or simply because the algorithm may appear to be (or actually may be) the best. Due to the complicated structure of software systems such as database management systems it is unlikely that the original algorithm incorporated into the system will ever be changed or even be improved, despite the fact that the system may be used under a variety of workload conditions in two separate settings or even in the same setting over a long time interval.

Recently researchers have begun to compare several different algorithms in an attempt to reach some conclusion concerning their operational merit. Naturally, if a clear cut

conclusion can be reached about one algorithm being "best" at almost all times then that algorithm should be employed by all database management systems. The studies range from the purely abstract [PAPA79] to more "conventional" ones where such measures as transaction throughput and cost of the concurrency control mechanism are evaluated [AGRA83], [CARE83], [GALL82]. The conclusions so far seem to indicate that variants of 2PL are cheap in terms of system resources required [CARE83] and lead to higher system throughput than certification algorithms. These studies, however, are by no means the last word since the simulations performed were not run under a wide variety of system workloads and parameters. In fact, not only does it appear as though no characterization of such workloads exists but that due to the changing application areas (e.g. artificial intelligence applications) the usefulness of such a characterization may be shortlived.

It is our thesis that the concurrency control mechanism in a database management system should be a versatile piece of software that has the ability to adapt itself to the system workload (or environment). We envision a mechanism that "knows" about locking and certification and uses its knowledge in one of the following ways:

(1) Use one of several algorithms known to it for all the transactions. Periodically switch from one algorithm to another (based on the system workload).

(2) Use several algorithms concurrently, each for a given predetermined class of transactions and ensure that conflicts between transactions from two different classes are resolved correctly.

A disadvantage of the first approach is that at any given moment exactly one algorithm is used to synchronize all the transactions. In an environment in which several transactions of different classes are run concurrently a concurrency control mechanism of this type may not perform well because of the need to change algorithms frequently. This approach has been investigated by Robinson [ROBI82].

In the second approach the allocation of algorithms to classes may be done statically (as was done in the SDD-1 concurrency control mechanism [BERN80][1]) or quasi-dynamically -- by using an initial static allocation that may change at run time. The quasi-dynamic approach seems to be the most promising of the three possibilities. A static allocation of algorithms may suffer because it is insensitive to the dynamics of the system.

We introduce the notion of integrated concurrency control algorithms (ICCAs) to implement the quasi-dynamic approach. An ICCA consists of a set of rw synchronization techniques and a set of ww synchronization techniques running concurrently. Each transaction may be mapped statically (before it is run) and dynamically (during its execution) to one rw technique and one ww technique.

---

[1] We envision a class as composed of transactions that share some common feature, i.e., "short writers." This is to be distinguished from the characterization of classes in [BERN80].

The model we use is the private workspace model of [BERN81]. We introduce the use of the PRE_WRITE -- a "temporary write" -- in addition to using DM_READs and DM-WRITEs to synchronize conflicting operations. Since PRE-WRITEs do not affect the database state the concurrency control mechanism has complete freedom in choosing how to use them in synchronizing conflicting operations thereby enabling it to control the level of concurrency in the system.

In this paper we show that it is possible to construct a dynamic concurrency control mechanism that employs both waiting and rollback, concurrently, to synchronize conflicting transactions. In Section 2 we give an overview of the database management system (DBMS) model used in the paper. Section 3 discusses the Transaction Manager (TM) model -- both data structures and the operations the TM performs on them on behalf of transactions. In Section 4 we present four synchronization techniques based on 2PL and Certification. The notion of an Integrated Concurrency Control Algorithm (ICCA) is introduced in Section 5. In Section 6 we present a formalization of the techniques and give the proof method used to show an ICCA correct. We conclude with summary of present work in Section 7.

2. THE DATABASE MANAGMENT SYSTEM MODEL

In this section we discuss the DBMS model. The model utilizes private workspaces allocated to active transactions to cache their previously read data items as well as those written by the transaction during its execution. Bernstein and Goodman [BERN81] and Kung and Robinson [KUNG81] used this model previously. The following description closely models that of [BERN81].

A centralized DBMS can be seen as composed of two components: A Transaction Manager (TM) and a Data Manager (DM). The TM controls interaction between users and the DBMS and is responsible for such functions as concurrency control. The DM is responsible for management of the database itself, i.e., accessing it. Two data manipulation operations are recognized by the DM: DM_READ(X) -- in which data item X is read; and, DM_WRITE(X,NEW_VALUE) -- in which the value NEW_VALUE is assigned to data item X in the database.

