Performance of Locking

70

- Locking aims to resolve conflicts among transactions by:
  - Blocking
  - Aborting

  Performance penalty
- Blocked Xacts hold locks other Xacts may want
- Aborting Xact wastes work done thus far
- Deadlock: Xact is blocked indefinitely until one of the Xacts is aborted

70

Locking Performance

71

- Locking performance problems are *common!*
- The problem is too much blocking.
- The solution is to reduce the "locking load"
- Good heuristic – If more than 30% of transactions are blocked, then reduce the number of concurrent transactions

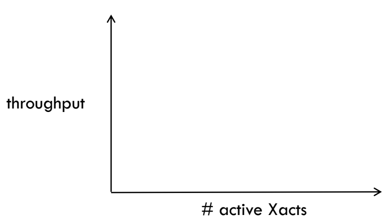
Credit: Phil Bernstein

71

Performance

72

- Locking overhead is primarily due to delays from blocking; minimizes throughout
- What happens to throughput as you increase the # of Xacts?



72

Improving Performance

73

- Lock the smallest sized object
  - Reduce likelihood that two Xacts need the same lock
- Reduce the time Xacts hold locks
- Reduce hot spots. A hot spot is an object that is frequently accessed, and causes blocking delays

73

Locking Granularity

74

- Granularity - size of data items to lock
  - e.g., files, pages, records, fields
- Coarse granularity implies
  - very few locks, so little locking overhead
  - must lock large chunks of data, so high chance of conflict, so concurrency may be low
- Fine granularity implies
  - many locks, so high locking overhead
  - locking conflict occurs only when two transactions try to access the exact same data concurrently

74

Deadlocks

75

- Deadlock: Cycle of transactions waiting for locks to be released by each other.
- Two ways of dealing with deadlocks:
  - Deadlock detection
  - Deadlock prevention

R. Ramakrishnan and J. Gehrke

75

Deadlocks

76

T1	T2
X(A)	
	X(B)
X(B)	
	X(A)

What happens?

Queued

Queued

T1 is waiting for T2 to release its lock
T2 is waiting for T1 to release its lock
→ Such a cycle of transactions is a *deadlock*

Implications:
•T1 and T2 will make no further progress
•They may hold locks needed by other Xacts
•DBMS tries to prevent or detect (and resolve) deadlocks

76

Deadlock Detection

77

- Detect deadlocks automatically, and abort a deadlocked transaction (the *victim*).
- Preferred approach, because it allows higher resource utilization
- Timeout-based deadlock detection - If a transaction is blocked for too long, then abort it.
  - Simple and easy to implement
  - But aborts unnecessarily (pessimistic) and
  - some deadlocks persist for too long

77

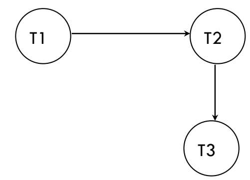
Deadlock Detection

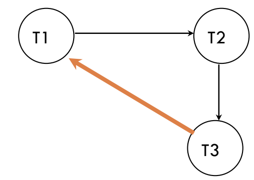
- Create a waits-for graph:
  - Nodes are transactions
  - There is an edge from Ti to Tj if Ti is waiting for Tj to release a lock
  - Lock mgr adds edge when lock request is queued
  - Remove edge when lock request granted
- A deadlock exists if there is a cycle in the waits-for graph
- Periodically check for cycles

78

Waits-For Graph: Example

T1: S(A), R(A), S(B)
T2: X(B),W(B) X(C)
T3: S(C), R(C)





79

Lecture Example

80

Selecting a Victim

- Deadlock is resolved by aborting a Xact in the cycle, and releasing its locks
- Different criteria may be used:

81

Commercial DBMS Approaches

82

- **MS SQL Server:** Aborts the transaction that is "cheapest" to roll back.
  - "Cheapest" is determined by the amount of log generated.
  - Allows transactions that you've invested a lot in, to complete.
- **Oracle:** The transaction that detects the deadlock is the victim.
- **DB2:** the deadlock detector arbitrarily selects one deadlocked process as the victim to roll back.

82

Deadlock Prevention

83

- When there is a high level of lock contention and an increased likelihood of deadlocks
- Prevent deadlocks by giving each Xact a priority
- Assign priorities based on timestamps.
  - Lower timestamp indicates higher priority
  - i.e., oldest transaction has the highest priority
  - Higher priority Xacts cannot wait for lower priority Xacts (or vice versa)

83

Deadlock Prevention (cont'd)

84

- Assume Ti wants a lock that Tj holds. Two policies are possible:
  - **Wait-Die:** If Ti has higher priority, Ti waits for Tj; otherwise Ti aborts
  - **Wound-wait:** If Ti has higher priority, Tj aborts; otherwise Ti waits
- Both schemes will cause aborts even though deadlock may not have occurred.
- If a transaction re-starts, make sure it has its original timestamp – WHY?

84

Wait-Die

85

- Suppose Ti tries to lock an item already locked by Tj.
- If Ti is the older transaction then Ti will wait
- Otherwise Ti is aborted and re-starts later with the same timestamp.
- Lower priority transactions never wait for higher priority transactions.

85

Example: Wait-Die

86

T1 starts before T2, so T1 is higher priority

T1	T2
S(A)	
R(A)	
	S(B)
	R(B)
X(B)	
	X(A)
	Abort
Resume	

T1 is older and so it is allowed to wait.

T2 is younger, and is aborted, which results in its locks being released. This allows T1 to proceed.

86

Wound-Wait

87

- Suppose Ti tries to lock an item locked by Tj.
- If Ti is higher priority (older transaction)
  - then Tj is aborted and restarts later with the same timestamp;
- Otherwise, Ti waits.

87

Example: Wound-Wait

88

T1 starts before T2, so T1 is higher priority

T1	T2
S(A)	
R(A)	
	S(B)
	R(B)
X(B)	
	Abort

T1 is older, so T2 is aborted, and that allows T1 to proceed.

88