Problems of Optimistic Concurrency Control in Distributed Database Systems

G. Schlageter
FernUniversität
Praktische Informatik
Postfach 940
D 5800 Hagen
West Germany

In [SCH81] some aspects of optimistic concurrency control (CC) in distributed database systems have been discussed, some important problems have, however, not been dealt with in sufficient detail in the paper but only in the oral presentation. Thus, the reader of the conference proceedings might get the impression that optimistic CC, as introduced in [KUR79], is very easily adapted to distributed systems. Unfortunately, when looking into the problem more thoroughly, the contrary turns out to be true. This note is meant to highlight the main problems of optimistic CC in distributed database systems. We cannot discuss possible solutions in detail - this will be done elsewhere.

1. OPTIMISTIC CC [KUR79]

In this note, as in [SCH81], we only consider optimistic CC as introduced in [KUR79]. Other "optimistic" methods, like [BAD79], are not discussed.

The basic idea of optimistic CC of the [KUR79] type is as follows:

Each transaction proceeds to its end without consideration of any parallel transaction; it works on local copies of the database objects. However, the transaction (say T) has to do *validation* before committing: it has to check whether its read set conflicts with the write set of any other transaction which has committed while T was active. In case no conflict is detected, T writes its objects and commits, otherwise T is backed up and restarted.

It is essential that validation and write are one critical section in the usual sense in the synchronization field. One can easily prove that whenever T' commits before T, and no conflict of the above type is detected by T, then an equivalent serial execution is T' before T. Note that positive validation does not mean that there is no conflict at all.

2. EXTENSION OF OPTIMISTIC CC TO DISTRIBUTED SYSTEMS

We assume that validation for a global transaction, say T, is done on a local basis, i.e. each subtransaction of T does validation independently of the other subtransactions of T, and signals the result to the coordinator of T. The coordinator can initiate commit only if all local validations are successful.

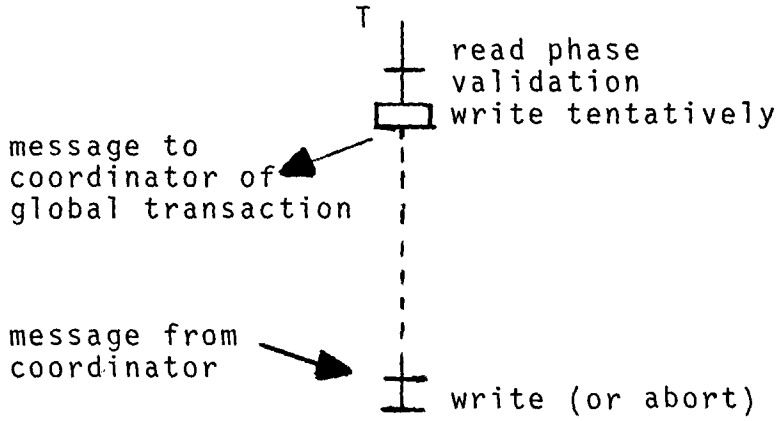
There are two sources of problems:

(1) for global transactions validation and write cannot be a critical section (because, after validation, communication is required before write can be performed)

(2) validation is asymmetric, such that the sequence of validations is essential.

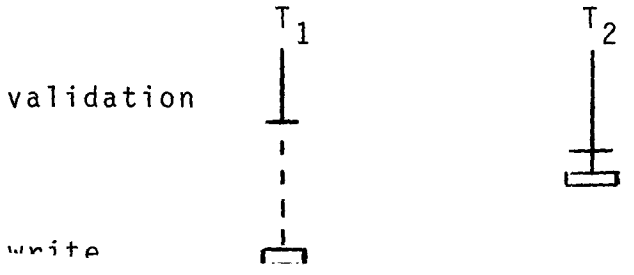
Problem (1):

One has to find a mechanism to substitute the critical section <validation-write> for global transactions. This problem was discussed in detail in [SCH81], and a simple solution was presented. Each subtransaction of a global transaction behaves as follows:



The idea for the solution is to extend validation. During validation of T not only those transactions are considered which have committed during the read phase of T, but also those which, at the point of validation, have tentatively written (and not yet written finally).

Because of the separation of validation and write the sequence of the validation is no longer equivalent to the sequence of the writes. Consider an example:



Since the write of T1 does not immediately follow the validation, we may have a dependency T1 → T2 (T1 reads an object which is written by T2 at a later point of time) and a dependency T2 → T1 (T2 writes an object which is written by T1 at a later point). This situation is not serializable.

