Transactions
(Concurrency, Recovery)

Distributed Databases

CSC 321/621 – 4/3/2012

Transactions: Review

- Previously:
  - Defined transactions & required (ACID) properties of transactions
  - Discussed how concurrency needed to be managed when using transactions
- Today:
  - Rollbacks and recovery
  - Distributed databases (if have time)

Concurrency Practice Problems

- Let transactions T1-T3 be the following:
  - T1: Add 1 to A
  - T2: Double A
  - T3: Display A on the screen and then set A to 1

- If A has initial value 0, how many possible correct results are there?

What does "correct result" even mean?

Concurrency Practice Problems

- Correct results stem from serial schedules, of which there are 6
  - T1,T2,T3: Results in 1
  - T1,T3,T2: Results in 2
  - T2,T1,T3: Results in 1
  - T2,T3,T1: Results in 2
  - T3,T1,T2: Results in 4
  - T3,T2,T1: Results in 3

Concurrency Practice Problems

Assume the structure of T1-T3 is as follow:

- T1: (a) Retrieve A into t1 (R1)
  (b) t1 := t1 + 1
  (c) Update A from t1 (U1)
- T2: (a) Retrieve A into t2 (R2)
  (b) t2 := t2*2
  (c) Update A from t2 (U2)
- T3: (a) Retrieve A into t3 (R3)
  (b) Display t3
  (c) Update A from 1 (1 == a constant) (U3)

If the transactions execute WITHOUT locking, how many possible schedules are there over Rx, Ux?

Don’t care about (b) statements as those are localized

Concurrency Practice Problems

There are some ordering constraints here:

R1 < U1, R2 < U2, R3 < U3

so not 6 *5 * 4 * 3 * 2 * 1 (720)

There are effectively six meta-orderings (Ri,Rj,Rk cover R1,R2,R3, not necessarily in that order; same for Up,Uq,Ur)

Ri-Rj-Rk-Up-Uq-Ur

Ri-Rj-Up-Rk-Uq-Ur

Ri-Rj-Up-Uq-Rk-Ur

Ri-Up-Rj-Rk-Uq-Ur

Ri-Up-Rj-Uq-Rk-Ur

Concurrency Practice Problems

Why is this a 2?

Total options:

Ri-Rj-Rk-Up-Uq-Ur = 3 * 2 * 1 * 3 * 2 * 1 = 36

Ri-Rj-Up-Rk-Uq-Ur = 3 * 2 * $\boxed{2}$ * 1 * 2 * 1 = 24

Ri-Rj-Up-Uq-Rk-Ur = 3 * 2 * 2 * 1 * 1 * 1 = 12

Ri-Up-Rj-Rk-Uq-Ur = 3 * 1 * 2 * 1 * 2 * 1 = 12

Ri-Up-Rj-Uq-Rk-Ur = 3 * 1 * 2 * 1 * 1 * 1 = 6

90 schedules

Concurrency Practice Problems

- Is the schedule R1-R3-U1-U3-R2-U2 a serializable schedule? Assume A initially 0.

- Remember:
  - T1,T2,T3: Results in 1
  - T1,T3,T2: Results in 2
  - T2,T1,T3: Results in 1
  - T2,T3,T1: Results in 2
  - T3,T1,T2: Results in 4
  - T3,T2,T1: Results in 3

- T1: (a) Retrieve A into t1 (R1)
  (b) t1 := t1 + 1
  (c) Update A from t1 (U1)
- T2: (a) Retrieve A into t2 (R2)
  (b) t2 := t2*2
  (c) Update A from t2 (U2)
- T3: (a) Retrieve A into t3 (R3)
  (b) Display t3
  (c) Update A from 1 (1 == a constant) (U3)

Concurrency Practice Problems

- Is the schedule R1-R3-U1-U3-R2-U2 a serializable schedule? Assume A initially 0.

- Remember:
  - T1,T2,T3: Results in 1
  - T1,T3,T2: Results in 2
  - T2,T1,T3: Results in 1
  - T2,T3,T1: Results in 2
  - T3,T1,T2: Results in 4
  - T3,T2,T1: Results in 3

- T1: (a) Retrieve A into t1 (R1)
  (b) t1 := t1 + 1
  (c) Update A from t1 (U1)
- T2: (a) Retrieve A into t2 (R2)
  (b) t2 := t2*2
  (c) Update A from t2 (U2)
- T3: (a) Retrieve A into t3 (R3)
  (b) Display t3
  (c) Update A from 1 (1 == a constant) (U3)

R1: Retrieve A into t1 (t1 = 0)
R3: Retrieve A into t3 (t3 = 0)
U1: Update A from t1 (A = 1)
U3: Update A from t3 (A = 1)
R2: Retrieve A into t2 (A = 1)
U2: Update A from t2 (A = 2)

Equivalent to T1, T3, T2 so yes serializable

Concurrency Practice Problems

- Can R1-R3-U1-U3-R2-U2 be produced from a 2-phase locking protocol?

