TRANSACTION CHOPPING

Dennis Shasha, New York University, USA

SYNONYMS

transactions; ACID transactions

DEFINITION

Transaction chopping is a technique for improving the concurrent performance of a database system by reducing the time locks are held. The idea is to break up transactions into smaller "pieces", such that each piece executes as a transaction, but the effect is as if the original transactions executed serializably.

MAIN TEXT

Imagine an application that locks the entire database and then accesses a few rows in a table that does not fit into memory. If you looked at the resource statistics, you would find low CPU consumption and low disk utilization, yet the throughput would be very bad. For example, if 10 pages were accessed even at 1 millisecond per page, then throughput would be only 100 transactions per second. The point is that it is possible to slow down performance greatly just because of a poor locking strategy, even if there are plenty of resources.
Let's consider a less extreme case: a transaction that processes an order. The transaction might check whether there is sufficient cash available, then add the appropriate quantities to inventory, subtract the value from cash, and commit the transaction. Because any application that invokes this transaction will access the "cash" data item, that data item may become a bottleneck. Transaction chopping is a technique for circumventing such bottlenecks by dividing transactions into smaller transactional pieces that will hold locks for only a short time, yet still preserve the serializability of the original transactions.

ASSUMPTIONS Transaction chopping makes the following main assumptions: (i) You can characterize all the transactions that will run in some time interval. The characterization may be parameterized. For example, you may know that some transactions update account balances and branch balances, whereas others check account balances. However, you need not know exactly which accounts or branches will be updated. (ii) Your goal is to achieve the guarantees of serializability, while obtaining as much concurrency as possible. That is, you would like either to use degree 2 isolation snapshot isolation, or to chop your transactions into smaller pieces. The guarantee should be that the resulting execution be equivalent to one in which each original transaction executes serializably. (iii) If a transaction makes one or more calls to rollback, you know when these occur or you can arrange the transaction to move them towards the beginning.
A *chopping* partitions each T_i into *pieces* $c_{i_1}, c_{i_2}, \ldots c_{i_k}$. Every database access performed by T_i is in exactly one piece. A chopping of a transaction T is said to be *rollback-safe* if either T has no rollback statements or all the rollback statements of T are in its first piece. The first piece must have the property that all its statements execute before any other statements of T. This will prevent a transaction from half-committing and then rolling back. We want all transactions to be *rollback-safe*. Each piece will act like a transaction in the sense that each piece will acquire locks according to some standard method that guarantees the serializability of the piece.
For example, suppose T updates an account balance and then updates a branch balance. Each update might become a separate piece, acting as a separate transaction. As a second example, the transaction T operates at degree 2 isolation in which read locks are released as soon as reads complete. In this case, each read by itself constitutes a piece. All writes together form a piece (because the locks for the writes are only released when T completes).

CORRECT CHOPPINGS We will characterize the correctness of a chopping with the aid of an undirected graph having two kinds of edges. (i) *C edges*: C stands for *conflict*. Two pieces p and p' from different original transactions conflict if there is some data item x that both access and at least one modifies. We assume this simple read-write model for the purposes of exposition. In this case, draw an edge between p and p' and label the

edge C. (ii) *S edges*: S stands for *sibling*. Two pieces p and $p^{'}$ are siblings if they come from the same transaction T. In this case, draw an edge between p and $p^{'}$ and label the edge S.
We call the resulting graph the *chopping graph*. (Note that no edge can have both an S and a C label.) We say that a chopping graph has an *SC-cycle* if it contains a simple cycle that includes at least one S edge and at least one C edge. We say that a chopping of $T_1, T_2, \ldots, T_n$ is *correct* if any execution of the chopping is equivalent to some serial execution of the original transactions. "Equivalent" is in the sense of the serializability entry *Gottfried, please confirm.*

Theorem 1: A chopping is correct if it is rollback-safe and its chopping graph contains no SC-cycle.

Theorem 1 shows that the goal of any chopping of a set of transactions should be to obtain a rollback-safe chopping without an SC-cycle.

Chopping graph example 1

Suppose there are three transactions that can abstractly be characterized as follows:

```
T1: R(x) W(x) R(y) W(y)
T2: R(x) W(x)
T3: R(y) W(y)
```

Breaking up `T1` into `T11: R(x) W(x)` and `T12: R(y) W(y)` will result in a graph without an SC-cycle (Figure 1).

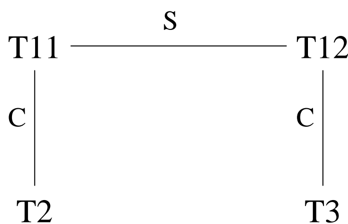

Figure 1: No SC-cycle

Chopping graph example 2

With the same `T2` and `T3` as in the first example, breaking up `T11` further into `T111: R(x)` and `T112: W(x)` will result in an SC-cycle (Figure 2).