Users of the DBMS interact with it by running transactions. The TM maintains a private workspace for each active transaction in which copies of records read or written by the transaction are kept. From the DBMS point of view a transaction executes four types of operations: TRANS, READ, WRITE, and SNART. The actions taken by the TM upon receipt of these commands are described below.

TRANS: The TM initializes a private workspace for the transaction.

READ(X): X is a data item. If X already exists in the private workspace then its value is returned to the transaction by the TM. Otherwise the TM issues a DM_READ(X) operation to the DM. The current value of X is returned to the TM which writes it

in the transaction's private workspace and returns it to the transaction.

WRITE(X,NEW_VALUE): X is a data item. NEW_VALUE is a value to be assigned to X. The TM executes a PRE_WRITE(X,NEW_VALUE) operation on the transaction's private workspace. This has the effect of updating the previous value of X in the private workspace to NEW_VALUE if a copy of X existed in the private workspace. Otherwise, X is created in the workspace with the value NEW_VALUE. Note that a PRE_WRITE operation does not alter any values in the database itself. From a synchronization point of view each WRITE causes a PRE_WRITE to be executed.

SNART: The TM checks whether allowing the transaction to commit (by making its changes permanent in the database) will leave the database in a consistent state. In the event that it does not, the transaction will be aborted. Otherwise the TM issues a DM_WRITE command for every previously executed PRE_WRITE command. This has the effect of making the last change to X in the private workspace a (temporarily) permanent value in the database. After all DM_WRITEs have been issued the private workspace is discarded -- the transaction has completed. Fom a synchronization point of view all the transaction's DM_WRITEs are executed atomically.

A transaction execution can be seen as composed of two phases. In the first phase the transaction reads values from the database, performs various computations and writes results into its private workspace. In the second phase, which takes place after the transaction finishes all computations, the TM first goes through (a possibly empty) procedure to ensure that committing the transaction will not cause inconsistencies, and then (a possibly empty) sequence of writes to the database (as described above). It is important to realize that this second phase is atomic.[2]

The notion of a (logical) private workspace is basic to our work. All references to data items in the private workspace are made through the TM which gives it the power to control the concurrency level in the system. Thus, the notion of a private workspace presented in this paper differs from ,that used by Network based database management system, where the user program "contains" its own private workspace (or user work area) which can be accessed at any time independently of the database management system.

3. THE TRANSACTION MANAGER MODEL

In this section we examine in more detail the actions taken by the TM upon receipt of a request from a transaction. We describe the data structures involved as well as the operations performed on them. These, in turn, will be used in the next sections to describe the actions taken by the various synchronization techniques.

Two data structures are required by the TM for its operation. One is a graph (known as the Serialization Graph -- SG) that represents precedence relationships among conflicting transactions. Two transactions conflict if they access the same data item and one issues a WRITE request. The second is a table of flags (FT) in which a list of transactions and their modes of access to data items is maintained.

[2]The physical implementation of the commit procedure need not be atomic as long as it appears atomic to the outside world. Kung and Robinson [KUNG81] discussed several ways of implementing non-atomic commits. In this paper we will refer to atomic commits from a logical point of view.

A node in SG represents an active or a committed transaction. An edge (T_i, T_j) in the graph indicates that in any execution order transaction T_i precedes transaction T_j. SG is used to represent all such precedence relationshps regardless of whether they originated in the deadlock detection phase of 2PL or in the detection of a conflict in a Certification algorithm.

An entry in FT exists for every data item that has been accessed, and consists of several pairs <FLAG, TRANSACTION_IDENTIFIER>. Each pair identifies the transaction that accessed the data item and the mode of access (READ or WRITE). No restriction is placed on the number and/or type of pairs associated with a single data item in an entry. It is up to the concurrency control mechanism to interpret the pairs in a single entry and to decide how to use that information.

Three types of flags are recognized:

(1) An r-flag indicates that a DM_READ operation was executed on this item on behalf of the transaction holding the flag.

(2) A p-flag indicates that a still active transaction issued a WRITE request on this data item. At commit time of a transaction all its p-flags are converted to c-flags if the transaction is allowed to commit.

(3) A c-flag indicates that a DM_WRITE operation was executed on this data item on behalf of the committed transaction holding the flag.

A TRANS operation causes the TM to add a node to SG representing the new transaction. Edges are added to SG by the synchronization techniques as described in the next section.

At execution time a READ or a WRITE request received by the TM undergoes a possibly empty waiting phase then a possibly empty synchronization phase followed by its execution.