- Remember:
  - T1,T2,T3: Results in 1
  - T1,T3,T2: Results in 2
  - T2,T1,T3: Results in 1
  - T2,T3,T1: Results in 2
  - T3,T1,T2: Results in 4
  - T3,T2,T1: Results in 3

- T1: (a) Retrieve A into t1 (R1)
  (b) t1 := t1 + 1
  (c) Update A from t1 (U1)
- T2: (a) Retrieve A into t2 (R2)
  (b) t2 := t2*2
  (c) Update A from t2 (U2)
- T3: (a) Retrieve A into t3 (R3)
  (b) Display t3
  (c) Update A from 1 (1 == a constant) (U3)

Concurrency Practice Problems

- Can R1-R3-U1-U3-R2-U2 be produced from a 2-phase locking protocol?

- Remember:
  - T1,T2,T3: Results in 1
  - T1,T3,T2: Results in 2
  - T2,T1,T3: Results in 1
  - T2,T3,T1: Results in 2
  - T3,T1,T2: Results in 4
  - T3,T2,T1: Results in 3

- T1: (a) Retrieve A into t1 (R1)
  (b) t1 := t1 + 1
  (c) Update A from t1 (U1)
- T2: (a) Retrieve A into t2 (R2)
  (b) t2 := t2*2
  (c) Update A from t2 (U2)
- T3: (a) Retrieve A into t3 (R3)
  (b) Display t3
  (c) Update A from 1 (1 == a constant) (U3)

No, assuming that transactions only request level of lock needed:

R1 requests Shared/Read Lock on A – granted
R3 request Shared/Read Lock on A – granted
R1 requests Exclusive/Write Lock on A – waits for R3 to release Shared/Read
R3 requests Exclusive/Write Lock on A – waits for R1 to release Shared/Read

Deadlock!
Break deadlock by aborting a transaction & requesting restart

Database Recovery In General

Data Storage Hierarchy

Type	Speed (Relative)	Online/Offline	Size	Volatility (Crash Survivability)
Main Memory (Primary Storage)	Fast	Online	Small	Volatile (Doesn’t survive)
Magnetic Disk (Secondary Storage)	Medium	Online	Mediu	Nonvolatile (Survives)
Optical Disk (Secondary Storage)	Slow (Random Access)	Offline	Large	Nonvolatile
Magnetic Tape (Secondary Storage)	Slow (Sequential Access)	Offline	Very Large	Nonvolatile

Stable storage: information replicated across multiple non-volatile media with independent failure modes (RAID arrays for example)

Database Recovery In General

- Need to be concerned with two primary types of failures:
  - Loss of main memory (system crashes)
  - Loss of secondary storage
- Also need to be able to handle failed/aborted transactions (an instance of needing to go back to last consistent state)

Relationship between Transactions and Recovery

- Transactions are the “unit” of recovery
  - Even implicit transactions: single-action *changes* which do not require denoting as transactions

- Recovery manager must ensure “all-or-nothing” view of transactions
  - Because of error, intention, or failure

Basic Disk Processes

- Deal with main memory (volatile) and hard disk (non-volatile) for now

- Managed through memory-based database buffers – temporary storage points --

- *Reads:*
  - Look up address of disk block containing tuple from table Y with primary key value X
  - Transfer the disk block into a database buffer in main memory
  - Copy the desired columnar data from the database buffer into memory for variable of interest

Basic Disk Processes

- *Writes:*
  - Look up address of disk block containing tuple from table Y with primary key value X
  - Transfer the disk block into a database buffer in main memory
  - Copy the data of interest from the memory for the variable of interest into the appropriate spot in the database buffer
  - Write the database buffer back to disk