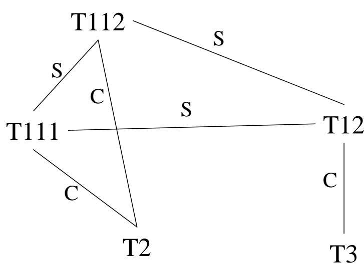

Figure 2: SC-cycle

Chopping graph example 3

By contrast, if the three transactions were

```
T1: R(x) W(x) R(y) W(y)
T2: R(x)
T3: R(y) W(y)
```

Then `T1` could be broken up into `T111: R(x)`, `T112: W(x)` and `T12: R(y) W(y)`. There is no SC-cycle (Figure 3). There is an S-cycle, but that doesn't matter.

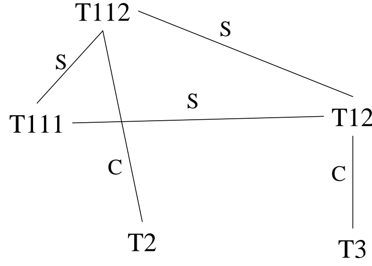

Figure 3: No SC-cycle

Chopping graph example 4

Now, let us take an example from a financial application. There are three types of transactions: (i) A transaction that updates a single depositor's account and the depositor's corresponding branch balance. (ii) A transaction that reads a depositor's account balance. (iii) A transaction that compares the sum of the depositors' account balances with the sum of the branch balances. For the sake of concreteness, suppose that depositor accounts D11, D12, and D13 all belong to branch B1; depositor accounts D21 and D22 both belong to B2. Here are the transactions:

```
 T1 (update depositor): RW(D11) RW(B1)
 T2 (update depositor): RW(D13) RW(B1)
 T3 (update depositor): RW(D21) RW(B2)
 T4 (read depositor): R(D12)
 T5 (read depositor): R(D21)
 T6 (compare balances): R(D11) R(D12) R(D13) R(B1) R(D21) R(D22) R(B2)
```

Thus, `T6` is the balance comparison transaction. Let us see first whether `T6` can be broken up into two transactions.

```
 T61: R(D11) R(D12) R(D13) R(B1)
 T62: R(D21) R(D22) R(B2)
```

The lack of an SC-cycle shows that this is possible.

FINDING THE FINEST CHOPPING Now, one might wonder whether there is an algorithm to obtain a correct chopping. Two questions are especially worrisome: (i) Can chopping a piece into smaller pieces break an SC-cycle? (ii) Can chopping one transaction prevent one from chopping another? Remarkably, the answer to both questions is "No."

Lemma 1: If a chopping is not correct, then any further chopping of any of the transactions will not render it correct.

Lemma 2: If two pieces of transaction T are in an SC-cycle as the result of some chopping, then they will be in a cycle even if no other transactions are chopped.

These two lemmas lead directly to a systematic method for chopping transactions as finely as possible. Consider again the set of transactions that can run in this interval $\{T_1, T_2, \ldots, T_n\}$. We will take each transaction T_i in turn. We call $\{c_1, c_2, \ldots, c_k\}$ a *private chopping* of T_i, denoted private(T_i), if both of the following hold: (i) $\{c_1, c_2, \ldots, c_k\}$ is a rollback-safe chopping of T_i. (ii) There is no SC-cycle in the graph whose nodes are $\{T_1, \ldots, T_{i-1}, c_1, c_2, \ldots, c_k, T_{i+1}, \ldots, T_n\}$. That is, the graph of all other transactions plus the chopping of T_i.

Theorem 2: The chopping consisting of $\{private(T_1), private(T_2) \ldots, private(T_n)\}$ is rollback-safe and has no SC-cycles.

Theorem 2 implies that the following algorithm yields the finest chopping:

```
procedure chop(T_1 , ... , T_n)
 for each T_i
  Fine_i := finest chopping of T_i with respect to
    all other unchopped transactions
  end for;
```

```
  the finest chopping for all transactions is
     { Fine_1 , Fine_2 , ... , Fine_n  }
```

Conclusion Transaction chopping is a method to enhance concurrency by shortening the time a bottleneck resource is held locked. We have seen it work well in practice and the theory extends naturally to snapshot isolation as well as read commited isolation levels.

CROSS REFERENCE

concurrency control; transaction execution; locking

REFERENCES

Database Tuning : Principles Experiments and Troubleshooting Techniques Dennis Shasha and Philippe Bonnet, Morgan Kaufmann Publishers, June 2002, ISBN 1-55860-753-6, Paper, 464 Pages.

"Making Snapshot Isolation Serializable" Alan Fekete, Dimitrios Liarokapis, Elizabeth O'Neil, Patrick O'Neil, Dennis Shasha ACM TODS, June 2005 vol. 30, number 2. pp. 492-528