FURTHER OPTIMISM IN OPTIMISTIC METHODS
OF
CONCURRENCY CONTROL

by

Rakesh Agrawal
David J. DeWitt

Computer Sciences Technical Report #470
March 1982
Further Optimism in Optimistic Methods of Concurrency Control

Rakesh Agrawal
David J. DeWitt

Computer Sciences Department
University of Wisconsin

This research was partially supported by the National Science Foundation under grant MCS78-01721 and the Department of Energy under contract #DE-AC02-81ER10920.
ABSTRACT

A family of concurrency control mechanisms is presented that has the potential of permitting additional concurrency in the optimistic methods of concurrency control proposed by Kung and Robinson.
1. Introduction

Mechanisms to control concurrent access to a shared database by multiple transactions have recently received a great deal of attention. Most of the proposed mechanisms are based on some explicit or implicit locking scheme [Esua76,Gray78]. These methods can be termed pessimistic as they assume that conflicts will occur frequently between transactions. These mechanisms attempt to prevent conflicts from occurring by providing a facility whereby a transaction can deny access to certain portion of the database to potentially conflicting transactions.

Recently, Kung and Robinson [Kung81] have proposed two families of non-locking concurrency control mechanisms. These techniques can be termed optimistic since they assume that conflicts between transactions are infrequent. When conflicts do occur between two transactions, transaction backup is used to resolve the conflict (i.e. one of the transactions is aborted).

In this paper, we propose another family of non-locking concurrency control mechanisms that extends those proposed by Kung and Robinson and that has the potential of permitting additional concurrency. The organization of rest of the paper is as follows. In Section 2, we briefly review the Kung-Robinson proposal, particularly the three validation conditions that give rise to the two families of concurrency control mechanisms. In Section 3, we propose a new validation condition that will allow greater concurrency. Section 4 describes the concurrency control mechanism that uses the proposed validation condition. In Section 5, we present our conclusions and suggestions for future
research.

2. Kung-Robinson Proposal

In optimistic methods of concurrency control, reads are completely unrestricted as reading can never result in a loss of data integrity. Writes are however severely restricted. As proposed in [Kung81], each transaction consists of two or three phases: a read phase followed by a validation phase and, possibly, a write phase. During the read phase, all writes are performed on local copies. Then, if the transaction can be validated during the validation phase, the local copies are made global during write phase.

Kung and Robinson have proposed the following validation conditions based on the notion of serial equivalence [Eswa76,Papa79,Stea76]. Assume each transaction $T_i$ is assigned an unique integer transaction number $t_i$ during its execution. To insure that an equivalent serial schedule exists in which $T_i$ precedes $T_j$ whenever $t_i < t_j$, the following criterion is used to validate a transaction $T_j$ with transaction number $t_j$: For all $T_i$ with $t_i < t_j$, one of the following conditions must hold:

1. $T_i$ completes its write phase before $T_j$ begins its read phase.

2. $T_i$ completes its write phase before $T_j$ starts its write phase and the write set of $T_i$ does not intersect the read set of $T_j$.

3. $T_i$ completes its read phase before $T_j$ completes its read phase and the write set of $T_i$ does not intersect the read set or the write set of $T_j$. 
3. A New Validation Condition

To enhance performance of the Kung-Robinson algorithm, we propose the addition of the following validation condition: If for each transaction $T_j$ with transaction number $t_j$ and for all $T_i$ with $t_i < t_j$,

(4) $T_i$ completes its read phase before $T_j$ begins its write phase and the write set of $T_i$ does not intersect the read set or the write set of $T_j$.

then there exists a serially equivalent schedule in which $T_i$ comes before $T_j$.

The fact that this validation condition permits more concurrency than condition (3) is illustrated by Example 1 below.

```
<table>
<thead>
<tr>
<th>R</th>
<th>---------------</th>
<th>V</th>
<th>W</th>
</tr>
</thead>
<tbody>
<tr>
<td>T_j</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>R</td>
<td>---------------</td>
<td>V</td>
<td>W</td>
</tr>
<tr>
<td>----</td>
<td>-----------------</td>
<td>----</td>
<td>----</td>
</tr>
<tr>
<td>T_i</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
```

Example 1

Assume that $T_i$ begins before $T_j$, that $t_i < t_j$, and that the write set of $T_i$ does not intersect the read set or write set of $T_j$. If validation condition (3) is used, then since $T_i$ completes its read phase after $T_j$ completes its read phase, $T_j$ cannot be validated and must be aborted. Validation condition (4), however, permits $T_j$ to be validated since since $T_i$ completes its read phase before $T_j$ begins its write phase. It is important to note that condition (4) still ensures a serially equivalent schedule as $T_i$ does not affect read or write phase of $T_j$ by the second part of the condition and $T_j$ does not affect the read phase of $T_i$ by the first part of the condition.
4. Proposed Optimistic Concurrency Control Mechanism