Buffers

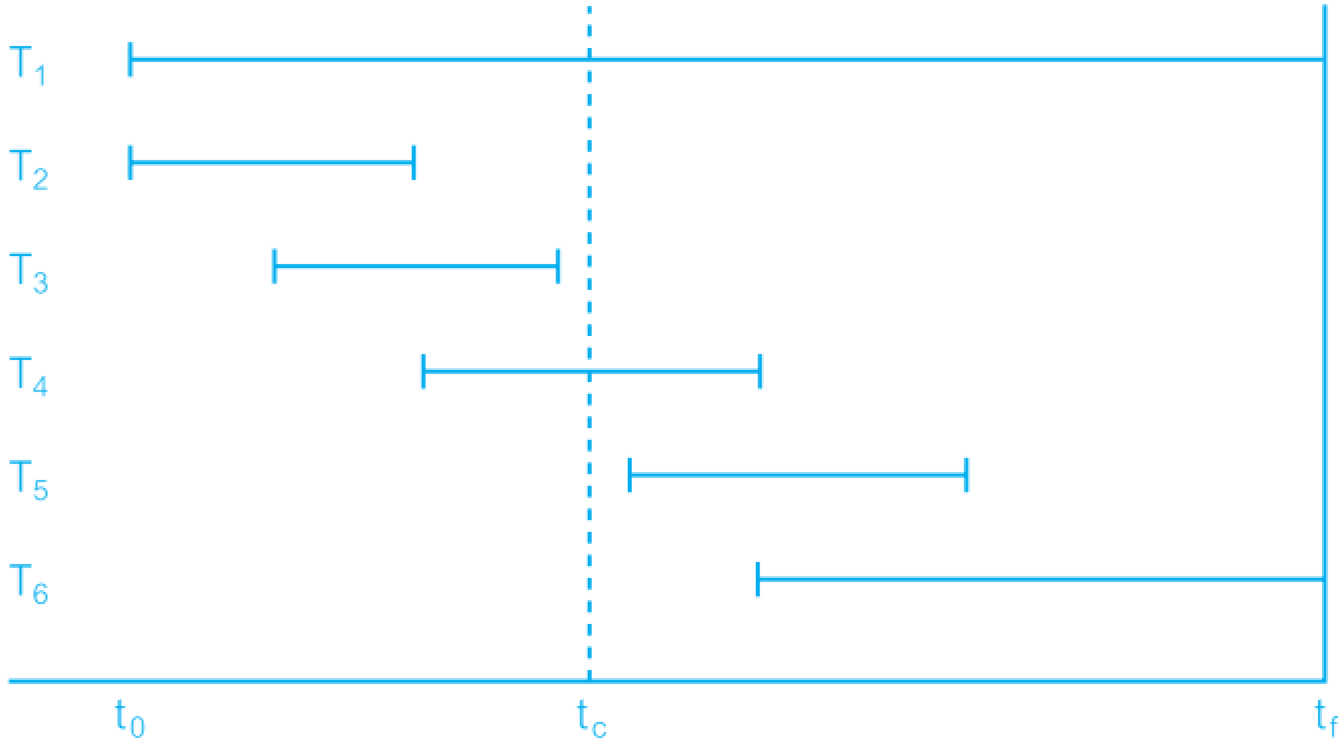
- Buffers – temporary storage points – are not typically written to disk immediately after an update for efficiency reasons
  - One large write faster than lots of small writes
  - “Flushing buffers”

- This opens up the possibility of failure between writing to the buffers and buffers writing to disk.

Buffers and Transactions

- If there is a primary storage (memory failure) before buffers are flushed, the following must occur with transactions:
  - Check if the transaction was already committed
  - If so, then redo the transactions updates to database (“roll-forward”)
  - If not, then undo the transaction’s updates to the database (“roll-back”)

Buffers and Transactions



tc = (checkpoint) write to disk of buffers, tf = failure, |- = start, -| = commit

Which transactions (T1..T6) need to be redone? Which transactions need to be undone?

Checkpoints

- A checkpoint is a point of synchronization between the database buffers and disk.
  - All modified blocks in database buffers are written to secondary storage
- Typically scheduled three to four times an hour.
- We also record the time we did the checkpoint (we'll come back to that in "logging").

Log File

- A journal is maintained containing:
  - Records of transaction progress
  - Records of checkpoints

What could we use for this?

- Transaction record:
  - Transaction identifier
  - Time of event
  - Type of record (trans start, trans abort, trans commit, insert, update, delete)
  - If an update/delete/insert, identifier of item affected
  - For updates/deletes, a “before-image” of the data item
  - For updates/inserts, an “after-image” of the data item
  - References to the next entry for the same transaction (to make undo/redo of a transaction fast)

Log File Example

Tid	Time	Operation	Object	Before image	After image	pPtr	nPtr
T1	10:12	START				0	2
T1	10:13	UPDATE	STAFF SL21	(old value)	(new value)	1	8
T2	10:14	START				0	4
T2	10:16	INSERT	STAFF SG37		(new value)	3	5
T2	10:17	DELETE	STAFF SA9	(old value)		4	6
T2	10:17	UPDATE	PROPERTY PG16	(old value)	(new value)	5	9
T3	10:18	START				0	11
T1	10:18	COMMIT				2	0
	10:19	CHECKPOINT	T2, T3				
T2	10:19	COMMIT				6	0
T3	10:20	INSERT	PROPERTY PG4		(new value)	7	12
T3	10:21	COMMIT				11	0

This section of the log file contains transaction records for T1, T2, T3 and one checkpoint record.

