Copyright Notice

Parallellism in multi-record transactions.

by E.W.Dijkstra and C.S.Scholten

We consider a data base comprising a great number of individual records and transactions to be carried out on this data base. Each transaction is a finite computation, involving a number of these records: the computation to be carried out -and even the identity of the records involved- will in general be dependent on the initial state of the data base when the transaction is initiated. When the data base grows, the following conflict emerges: on the one hand one may expect the number of transactions to be carried out to grow as well, on the other hand the growing data base will make individual record selection a more and more painful process, slowing down the individual transaction executions. Comes the moment that the stream of transactions, carried out one after the other, no longer fits in real time. To solve this real time problem we must be willing to carry out a number of transactions in parallel. This paper is devoted to the logical problems that then emerge.

The purpose of this paper is twofold: firstly to isolate (and to solve to a certain extent) the logical problems involved and secondly, to demonstrate the viability of our top-down approach in problem solving. This means that those readers that are unfamiliar with the top-down approach but are familiar with a number of these logical problems, must be patient. If they find us ignoring a number of practical considerations in the beginning, they should read on quietly: there is a fair chance that they will be taken into account in due time.

The first model.

In the purely sequential execution of the transactions, we can execute the transactions in the (supposedly unique) order in which they are requested and at any moment in time there is at most one transaction under execution. In our first model, we still assume that the requests for transactions reach the system in a unique order and with a speed regulated by the system in such a way that the system can cope with the requests. We admit, however, that at any moment in time the number of transactions currently under execution may be larger than one, although not exceeding some given finite upper bound. The

execution of a transaction extends from the moment that the system has acknowledged the request for the transaction until the moment that the system has completed the transaction.

In the purely sequential execution, the system's net reaction to a number of transaction requests may depend on the order in which the transactions are requested. In the case of parallellism we do not require that the system's net reaction is identical to that of the sequential system when faced with the stream of requests in the order in which the parallel system has acknowledged these requests. We do require, however, that it is possible to order the requests in such a way that the net reaction of the sequential system faced with the requests in that order will be identical to the reaction of the parallel system. (In many cases, viz. when we have two mutually non-interfering transactions, this order need not be unique.)

Our parallel system has three main obligations; it has to prevent 1) undesired interference, 2) deadlock and 3) individual starvation.

ad 1.

We assume each transaction identified only for the period of its execution. Let $T[i]$ be a transaction currently under execution, let $M[i]$ be the set of records manipulated up till now by $T[i]$. This implies that during the execution of $T[i]$, the set $M[i]$ can never decrease, until transaction $T[i]$ is terminated and $M[i]$ ceases to exist. We can guarantee the absence of undesirable interference when at any moment in time

$$i \neq j \quad \Rightarrow \quad M[i] \cap M[j] = \emptyset \qquad 1)$$

i.e. for two different transactions the intersection of the corresponding sets M is empty.

ad 2.

If a and b are two different records and for $i \neq j$ we have at a given moment

$$a \in M[i] \qquad \text{and} \qquad b \in M[j]$$

then we will find ourselves in trouble when the progress of $T[i]$ requires record b to be added to $M[i]$ and also the progress of $T[j]$ requires record a to be added to $M[j]$, for then there is no way in which $T[i]$ or $T[j]$ can progress without violating condition 1. This is called "deadlock". If we insist on the

absence of the danger of deadlock -and we do- the above observation tells us that without any further knowledge about the future requirements of the transactions, parallellism is impossible. We therefore associate with each transaction T[i] a set F[i] of records, containing all the records that may possibly be added to M[i]. (Note that this definition implies $M[i] \cap F[i] = \emptyset$.)

When the current transactions can be renumbered such that

$$i < j \quad \Rightarrow \quad F[i] \cap M[j] = \emptyset \qquad 2)$$

the danger of deadlock is absent, for then T[0] can be carried to completion and after that the new T[0] etc. We call the situation "safe" when ~~the current~~ besides relation 1) transactions can be renumbered such that relation 2 holds ~~and~~ We shall keep the system in a safe state. From the above we can conlude that decrease of the set F[i] -as a result of progress of T[i]- will leave a safe situation safe; it furthermore follows that such a decrease is something to be encouraged, because as long as F[0] = the universe, all M[j] with $j > 0$ must be empty, i.e. parallellism is not possible.

