Concurrency Control Performance Modeling: Alternatives and Implications

Agrawal, Carey, Livny (1987)

What kind of paper is this?
• Unifying theme.
• Analytic modeling.
• Bring order to chaos; answer the questions, why do all the papers seem so contradictory.

The dilemma
• Lots of studies comparing concurrency control algorithms.
• Every study yields a different answer.
• Need a common, general framework that lets us clearly see the assumptions and explain why the results are so contradictory.
• Some problems
  • Infinite resources: no contention for disk, CPU, etc.
  • Fake restart: aborted transactions lead to new transactions instead of a replay of the aborting transaction.
  • Write locks: assume that we know a priori which items to write lock
The model

- Transaction is a sequence of reads/writes to objects.
- Operations conflict if they belong to different transactions and one is a write.

- Algorithm 1: Locking w/blocking
  - Always acquire read lock first.
  - Upgrade read to write when necessary.
  - If lock is unavailable, wait.
  - In case of deadlock, abort youngest.
  - Perform detection on every wait.

- Algorithm 2: Immediate restart
  - Same as above, but instead of waiting, abort immediately.
  - Retry same transaction after a delay.
  - Delay is on the order of response time.

- Algorithm 3: Optimistic
  - Validate unless transaction finds that something it read has been written in the interim.
  - If validation fails, retry.
  - Note that no delay is necessary, because the conflicting transaction has already committed.

Performance model

- Closed queueing model.
  - Fixed number of active transactions (multiprogramming level).
  - Transactions move between concurrency control requests (queues) and object requests (queues).
  - Performs all reads before writes.
  - Restarted transactions issue the same operations over.
  - Updates are deferred until transaction commit.
  - CPU and I/O time are provided as model parameters.
• Ignore modeling concurrency control time.
• Model transaction size (number of operations), write density, inter-
  transaction think time, internal think time, database size.

Experimental information
• Metric: transactions/second.
• Measure disk utilization (both actual and useful, which is utilization
  for committed transactions).
• Repeated runs so that deviation is within a few percent of the mean.
• Consider blocking ratio and restart ratio.
• Balance disks and CPU (1 CPU to 2 disks).
• Vary multiprogramming from 5 to 200.
• If probability of conflict was low (i.e., database was sufficiently large),
  then all the algorithms performed comparably.

Results
• Infinite Resources
  • Optimistic wins
  • Blocking beats immediate restart.
  • Conflict give rise to a loss of performance with
    locking.
  • Immediate restart gets into a negative feedback loop where as contention increases, so does
    response time, and therefore restart delay does as well.
  • Immediate restart also sees the highest variance due to the restart delay.
• Resource limited
  • All peak at low multi programming and then degrade.
  • Blocking does better than the other two.
  • More and more expensive restarts using optimistic.
  • Blocking consistently yields best response time.

• Multiple resources
  • At low multiprogramming, blocking wins.
  • At high multi programming, both immediate restart and optimistic are significantly better.
  • At very high mp, optimistic dominates.
  • Optimistic and immediate restart have higher disk utilizations.
  • For very large systems (25 resource units), optimistic beats blocking (by a little).
  • For large systems, there are resources to burn and the restarts impose less penalty.

• Interactive workload
  • Impose a delay (think time) before issuing writes.
  • Large think times (over 1 second) reduce contention (since most of the transaction is think time), so optimistic methods work best.

• Conclusions
  • At high contention (i.e., either low think time or few resource units), blocking wins.
  • At low contention, restart-oriented protocols win, and in particular, optimistic dominates locking with restart.
  • Multiprogramming level should be tuned to avoid thrashing of all the algorithms.
Results for common assumptions.
  • Restarts: do fake restarts (after an abort, issue a new transaction).
    • Fake restarts improve optimistic and immediate restart significantly.
    • More sensitive because they restart transactions more often.
  • Write lock acquisition
    • No effect at low MP, because there are few conflicts.
    • With infinite resources, write locks are acquired sooner and held longer, so throughput is worse.
    • The fact that there are no deadlocks doesn’t help much since resources are plentiful.
    • With limited resources, the no-upgrades produces better results, because fewer deadlocks and earlier restarts.
  • Conclusions
    • Assumptions have tremendous impact.
    • Relative performance isn’t that different for fake-restart, but absolute performance is.
    • No-upgrades biases in favor of restart algorithms.

Overall conclusions
  • Assumptions matter!
  • When resources are limited, blocking is the way to go.
  • If resources are plentiful, optimistic wins.