In the waiting phase some synchronization techniques may force the requesting transaction to wait until transactions that "hold" conflicting flags on the same data item have completed execution, whereas other techniques always enable continuation of the execution to the synchronization phase. Edges are added to SG to reflect the precedence relation imposed by the waiting.

In the synchronization phase the request is synchronized with conflicting operations from other transactions. The result of this synchronization may be abortion of the issuing transaction or continuation with execution. Edges are added to SG to reflect the precedence relation imposed by the execution of the request.

Execution of the request includes appending the appropriate flag to FT and issuing the appropriate DM_READ, operation to the DM, or executing a PRE_WRITE directly by the TM.

A SNART operation causes the TM to perform a possibly empty validation phase to ensure that allowing the transaction to commit will not leave the database in an inconsistent state.

The difference between the synchronization techniques presented in the next section is in the manner in which they act in each of the three phases described above. What is

common to all the techniques is that they update SG during each phase they undergo. For example, a technique that undergoes the waiting phase and discovers that transaction T_i must wait for transaction T_j will add the edge (T_j, T_i) to SG. Similarly, FT is also updated for every request.

Note that we have specified above various operations that add nodes and edges to SG and pairs to FT. Information about committed transactions remains in these data structures although it can be shown that it need not be kept indefinitely. All traces of aborted transactions, however, are removed from both SG and FT. That is, the node representing an aborted transaction in SG is removed along with all incoming and outgong edges. All pairs detailing the accesses made by the aborted transaction are also removed from FT.

4. SYNCHRONIZATION TECHNIQUES

The Decomposition Theorem of concurrency control [BERN81] enables the designer of a concurrency control mechanism to address himself to two subproblems, namely: synchronization of READ WRITE requests (rw-synchronization) and synchronization of WRITE WRITE requests (ww-synchronization) rather than to a single more complex problem -- that of concurrency control. An rw (ww) synchronization technique is defined to be the procedure that guarantees correct rw (ww) synchronization. The concurrency control mechanism must then ensure that the use of a given rw synchronization technique together with a given ww synchronization technique will yield serializable execution orders.

The decomposition theorem serves as the foundation for the notion of an integrated concurrency control algorithm. Rather than the use of one rw technique together with one ww technique as is the norm in working systems and the plethora of proposed algorithms, we suggest designing an algorithm that would be composed of several rw techniques and several ww techniques. The number of techniques to be used at a given instance is something that would be left up to the system designer. The decision may be made statically or dynamically as indicated in Section 1. In this section we will present two rw techniques and two ww techniques. In Sections 5 and 6 we will discuss their use in ICCAs and present the proof method to show their correctness.

4.1 Characterization of Synchronization Techniques

We propose two characterization parameters for synchronization techniques: Synchronization Strategy and Synchronization Time. By synchronization strategy we mean the method used to synchronize conflicting transactions. We differentiate between methods that use locking (in particular 2PL) and methods that use rollbacks (in our case certification methods). Synchronization between two conflicting transactions may be performed at the time the conflicts occurred (i.e., at execution time -- ET) or during the commit procedure (commit time synchronization -- CT) that each transaction must undergo

before its updates to the database take effect.

The meaning of ET synchronization is that the order in which READ and WRITE operations arrive is important and that the synchronization algorithm must take that order into account. In a 2PL algorithm this order is determined by the flags transaction hold and by maintaining a queue of waiting transactions for flagged data items.

The meaning of CT synchronization is that the arrival order of READ and WRITE operations is unimportant. Thus, the CT synchronization algorithm need only collect and maintain information about each transaction's conflicts at execution time to enable it to discover inconsistencies at commit time.

Using these two characterization parameters we can obtain four synchronization techniques for each of the two types of synchronization. For example, we may use certification synchronization at transaction execution time or transaction commit time for either the rw or ww synchronizations.

We use the following notation to represent the possible techniques. Each technique's name will be composed of three components: synchronization strategy (2PL or CERT), synchronization item (ET or CT), and synchronization type (rw and ww). For example, CERT-CT-ww is a technique that uses the certification approach to achieve synchronization at transaction commit time. Figure 1 summarizes all eight possible techniques. In Section 4.3 we give brief descriptions of the actions taken by four of the techniques -- some of the remainder are of no interest (e.g., 2PL-CT-*) whereas others (e.g., CERT-CT-rw) are omitted for reasons of complexity and paper's length. Before describing the techniques we introduce, in the next section, the notion of a transaction's WaitFor Set.