Log File: Costs

- A lot of logging occurs!
  - Leads to large files
  - Files often needed for fast recovery, so want online when possible.
- A useful approach:
  - Maintain two logs at a time
  - Log to one until it reaches a certain size
  - At that point, open a second log, start writing new transactions to the new log
  - Old transactions continue to use old log until done.
  - When old transactions all finish writing to old log, archive to offline storage

Log File: Storage

- Log files are typically stored on a different hard disk than the database proper

Why?

- Reduce likelihood of loss of both the database and log file

Back to Checkpoints

- In addition to a point of synchronization between database buffers and disk, checkpoints also affect log
  - All log records in main memory are saved to secondary storage (just like the database buffer data)
  - A checkpoint record is written in the log file, containing identifiers of active (still running) transactions

Log File Example

Tid	Time	Operation	Object	Before image	After image	pPtr	nPtr
T1	10:12	START				0	2
T1	10:13	UPDATE	STAFF SL21	(old value)	(new value)	1	8
T2	10:14	START				0	4
T2	10:16	INSERT	STAFF SG37		(new value)	3	5
T2	10:17	DELETE	STAFF SA9	(old value)		4	6
T2	10:17	UPDATE	PROPERTY PG16	(old value)	(new value)	5	9
T3	10:18	START				0	11
T1	10:18	COMMIT				2	0
	10:19	CHECKPOINT	T2, T3				
T2	10:19	COMMIT				6	0
T3	10:20	INSERT	PROPERTY PG4		(new value)	7	12
T3	10:21	COMMIT				11	0

T2 and T3 are still running at the checkpoint

Checkpoints & Logging for Recovery

- When a failure occurs,
  - Find most recent checkpoint
  - Redo all transactions committed since checkpoint passed
  - Undo all transactions active at failure

Actual Recovery Protocols: Immediate Update

- Immediate update:
  - Updates are sent to the database as they occur (our common way of thinking about things)
  - Transaction handling:
    - When transaction starts, write trans start record
    - When a write occurs, write the record to the log file
    - After the log file is written, write the update to the database buffers
    - When a transaction commits, write the trans commit record (or for aborts, a trans abort record)
    - Database buffers flushed when checkpoint encountered

Actual Recovery Protocols: Immediate Update

- Immediate update:
  - Updates are sent to the database as they occur (our common way of thinking about things)
  - Transaction handling:
    - …
    - When a write occurs, write the record to the log file
    - After the log file is written, write the update to the database buffers
    - …

Why do these happen in that order? Called "write-ahead log protocol"

Actual Recovery Protocols: Immediate Update

- Using immediate update,
  - It is possible that data has been written to secondary storage for transactions not committed
  - Has two effects:
    - If an abort occurs for the transaction, use the log and write the before images back to the database, in reverse order (most recent first)
    - If a crash occurs,
      - Redo any transactions that indicate a commit since last checkpoint by re-writing after-image data, in normal oldest to newest order, as recorded in log file
      - Undo any transactions without a commit since last checkpoint by going in reverse and writing before-image data, request that thost transactions restart

Actual Recovery Protocols: Deferred Update

- Deferred update:
  - Updates are not written to the database (even to the database buffers) until a transaction commit occurs
  - Transaction handling:
    - When transaction starts, write trans start record
    - When a write occurs, write the record to the log file
    - When a transaction commits,
      - Write the trans commit record (or for aborts, a trans abort record)
      - Write all log records for that transaction to disk
      - Use the log records to write the actual updates to the database buffers
    - Database buffer writes to disk are performed periodically at checkpoints

Deferred Updates

- Given the notion of deferred updates, think about the following statements which were true for immediate updates. *Do they still hold for deferred updates?*
  - It is possible that data has been written to secondary storage for transactions not committed
  - Has two effects:
    - If an abort occurs for the transaction, use the log and write the before images back to the database, in reverse order (most recent first)
    - If a crash occurs,
      - Redo any transactions that indicate a commit since last checkpoint by re-writing after-image data, in normal oldest to newest order, as recorded in log file
      - Undo any transactions without a commit since last checkpoint by going in reverse and writing before-image data, request that thost transactions restart