Today’s Plan

- Last week’s homework
- 2PL, Deadlocks
- O2PL, Altruistic Locking
- Homework
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
2PL

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

- Do not confuse WFGs and conflict graphs!
- Lock conversion 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
Bigger subset of CSR than 2PL
Theoretically superior to 2PL but more complex to implement (run-time overhead)
Lock mode “ordered sharing”: lock operations and data operations must be executed in the same order
Unlocking rule: while \( pl_i(x) \rightarrow ql_j(x) \) and \( t_i \) has not started releasing locks, \( T_j \) is on hold and may not release any locks
\( Gen(O2PL) = OCSR \)
Altruistic Locking

- Extension of 2PL for long-running transactions
- “Donate” operation $d_i(x)$ that does not interfere with the 2PL of locking and unlocking steps, does still require unlocking
- When $T_j$ uses a donation by $T_i$ it has to stay in the wake until $T_j$ starts unlocking, but $T_j$ can also choose to ignore the donation
- $\text{Gen}(AL) \supset \text{Gen}(2PL)$ and $\text{Gen}(AL) \subset \text{CSR}$
Homework

- Already uploaded to our website.