4.2 The WaitFor Set of a Transaction

From a correctness point of view a transaction that is waiting for a flag need not wait for all active transactions that hold conflicting flags to terminate. For example, if T_i using 2PL synchronization requests an r-flag on X and T_j also using 2PL holds a p-flag on X then clearly T_i should wait for T_j and this should be reflected by an edge in SG from T_j to T_i. If, however, T_k which uses a certification synchronization strategy holds a p-flag on X, from a correctness point of view we may choose to have T_i wait for T_k or not do so. Either way would be correct (provided the synchronization phase operates correctly). This decision is a policy decision and for that reason we chose not to make it at this point.[3] Rather, we allow the implementor to define a waitfor set of transactions in terms of concurrency level desired.

Definition 1, below, formalizes this notion.

---

[3] We were influenced to a great degree in this part of our design by Robinson's notion of separation of correctness and policy in concurrency control [ROBI82].

Definition 1: The WaitFor Set of a transaction T_i for rw (ww) synchronization, $WFS_{rw(ww)}(T_i)$ is the set of all transactions holding a conflicting flag on the data item T_i is attempting to access that T_i must wait for in its rw (ww) waiting phase. The Minimal WaitFor Set of a transaction T_i for rw (ww) synchronization $MWFS_{rw(ww)}(T_i)$ consists of only those transactions for which T_i must wait to ensure correctness of rw (ww) synchronization.

We say that transaction T_i is using strict 2PL policy for rw(ww) synchronization when $WFS_{rw(ww)}(T_i)$ = the set of all transactions holding a conflicting flag on the data item T_i is attempting to access. We say that transaction T_i is using standard 2PL policy for rw(ww) synchronization when $WFS_{rw(ww)}(T_i) = \{T_j \mid T_j$ is using 2PL for rw(ww) synchronization$\}$. If the same synchronization policy applies to rw and ww synchronization type designation of the type will be omitted. For transaction T_i using CERT synchronization strategy we define $WFS_{rw}(T_i) = WFS_{ww}(T_i) = \emptyset$.

|_|

The waitfor set of a transaction T_i is implementation dependent and may be defined by means of synchronization techniques, transaction classes, the concurrency level desired, and even as a function of T_i waiting time on a givn request or all its past requests.

4.3 The Synchronization Techniques

To facilitate the description of the various techniques below we utilize a compatibility-action matrix (c-a matrix). An entry in the matrix indicates how a synchronization technique interprets the values of existing flags on a data item when processing a request from a transaction to access that item. In addition, the entry specifies what edges are added to SG. The c-a matrices for all four techniques described below are shown in Figure 2.

4.3.1 RW Synchronization Techniques

Rw synchronization techniques synchronize between DM_READ and PRE_WRITE operations as well as DM_READ and DM_WRITE operations. In the following, if adding an edge to SG causes a cycle in the graph the transaction that caused the cycle is aborted, all its flags released and all information about it removed from SG.

4.3.1.1 2PL-ET-rw

Assume T_i is using 2PL-ET-rw and let T_j be an active transaction in $WFS_{rw}(T_i)$. The c-a matrix in Figure 2 for 2PL-ET-rw applies to the waiting phase. During its waiting phase T_i's request can not proceed to the synchronization phase as long as T_j owns a conflicting flag. The c-a matrix for CERT-ET-rw describes the actions taken in the synchronization phase.

4.3.1.2 CERT-ET-rw

Assume T_i is using CERT-ET-rw and let T_j hold a flag on data item X. T_i's waiting phase is null. During its synchronization phase, T_i

resolves conflicts with T_j's flags using a c-a matrix for CERT-ET-rw.

4.3.2 WW Synchronization Techniques

Ww synchronization techniques synchronize between conflicting PRE_WRITE operations of active transactions as well as PRE_WRITE operations of an active transaction and conflicting DM_WRITE operations of committed transactions.

4.3.2.1. 2PL-ET-ww

The meaning of the c-a matrix entries in 2PL-ET-ww is as in the matrix for 2PL-ET-rw. Conflicts between two active transactions are synchronized in the waiting phase of a request whereas conflicts between an active and a committed transaction are synchronized during the request's synchronization phase.

4.3.2.2 CERT-CT-ww

In this technique synchronization is performed at commit time. All WRITE requests are allowed to run unhindered. The c-a matrix for this technique applies to the commit phase of the transaction. The committing transaction T_i follows all transactions committed earlier and indicates that active writers will follow it (synchronization with active writers is required only when they use the 2PL-ET-ww synchronization technique).

5. INTEGRATED CONCURRENCY CONTROL ALGORITHMS

In this section we give a formal definition of an ICCA. We also outline the proof method used in showing an ICCA to be correct. In the next section we give the theoretical basis and give an example of a proof of correctness.

