CUCS-455-89

An Optimization to the Two-Phase Commitment Protocol

Dan Duchamp
Carnegie-Mellon University[1]

The basic two-phase distributed commitment protocol as described in [3, pp. 381-382] can be optimized so that a subordinate update site drops its locks more promptly and makes one fewer log force per transaction. The optimization applies as well to the variations of two-phase commitment (i.e., hierarchical, presumed commit, and presumed abort) described in the same paper.

As described in [3], the sequence of actions during the second phase of an execution of two-phase commitment in which the transaction commits is:

1. Coordinator forces commit record into its log.
2. Coordinator drops its locks.
3. Coordinator sends commit notice to subordinate.
4. Subordinate *forces* commit record into its log.
5. Subordinate drops its locks.
6. Subordinate sends commit acknowledgement to coordinator.
7. Subordinate forgets about transaction.
8. Upon receipt of all commit acknowledgements, coordinator forgets about transaction.

An optimization is possible wherein events 4 through 6 become:

4. Subordinate drops its locks.
5. Subordinate *spools* commit record into its log.
6. Subordinate sends commit acknowledgement to coordinator once the spooled commit record has been placed in the log.

The subordinate drops its locks *before* writing a commit record.

Of course, a subordinate may not drop its locks until the transaction is committed. In the unoptimized protocol, a subordinate writes its own commit record to indicate that the transaction is committed and therefore that locks may be dropped. The optimized protocol uses the commit record *at the coordinator* to indicate the same fact. So the coordinator must not forget about the transaction before the subordinate writes its own commit record; hence, the commit acknowledgement cannot be sent until the subordinate's commit record is written.

---

[1]Author's current address: Computer Science Department, Columbia University, New York, NY 10027.

This work was supported by IBM and the Defense Advanced Research Projects Agency, ARPA Order No. 4976, monitored by the Air Force Avionics Laboratory under Contract F33615-84-K-1520. The views and conclusions contained in this document are those of the author and should not be interpreted as representing the official policies, either expressed or implied, of any of the sponsoring agencies or of the United States Government.

The advantages of the optimization are:

1. Throughput at the subordinate is improved because fewer log forces are required. The amount of improvement is dependent upon the fraction of transactions that require distributed commitment.
2. Locks are retained at the subordinate for a slightly shorter time; this factor is important only if the transaction is very short.

Throughput is improved at no cost to latency. This optimization is independent from the notion of group commit [1, p. 7], which is a more widely applicable technique that improves throughput at the cost of increasing latency.

The disadvantages of the optimization are:

1. Some mechanism must exist to delay the sending of the commit acknowledgement notice until after the subordinate has written its commit record. (A system that implements group commit very likely already has this mechanism.)
2. The subordinate's "window of vulnerability" is increased.[2]

Both disadvantages are relatively minor, and are worth suffering considering that the optimization can result in noticeable throughput improvement, as reported in [2].

An important observation is that, even using the optimization, the serialization order of two transactions remains the same even if the second obtains the locks dropped "early" at a subordinate by a prior transaction whose commit record is not yet logged. That is, the contents of the subordinate log are the same whether or not the commit record of the first transaction is forced before its locks were dropped. This is true provided that the first log record written by the second transaction is either in the same commit group as the commit record of the first (for systems implementing group commit) or is always one that is forced, not spooled (for systems not implementing group commit). The simple case analysis below shows that the serialization order is preserved in systems not having group commit; the argument for systems with group commit is trivial.

Suppose that transaction 1 has committed and dropped its locks at a subordinate site X, but that its commit record has not yet been placed in the subordinate log. Suppose further that transaction 2 then locks some of the data that was updated by transaction 1. The possible next actions and their consequences are:

- Site X crashes: the recovery process aborts transaction 2 and re-prepares transaction 1. Since the commit acknowledgement for transaction 1 has not yet been sent, the subordinate queries the coordinator and discovers that transaction 1 is committed.
- Transaction 2 aborts: same as if transaction 2 never executed.
- Transaction 2 commits:

[2]The window is the time between the moment when the subordinate writes its prepare record and the moment it writes its commit/abort record. If the subordinate crashes during this window its recovery is dependent upon the availability of the coordinator.

- Transaction 2 is read-only: same as if transaction 2 never executed.
- Transaction 2 is not distributed: the commit record of transaction 2 is forced, thereby placing the commit record of the first transaction in the log before it.
- Transaction 2 is distributed, and Site X is the coordinator: the commit record of the second transaction is forced.
- Transaction 2 is distributed, and Site X is a subordinate: the prepare record of the second transaction is forced.

References

[1] D. J. DeWitt, et. al.
Implementation Techniques for Main Memory Databases.
In *Proc. ACM-SIGMOD 1984 Intl. Conf. om Mgmt. of Data*, pages 1-8. May, 1984.

[2] D. Duchamp.
Transaction Management.
PhD thesis, Carnegie-Mellon University, 1988.
Available as Technical Report CMU-CS-88-192, Carnegie-Mellon University.

[3] C. Mohan, B. Lindsay, R. Obermarck.
Transaction Management in the R* Distributed Database Management System.
ACM Trans. on Database Systems 11(4):378-396, December, 1986.