When we start each transaction with its F equal to the universe and insist that T[i] can only add a record to M[i] by transferring it from F[i], then this is the only transition that might violate condition 1 or the safety, i.e. this is the only place where it might be necessary to hold up the further execution of the transaction,"to put the transaction to sleep". The "counter-occurrences", on account of which a sleeping transaction could be woken up again are ~~either the explicit decrease of its F by a progressing transaction or by termination of a progressing transaction~~ when a another progressing transaction decreases its F explicitly or (un) terminates.

In the above we have assumed that for each transaction, F would start equal to the universe and would only decrease. Because this set is so huge, one could think that it could be profitable to divide the execution of a transaction into two successive phases, a first phase in which F is still allowed to grow and a second phase in which this is no longer permissible. But as far as the avoidance of deadlock is concerned, such a transaction is equivalent to one with F equal to the universe during the first phase, decreasing F to the stated amount upon the transition from the first to the second phase.

ad 3.

Our system has to allocate records to transitions. When the allocation strategy is such that each request of a F → M transition is honoured as soon as

this is compatible with the simultaneity restriction 1 and the safety condition, it is well-known that the execution of an acknowledged transaction may be postponed undefinitely long. If we have

$M[1] = \{a\}$ $\quad F[1] = \{c\}$
$M[2] = \{b\}$ $\quad F[2] = \{c\}$
$M[3] = \emptyset$ $\quad F[3] = \{a, b, c\}$

and suppose that $T[3]$ would like to transfer record c to $M[3]$, then it cannot do os because otherwise the deadlock danger woul be introduced with respect to both $T[1]$ and $T[2]$. In the case of an infinite supply of transactions of type 1 and type 2, $T[3]$ could be kept asleep forever. This phenomenon is called individual starvation and as a rule it is considered to be undesirable.

A crude way to exorcize the danger of individual starvation is the following: as soon as a transaction is put to sleep, a fixed upper limit is imposed upon the number of transaction that may be initiated during that nap. We are not going to look for a more refined technique now, for there are other reasons why we consider our first model as too crude, and in our second model we shall depart from it.

The second model.

Our main compplaint about the first model is that a record once in set $M[i]$ remains in set $M[i]$ until the transaction has run to completion. We would like to be able to express that a transaction is such that a manipulated record is no longer essential for the correct progress of the transaction. We therefore split $M[i]$ into two disjoint sets $A[i]$ and $P[i]$, i.e. the records that are still active and the records that have been processed. A record in set $P[i]$ has arrived there from set $F[i]$ via set $A[i]$ and will remain there until termination of $T[i]$.

Obviously

$$i \neq j \quad \Rightarrow \quad A[i] \cap A[j] = \emptyset$$

ia a necessary condition, but this is no longer sufficient to guarantee that the net reaction of the parallel system is identical to the reaction of a sequential system after proper ordering of the requests, for it would not exclude

$$A[i] \cap P[j] \neq \emptyset \quad \underline{\text{and}} \quad A[j] \cap P[i] \neq \emptyset \ .$$

The first condition expresses that in the sequential ordering T[i] should follow T[j] and the second condition requires it to be the other way round. The situation is even worse, because if $P[i] \cap P[j] \neq \emptyset$, apparently, the order in which the shared record has been processed has been decided in the past, and this order is no longer expressed in the population of the various sets, but in general it is still relevant.

In our second model, the virtual order for the pair T[i], T[j] is irrevocably decided as far as their interference with the data base is concerned, as soon as for the first time holds

$$A[i] \cap P[j] \neq \emptyset \text{ or } A[j] \cap P[i] \neq \emptyset \quad .$$

Therefore we associate with each pair an antisymmetric function $V(i,j) = -V(j,i)$; when the pair is created -i.e. when the second transaction starts to be under execution - $V(i,j)$ is initialized with the value 0. During its life time it may remain constant, it may change its value once to either +1 or -1, where

$$V(i,j) = +1$$

means that in the virtual order T[i] has to precede T[j].

We now have the following invariant relations

$$i \neq j \Rightarrow A[i] \cap A[j] = \emptyset \qquad 3)$$

$$A[j] \cap P[i] \neq \emptyset \Rightarrow V(i,j) = +1 \qquad 4)$$

$$V(i,j) = +1 \Rightarrow A[i] \cap P[j] = \emptyset \qquad 5)$$

$$P[i] \cap P[j] \neq \emptyset \Rightarrow V(i,j) \neq 0 \qquad 6)$$

and deadlock is prevented, provided that we can renumber the transactions currently under execution is such a way that

$$i < j \Rightarrow \{F[i] \cap (A[j] \cup P[j]) = \emptyset \text{ and } V(i,j) \geq 0\} \qquad 7)$$

for then T[0] can be carried to completion without violation to the decided virtual order.