Definition 2: An Integrated Concurrency Control Algorithm (ICCA) is a triple (S_{rw}, S_{ww}, F), where

- S_{rw} is a non empty set of rw-synchronization techniques
- S_{rw} is a non empty set of ww-synchronization techniques, and
- F is a mapping function F: T -> S_{rw} X S_{ww}. At any given instance each T_i in T is mapped to exactly one S_{rw} technique and one S_{ww} technique.

|_|

Example 1:

```
ICCA1 = ({2PL-ET-rw,CERT-ET-rw},
         {CERT-CT-ww},
         F: if Ti is a reader then
              CERT-ET-rw X CERT-CT-ww
            else
              2PL-ET-rw X CERT-CT-ww
        )
```

|_|

An ICCA operates correctly if it allows only serializable execution orders and it avoids deadlock situations. How do we prove a given ICCA as correct? We need to show that each constraint placed on the total execution order by an operation is represented in SG regardless of the transactions involved, and more importantly, regardless of the synchronization technique used presently and/or techniques used to synchronize the conflicting transactions in the past. It will be shown subsequently that the four techniques introduced in the previous section indeed satisfy this condition.

6. FORMALIZATION

6.1 Serializability

Our model differs from those of others in our use of the PRE_WRITE operation to synchronize conflicting transactions. In particular, we saw that some of the techniques cause transactions to wait in the presence of a conflicting p-flag and others do not. In this section we review the basic serializability theory[4] results and show that our use of PRE_WRITE as a synchronization primitive does not affect known results. We also introduce several new precedence relations to be used in the proof of correctness of ICCAs.

Definition 3: Let $T = \{T_1, T_2, \ldots T_n\}$ be a set of transactions. E, the execution schedue of T, is modeled by L^S, the synchronization log of T, which consists of DM_READ, PRE_WRITE, and DM_WRITE operations in the order in which they were scheduled. L, the execution log of T, is derived from L^S by removing from it all PRE_WRITE operations.

|_|

In subsequent lemma and theorem statements we shall assume that T, E, L, and L^S as defined in Definition 3 are given. Furthermore, references to T_i and T_j are to any two transactions in T. A DM_READ(X) operation by transaction T_i will be denoted by $r_i[X]$. Similarly, $p_i[X]$ and $w_i[X]$ will denote PRE_WRITE and DM_WRITE operations (respectively) on X made by transaction T_i. Finally, $O_i[X] < O_j[X]$ means that $O_i[X]$ precedes $O_j[X]$ in L^S.

Definition 4: L^S is serializable if it is computationally equivalent to a serial synchronization log.

|_|

Theorem 1: L^S is serializable iff L is serializable.

|_|

Theorem 1 establishes that to obtain serializable synchronization logs it is sufficient to maintain serializable execution logs. That is, use of the PRE_WRITE operation as a synchronization primitive while maintaining serializable execution logs does not affect the basic theory.

Definition 5 sets the background for stating the Decomposition Theorem in our extended model.

Definition 5: For each data item X, we define the binary relations $-->_u$, where values for u are given below, as follows:

(1) $T_i -->_{rw} T_j$ if $r_i[X] < w_j[X]$ in L^S

(2) $T_i -->_{wr} T_j$ if $w_i[X] < r_j[X]$ in L^S

(3) $T_i -->_{ww} T_j$ if $w_i[X] < w_j[X]$ in L^S

(4) $T_i -->_{rp} T_j$ if $r_i[X] < p_j[X]$ in L^S

(5) $T_i -->_{pr} T_j$ if $p_j[X] < r_i[X] < w_j[X]$ in L^S

(6) $T_i -->_{wp} T_j$ if $w_i[X] < p_j[X]$ in L^S

(7) $T_i -->_{pw} T_j$ if $p_j[X] < w_i[X] < w_j[X]$ in L^S

[4] For the sake of brevity, we eliminate the definitions of the various terms, such as computational equivalence.