There seem to be at least two ways of approaching this problem:

- enforce equivalence of the sequences of validations and writes (i.e. T2 would have to wait until T1 has written)

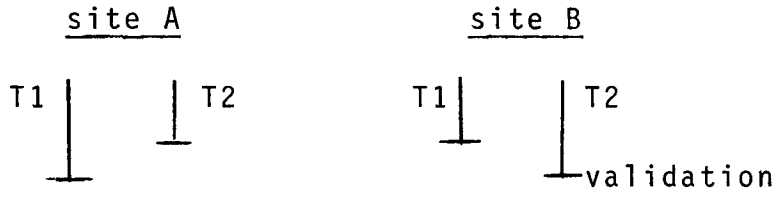
- extend validation such that T performs the following test against a parallel transaction t:

  $rs(T) \cap ws(t) \cup$
  $ws(T) \cap ws(t) = \emptyset$

where rs stands for read set and ws stands for write set.

Problem (2):

Consider two global transactions running in parallel. Since validation is done on a purely local basis and is not coordinated, the following situation may occur:



T1 does validation on site A after T2, and before T2 on site B. The consequence is: there may be dependencies according to the performed operations such that we have T2 → T1 on A, but T1 → T2 on B. Though the local situations are serializable, T1 and T2 form a non-serializable system.

Again, there seem to be at least two potential solutions:

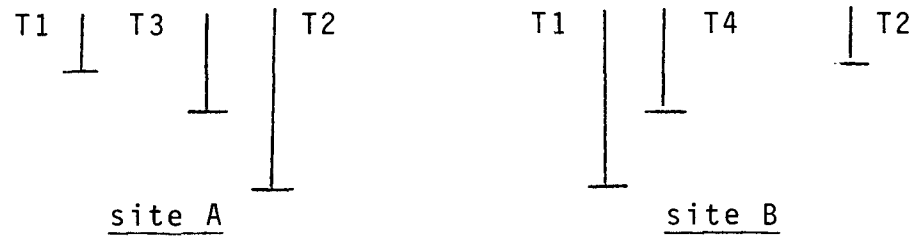
POTENTIAL APPROACH 1:

Include the read sets into validation, i.e. T performs the test

$rs(T) \cap ws(t) \cup$
$ws(T) \cap ws(t) \cup$
$ws(T) \cap rs(t) = \emptyset$

The unfortunate point is that now not only the write sets of committed transactions must be recorded, but also the read sets.

The overhead for keeping read sets may be acceptable if it can be restricted to global transactions. One has to be very careful about this point, because in this case local transactions may poduce "transitive" dependencies between global transactions which otherwise do not conflict at all. This problem has been brought to my attention by S. Ceri. Consider an example:



T1 and T2 are global transactions, T3 and T4 are local. Validation of T2 on site A and T1 on site B are positive, especially there is no conflict between T1 and T2. If the read set of the local transactionsis not considered, and if local transactions do not consider the read sets of global transactions, the following can happen on site A: T1 reads an object which is afterwards written by T3; T3 reads an object which is afterwards written by T2; we thus have

T1 → T3 → T2.

On site B, we may have in a similar way:

T2 → T4 → T1.

This is a non-serializable situation.

At least, the local transactions have to consider the read sets of global transactions: this would prevent T1 → T3 and T2 → T4 from happening, and thus, because there is no conflict at all between T1 and T2, would produce a serializable execution.

POTENTIAL APPROACH 2:

Enforce that the sequence of validations between any two global transactions is the same on all sites.

This could be achieved by imposing a global order on the global transactions, which could e.g. be a global transaction identifier <time of creation, site number>. This type of identifier has been used for other purposes also. Time of creation of the transaction, i.e. its coordinator, is local time; the clocks of the sites need not to be synchronized tightly.

With the total order given, the rule for validation might be: T is only allowed to validate if T > T' for all T' which wrote tentatively on this site so far (T' is "older" than T). Otherwise T has to be backed up.

With a rule of this type the read sets need not to be kept for later validations.

CONCLUSION

When trying to extend optimistic CC to distributed database systems one has to cope with some non-trivial synchronization problems due to the fact that validations and writes of a global transaction have to be coordinated in some way. The simplicity of the approach in centralized systems is lost to some extent, though there are solutions which do not depart much from the original mechanism. This note outlined some possible solutions.

More work is required to get a better understanding of where optimistic CC is appropriate. This includes, as a most important point, the investigation of performance aspects and comparison with other CC methods.

Acknowledgement: I wish to thank S. Ceri for his comments on my presentation at VLDB 81, and U. Prädel and R. Unland for the discussion about the subject.

References

[BAD79] Badal, D.Z.: Correctness of Concurrency Control and Implications in Distributed Databases. Proc. COMPSAC79, Chicago 1979.

[KUR79] Kung, H.T., Robinson, J.T.: On optimistic methods for concurrency control. Proc. VLDB, Rio de Janeiro, Oct. 1979, Conf.

[SCH81] Schlageter, G.: Optimistic Methods for Concurrency Control in Distributed Database Systems. Proc. VLDB81, Cannes 1981.