The second model shows great similarity to the first one. Again, the only point where it might be necessary to put a transaction to sleep is where it would like to transfer a record from set F to set A. The points of progress in one transaction that could result in the situation that sleeping transactions could be woken up are (as before) explicit F-decrease and termination, but in addition to those two the transition from A to P.

The problem of individual starvation can be dealt with in the same crude

fashion as in the first model and for the time being we shall leave it at that.

The third model.

The second model is appropriate when each transaction modifies all its active records. But that seems a rather exceptional situation and in our third model we would like to exploit that simultaneous inspection of a current record value by a number of parallel transactions is an absolutely innocent operation. For that reason we split all sets into two: F in FR and FW, A into AR and AW and P into PR and PW. Here AR are the "read only records", while a records in set AW may also be modified. Initially the transaction starts with FW equal to the universe and the other five sets empty. Permissible transfers of a record are: from FW to FR and AW, from FR to AR, from AW to PW and from AR to PR.

Now formulae 3) through 7) can be modified systematically by changing

$$X[i] \cap Y[j]$$

into

$$(XW[i] \cap YW[j]) \cup (XW[i] \cap YR[j]) \cup (XR[i] \cap YW[j])$$

i.e. from the four cross-products only the three in which writing is possibly involved, but not the fourth, the RR combination.

After this sytematic change we have formulae 3') through 7'), describing a model in which records shared for inspection only do not impose any mutual exclusion or virtual ordering. The only difference between the third and the second model is that in one transaction the transfer of a record from FW to FR could have the side-effect of waking up a sleeping transaction.

Note. If a transaction upon inspection of a record in set AW (because it might have to modify it) discovers that it can leave the record unchanged, we can, if we so desire, admit the transfer of this record from set AW to set AR. In that case also this transition could have the side-effect that another sleeping transaction can now be woken up.

Avoiding the danger of individual starvation.

In view of the formal relationship between the second and the third model it suffices to discuss the starvation problem in terms of the simpler formalism of the second model.

By the time, however, that we are going to tackle the starvation problem seriously, we should bear in mind that up till now we have assumed that the only reason for preventing progress of a transaction would be that otherwise relations 3 or 7 would be violated. In a general system one must assume that there will be other reasons as well: by the time that we bring into the picture that most of the records will be in secondary store most of the time, reduction of the traffic density between primary and secondary store might become a worthy goal and we can envisage a system trying to collect transactions involving the same records. The system can try to do so by postponing transactions, but also that strategic postponement must be void of the starvation danger.

With each transaction T[i] currently under execution we can associate an so-called "allowance counter" ac[i] and its value will be equal to the maximum number of other transactions allowed to run to completion before T[i] will run to completion. This implies that upon termination of a transaction all ac's associated with the remaining transactions will be decreased by 1. We now superimpose upon our original safety condition that the transaction can be renumbered in such a fashion that besides relation 7 also

$$i \leq ac[i] \qquad 8)$$

holds.

In that case T[0] can run to completion; its termination will decrease the remaining ac's by 1; simultaneously the remaining transactions will shift down over one place (i.e. the old T[1] becomes the new T[0]) and as a result relation 8 will continue to hold.

Inside a transaction we have now three types of points where the system may decide to put a transaction to sleep:

request for record transfer from F to A

request for potential strategic postponement

request to terminate.

Whenever a transaction makes such a request that can be honoured without violating conditions 3, 7 and 8, the system is in general free to refuse the request and to put the transaction to sleep. That would admit the possibility of a completely sleeping system and no real time guarantee could be given, even if a maximum execution time for a transaction is known. We therefore impose the requirement that

when the set of current transactions is non-empty, at least one

transaction must be non-sleeping.

When a transaction is initiated and its ac is introduced its initial value must be sufficiently high to guarantee 8; the number of transactions currently under execution will certainly be sufficient. The higher the initial value of the ac's, the greater the systems freedom in shuffling with the transaction order, but the weaker any real time guarantee about possible delays.

Finally in the above parallel system the order in which the transactions are terminated is a possible order for the transaction stream processed by the purely sequential system that should show the same net reaction.