(8) $T_i \rightarrow_{pp} T_j$ if $p_i[X] < p_j[X]$ and $w_i[X] < w_j[X]$ in L^s

(9) $T_i \rightarrow_{rwr} T_j$ if $T_i \rightarrow_{rw} T_j$ or $T_i \rightarrow_{wr} T_j$

(10) $T_i \rightarrow_{rpr} T_j$ if $T_i \rightarrow_{rp} T_j$ or $T_i \rightarrow_{pr} T_j$

(11) $T_i \rightarrow_{pwp} T_j$ if $T_i \rightarrow_{pw} T_j$ or $T_i \rightarrow_{wp} T_j$

(12) $T_i \rightarrow T_j$ if $T_i \rightarrow_{rwr} T_j$ or $T_i \rightarrow_{ww} T_j$

(13) $T_i \rightarrow \rightarrow T_j$ if $T_i \rightarrow T_j$ or $T_i \rightarrow T_k$ and $T_k \rightarrow \rightarrow T_j$

|_|

The binary relations (1)-(3), (9), and (12) are exactly those defined in [BERN81]. The remaining relations are new relations introduced in this paper based on our use of the PRE_WRITE as a synchronization primitive. Clearly all of these relations can be derived from L^s.

Theorem 2 (Decomposition): Let $\rightarrow_{rwr}$ and $\rightarrow_{ww}$ be asociated with an execution schedule E modeled by L^s. E is serializable if:

(1) $\rightarrow_{rwr}$ and $\rightarrow_{ww}$ are acyclic, and

(2) There is a total ordering of the transactions consistent with all $\rightarrow_{rwr}$ and all $\rightarrow_{ww}$ relationships.

|_|

Lemmas 1 and 2 will be used subsequently.

Lemma 1: $\rightarrow_{rpr} \supseteq \rightarrow_{rw}$

Corollary 1: $\rightarrow_{rpr} \cup \rightarrow_{wr} \supseteq \rightarrow_{rwr}$

Lemma 2: $\rightarrow_{pwp} \supseteq \rightarrow_{ww}$

6.2 Formalization of the Techniques

The following lemma characterizes the correspondence between the relations defined above and edges in SG.

Lemma 3: The relations $\rightarrow_{rp}$, $\rightarrow_{pr}$, $\rightarrow_{rw}$, $\rightarrow_{wr}$, $\rightarrow_{pw}$, $\rightarrow_{wp}$, and $\rightarrow_{ww}$ derived from L^s, and SG maintained by the synchronization techniques satisfy the following conditions:

(1) If T_i is using 2PL-ET-rw then:
  (a) if $T_j \rightarrow_{wr} T_i$ then (T_j, T_i) is an edge in SG.
  (b) if $T_j \rightarrow_{rp} T_i$ then (T_j, T_i) is an edge in SG.
  (c) if T_j in $WFS_{rw}(T_i)$ then $T_i \rightarrow_{pr} T_j$ is not in $\rightarrow_{pr}$.
  (d) if T_j not in $WFS_{rw}(T_i)$ then if $T_i \rightarrow_{pr} T_j$ then (T_i, T_j) is an edge in SG.

(2) If T_i is using CERT-ET-rw then:
  (a) if $T_i \rightarrow_{pr} T_j$ then (T_i, T_j) is an edge in SG.
  (b) if $T_j \rightarrow_{rp} T_i$ then (T_j, T_i) is an edge in SG.
  (c) if $T_j \rightarrow_{wr} T_i$ then (T_j, T_i) is an edge in SG.

(3) If T_i is using 2PL-ET-ww then:
  (a) if $T_j \rightarrow_{wp} T_i$ then (T_j, T_i) is an edge in SG.
  (b) if T_j in $WFS_{ww}(T_i)$ and T_i in $WFS_{ww}(T_j)$ then $T_i \rightarrow_{pw} T_j$ is not in $\rightarrow_{pw}$.

(4) If T_i is using CERT-CT-ww then:
  (a) if $T_j \rightarrow_{ww} T_i$ then (T_j, T_i) is an edge in SG.
  (b) if $T_i \rightarrow_{pw} T_j$ then (T_i, T_j) is an edge in SG.

6.3 Showing ICCAs Correct

Definition 6: An ICCA = (S_{rw}, S_{ww}, F) is correct, if for every mapping of F the following conditions hold:

(1) Serializable synchronization logs are attained, and

(2) No deadlock results

To illustrate the proof method introduced in Section 5 consider $ICCA_2$ below. Since $ICCA_2$ consists of all four techniques, showing it correct will mean that other ICCAs that use subsets of the techniques are also correct.

$ICCA_2$ = ({2PL-ET-rw, CERT-ET-rw}, {2PL-ET-ww, CERT-CT-ww}, F_2)

Lemma 4: Let L^S be the synchronization log for an execution using $ICCA_2$. If T_j --> T_i then (T_j, T_i) is an edge in SG.

