Today’s Plan

- Written exam (if there will be one...): July 14, 6pm
- (Last week’s homework)
- Questions, What do you want to practice?
- Review: CSR, OCSR, COCSR
- Commit serializability, Interleaving specifications
- Schedulers, 2PL, Deadlocks in 2PL
- Homework
Is the schedule in CSR, OCSR, COCSR? If yes, list all equivalent serial schedules.

- $r_2(x) \ r_1(x) \ c_1 \ r_3(y) \ w_3(z) \ c_3 \ w_2(x) \ c_2 \ r_4(x) \ r_4(y) \ r_4(z) \ c_4$
- $r_1(z) \ w_1(y) \ w_2(z) \ c_2 \ r_4(z) \ c_4 \ w_3(x) \ c_3 \ r_1(x) \ c_1$

For the fast students: Find a schedule that is in OCSR, but not in COCSR.
Commit serializability

- Avoids dirty reads
- Schedules should be prefix-commit-closed in environments with failures (e.g., CSR)
- CMFSR, CMVSR, CMCSR: every prefix of committed transactions is serializable
Interleaving specifications

- Less restrictive than conflict-based serializability
- Attempts to include semantic information
- Breakpoints: define indivisible units (IUs) within transactions
- “depends-on” relation takes conflicts and IUs into account
- “relatively serial schedule” and “relative serializability”
- Push forward and pull backward operations
Schedulers

- Filter that only fulfilling schedules can pass
- $Gen(S)$: all schedules possibly produced by $S$
- Criteria
  - safety: scheduler only produces schedules in a given class
  - scheduling power: can the scheduler produce all schedules in the given class?
Schedulers (2)

- Actions upon receiving a \(r, w, a, c\)
  - output directly
  - reject (aborting the transaction)
  - delay (postpone the operation)

- Optimistic scheduler: aggressive, executes most operations directly but might get stuck later

- Pessimistic scheduler: conservative, delays many operations, might lead to a \textit{serial} schedule, e.g. locking schedulers
Locking schedulers

- Most prominent type, outperforms other types in most cases
- Easy to implement, little runtime overhead
- Can be generalized to various transaction-related settings
- Lock requests, conflicts, waits, and releases
- Lock modes: shared, exclusive
- Well-formed locking rules: locking before access, unlocking after access, do not lock same item twice, no redundant unlocking, follow the lock compatibility rules
- $DT(s)$: schedule without locking and unlocking steps
Show the output produced by 2PL, S2PL, SS2PL.

\[ s = r_1(x) \ r_3(y) \ w_3(y) \ r_2(z) \ w_2(x) \ r_4(y) \ c_3 \ w_4(z) \ c_4 \ c_2 \ c_1 \]
2PL

- Credits: Dr. Andrey Gubichev, 2013
2PL: handling deadlocks

- 2PL can lead to deadlocks
- Wait-for graph: nodes are active transactions, edges: $T_i$ waits for $T_j$.
- Cycle in the WFG means deadlock
- $s = r_1(x)r_2(x)w_3(x)w_4(x)w_1(x)c_1w_2(x)c_2c_3c_4$
  - $T_1$ waits for $T_2$ and vice versa
Credits: Dr. Andrey Gubichev, 2013
Do not confuse WFGs and conflict graphs!

Lock conversation can also cause deadlocks

Resolution via victim criteria (last blocked, random, youngest transaction, transaction with fewest locks, transaction with fewest resource consumption, transaction involved in most cycles, transaction that eliminates most edges, ...)

Prevention via restrictions

- wait-die
- wound-wait
- immediate restart of the transaction that requests the lock
- running priority: blocked transactions may not impede running transactions by having the latter ones wait
Homework

- Already uploaded to our website.