4.1. Read and Write Phases

In this section we describe a concurrency control mechanism that uses validation conditions (1), (2), and (4). As in [Kung81], we assume, for the sake of simplicity, that all objects are of same type. Objects are manipulated by the following functions:

(1) **Create** - creates an object and return its name
(2) **Delete**(n) - deletes object n
(3) **Read**(n,i) - reads item i from object n
(4) **write**(n,i,v) - writes v as item i of n

All transactions are bracketed with a *begin* and a *end* call, and all use syntactically identical procedures tcreate, tdelete, tread, and twrite. The body of a transaction constitutes the read phase and *end* signals the beginning of the validation phase. The concurrency control maintains various sets of object names accessed by each transaction.

The semantics of tcreate, tdelete, tread, and twrite is same as in [KUNG81] and are presented below for the sake of completeness.

```haskell
  tcreate = (n := create;
              create-set := create-set U \{n\};
              return(n))
```
twrite(n,i,v) = (
    if n ∈ create-set then write(n,i,v)
    else if n ∈ write-set then write(local-copies[n],i,v)
    else (local-copies[n] := copy(n);
          write-set := write-set U {n};
          write(local-copies[n],i,v))
)

tread(n,i) = (read-set := read-set U {n};
    if n ∈ write-set then return(read(local-copies[n],i))
    else return(read(n,i))
)

tdelete(n) = (delete-set := delete-set U {n} )

During the write phase, all local copies become global, all created objects become accessible, and all deleted objects become inaccessible. The cleanup procedure deletes all inaccessible objects and the local copies.

4.2. Validation

To implement validation condition (4), the concurrency control mechanism must maintain two global sets that are initialized to be empty:

Global-W-active: transaction ids of transactions that have completed their validation phase and are in the write phase.

Global-V-active: transaction ids of transactions that have completed their read phase and are in the validation phase.

As we will be shown below, during the validation of a transaction, the transaction makes a local copy of both of these sets.

Next we describe the semantics of tbegin and tend. Assume that T_j is the transaction to be validated and that ID_j is the identifier of the transaction. The variable tnc is a global integer that is used to assign transaction numbers. At any instant in time tnc reflects the number of the last transaction
to have committed (ie. finished both its validation and write phases). The symbols < > are used to bracket those sections of the algorithm that must be contained in a critical section.

tbegin = {
    create-set := Ø;
    read-set := Ø;
    write-set := Ø;
    delete-set := Ø;
    start-tn := tnc)

tend = {
    < finish-tn := tnc;
    local-W-active := Global-W-active;
    local-V-active := Global-V-active;
    Global-V-active := Global-V-active U {ID_j};
    >
    valid := true; /* Assume that the transaction will be validated */
    for t := start-tn + 1 to finish-tn do
        if (write-set of T_t | read-set of ID_j ≠ Ø) then
            valid := false;

    for i ∈ local-W-active do
        if (write-set of T_i | read-set or write-set of ID_j ≠ Ø) then
            valid := false;

    if valid then
        < local-W-active := local-W-active ∩ global-W-active;
        global-W-active := global-W-active U {ID_j};
        >
        for i ∈ local-W-active do
            if (write-set of T_i | read-set or write-set of ID_j ≠ Ø)
                then valid := false;

    if valid then
        ((write phase);
        < tnc := tnc + 1;
        tn := tnc;
        global-W-active := global-W-active - {ID_j};
        global-V-active := global-V-active - {ID_j};
        >
        (cleanup))

    else
        < global-W-active := global-W-active - {ID_j};
        global-V-active := global-V-active - {ID_j};
        >
        (backup)))
Multi-stage validation for optimization as proposed in [KUNG81] is also possible in our scheme. This gives rise to a family of concurrency control mechanisms depending on the number of stages.

5. Conclusions

In this paper, we have proposed a new validation condition for the Kung-Robinson [KUNG81] optimistic concurrency control mechanism that permits additional concurrency among concurrent transactions. We have also presented the concurrency control mechanism that uses the proposed validation condition. More research, however, is required to determine those situations in which the different mechanisms would be useful. An interesting extension would be to design a transaction manager that could dynamically choose the optimum control mechanism.

References


Vol. 6, 2 (June 1981), 213-226.