Proof: First we show that if T_j $-->_{rwr}$ T_i then (T_j, T_i) is an edge in SG (*)

F_2 may map T_i and T_j to 2PL-ET-rw and CERT-ET-rw in 4 ways:

(1) Let F_2: T_i --> 2PL-ET-rw, T_j --> 2PL-ET-rw

By Lemma 3.1.a if T_j $-->_{wr}$ T_i then (T_j, T_i) is an edge in SG.

By Lemma 3.1.b if T_j $-->_{rp}$ T_i then (T_j, T_i) is an edge in SG.

By Lemma 3.1, substituting T_j for T_i and T_i for T_j we have:

If T_i in $WFS_{rw}(T_j)$ then by Lemma 3.1.c T_j $-->_{pr}$ T_i is not possible, hence we can write if T_j $-->_{pr}$ T_i then (T_j, T_i) is an edge in SG.

If T_i not in $WFS_{rw}(T_j)$ then by Lemma 3.1.d if T_j $-->_{pr}$ T_i then (T_j, T_i) is an edge in SG.

(2) Let F_2: T_i --> 2PL-ET-rw, T_j --> CERT-ET-rw

By Lemma 3.1.a if T_j $-->_{wr}$ T_i then (T_j, T_i) is an edge in SG. By Lemma 3.1.b if T_j $-->_{rp}$ T_i then (T_j, T_i) is an edge in SG.

By Lemma 3.2.a substituting T_j for T_i and T_i for T_j we get if T_j $-->_{pr}$ T_i then (T_j, T_i) is an edge in SG.

(3) Let F_2: T_i --> CERT-ET-rw, T_j --> 2PL-ET-rw

By Lemma 3.2.c if T_j $-->_{wr}$ T_i then (T_j, T_i) is an edge in SG.

By Lemma 3.2.b if T_j $-->_{rp}$ T_i then (T_j, T_i) is an edge in SG.

By Lemma 3.1, substituting T_j for T_i and T_i for T_j we have:

If T_i in $WFS_{rw}(T_j)$ then by Lemma 3.1.c T_j $-->_{pr}$ T_i is not possible, hence we can write if T_j $-->_{pr}$ T_i then (T_j, T_i) is an edge in SG.

If T_i not in $WFS_{rw}(T_j)$ then by Lemma 3.1.d if T_j $-->_{pr}$ T_i then (T_j, T_i) is an edge in SG.

(4) Let F_2: T_i --> CERT-ET-rw, T_j --> CERT-ET-rw

By Lemma 3.2.c if T_j $-->_{wr}$ T_i then (T_j, T_i) is an edge in SG.

By Lemma 3.2.b if T_j $-->_{rp}$ T_i then (T_j, T_i) is an edge in SG.

By Lemma 3.2.a substituting T_j for T_i and T_i for T_j we get that if T_j $-->_{pr}$ T_i then (T_j, T_i) is an edge in SG.

In all possible mappings of T_i and T_j by F_2 we have that if T_j $-->_{rpr}$ T_i or T_j $-->_{wr}$ T_i then (T_j, T_i) is

an edge in SG. By Corollary 1 we conclude (*).

In the second part of the proof we have to show that if $T_j \rightarrow_{ww} T_i$ then (T_j, T_i) is an edge in SG. The proof is similar to that of the first part and is left out.

Theorem 3: $ICCA_2$ is correct.

Proof: Let E be Execution using $ICCA_2$ modeled by L^s. Let - -> be derived from L_s.

(1) By Lemma 4 if T_j --> T_i then (T_j, T_i) is an edge in SG. Since SG is maintained acyclic at all time it follows that --> is acyclic and by Theorem 1 we conclude that E is serializable.

(2) Whenever T_i waits for T_j upon rw or ww conflict on edge (T_j, T_i) is added to SG. Since SG is maintained acyclic at all time deadlock is prevented. Furthermore, since we know that T_i will follow T_j in any execution schedule (1) guarantees overall consistency.

7. CONCLUSIONS

7.1 SUMMARY OF CONTRIBUTIONS

In this paper we proposed a new approach to centralized concurrency control. Our intention is that a concurrency control mechanism would be a versatile piece of software that is both knowledgeable about different approaches to the problem and capable of selecting the right approach at a given time instance. We have shown that such mechanisms that rely on 2PL and certification methods can operate correctly.

The main difference between our mechanism and other proposals is our use of several techniques concurrently in a single algorithm. The key ideas proposed in this paper are:

(1) The use of several techniques concurrently in a single algorithm.

(2) Use of the PRE_WRITE operation as a synchronization primitive in addition to the previously used DM_READ and DM_WRITE operations. Conflicts involving a PRE_WRITE operation essentially foresee possible conflicts between DM_READs and DM_WRITEs. Different techniques interpret and use this information in a different manner.

(3) The two phases of request processing (waiting and synchronization) enable a clean transition from locking to non-locking based techniques.

(4) The notion of a transaction's waitfor set introduced in this paper. It is a means for controlling the concurrency level in the system.

7.2 Related Work

The primary influences on our work have been the concurrency control algorithm of Bayer et. al. [BAYE80] in which 2PL and a certification strategy were combined into a single algorithm; Bernstein and Goodman's work -- in particular their use of the decomposition theorem to derive several distributed algorithms that utilize 2PL and timestamps [BERN81]; Wilkinson's centralized algorithm for a local network in which certification and locking were also combined [WILK81]; and, Robinson's notion of separating policy from

correctness in the design of concurrency control algorithms [ROBI82].

To the best of our knowledge only Robinson [ROBI82] has proposed a mechanism that is similar to ours. In his proposal the concurrency control mechanism uses a single concurrency control algorithm at a time, and selects the proper algorithm from a set of algorithms available to it based on some parameter, e.g., system workload.

7.3 Present Work

Some additional theoretical work remains in characterizing the power of the model. However, at the moment we are engaged in the construction of a simulation model. In the short run we wish to arrive at a comparison of our algorithm with more conventional algorithms using an abstract measure such as level of concurrency. In the long run we are interested in deriving more useful measures such as effect of the algorithm on system throughput and transaction response time.

8. References

[AGRA83] Agrawal R., "Concurrency Control and Recovery in Multiprocessor Database Machines: Design and Performance Evaluation," PhD Dissertation, University of Wisconsin, (1983).

[BAYE80] Bayer R., H. Heller, and A. Reiser, "Parallelism and Recovery in Database Systems," ACM TODS, Vol. 5, No. 2, (1980).

[BERN80] Bernstein P.A., D.W. Shipman, and J.B. Rothni Jr., "Concurrency Control in a System for Distributed Databases," ACM TODS, Vol. 5, No. 1, (1980).

[BERN81] Bernstein P.A. and N. Goodman, "Concurrency Control in Distributed Database Systems," Computing Surveys, Vol. 13, No. 2, (1981).

[CARE83] Carey M.J., "Modeling and Evaluation of Database Concurrency Control Algorithms," PhD Dissertation, University of California Berkeley, (1983).

[ESWA76] Eswaran K.P., J.N. Gray, R.A. Lorie, and I.L. Traiger, "The Notions of Consistency and Predicate Locks in a Database System," Communications of the ACM, Vol. 19, No. 11, (1976).

[GALL82] Galler B., "Concurrency Control Performance Issues," PhD Dissertation, University of Toronto, (1982).

[KUNG81] Kung H.T. and J.T. Robinson, "On Optimistic Methods for Concurrency Control," ACM TODS, Vol. 6, No 2, (1981).

[PAPA79] Papadimitriou C.H., "The Serializability of Concurrent Database Updates," Journal of the ACM, Vol. 26, No. 4, (1979).

[ROBI82] Robinson J.T., "Design of Concurrency Controls for Transaction Processing Systems," PhD Dissertation, Carnegie Mellon University, (1982).

[WILK81] Wilkinson W.K., "Database Concurrency Control and Recovery in Local Broadcast Networks," PhD Dissertation, University of Wisconsin, (1981).

Srw Synchronization Technique	Sww Synchronization Technique
2PL-ET-rw	2PL-ET-ww
2PL-CT-rw	2PL-CT-ww
CERT-ET-rw	CERT-ET-ww
CERT-CT-rw	CERT-CT-ww

Figure 1: The Synchronization Techniques

2PL-ET-rw	T_j in $WFS_{rw}(T_i)$ r	p
T_i r	+	- T_j-->T_i
T_i p	- T_j-->T_i	*

2PL-ET-ww	T_j in $WFS_{ww}(T_i)$ p	T_j c
T_i p	- T_j-->T_i	+ T_j-->T_i

CERT-ET-rw	T_j r	p	c
T_i r	+	+ T_i-->T_j	+ T_j-->T_i
T_i p	+ T_j-->T_i	*	*

CERT-CT-ww	T_j using 2PL-ET-ww p	T_j c
T_i c	+ T_i-->T_j	+ T_j-->T_i

LEGEND	
+	request granted
-	request not granted on conflicts with active transactions
*	irrelevant

Figure 2: Compatibility-Action Matrices for the Synchronization Techniques