Centralised Txn Management Review Slides

Jyrki Nummenmaa
http://www.cs.uta.fi/dtm/
jyrki@cs.uta.fi

Purpose of these slides

- These slides contain concepts, that are

Transaction

- A txn consists of the execution of a sequence of client requests that access or update one or more of the data items.
- A txn may commit (complete successfully), or
- be rolled back to the beginning & re-started, or
- may be killed off without any of its requested changes becoming permanent.
- In a txn, individual modifications to the database are aggregated into a single large modification that appears to occur either entirely in a single moment or not at all.

Transaction Processing system

- Transaction processing systems (TP systems) provide tools to help software development for applications that involve querying and updating databases.
- The term ”TP system” is generally taken to mean a complete system, including application generators, one or more database systems, utilities and networking software.
- Within a TP system, there is a core collection of services, called the TP monitor, that coordinates the flow of txns through the system.

ACID properties

- A txn, T, is a collection of operations on the state of the system that has the following properties (known as the ACID properties):
- Atomicity: T' changes to the state are atomic: either all happen or none happen.
- Consistency: The actions of T , taken as a group, must not violate any of the integrity constraints associated with the state.
- continued on the next slide...

ACID properties (continued)

- Isolation: Even though txns execute concurrently with T, it appears to T that each other txn executed either before or after T.
- Durability: Once T completes successfully (commits), its changes to the state of the system survive failures.

Statistics & Checkpointing

- Gray and Reuter give the following figures for a ``typical'' txn system:
  - 96 percent of all txns complete successfully.
  - 3 percent of all txns ``commit suicide''.
  - 1 percent of all txns are killed by the system.
- To minimise the loss of work due to an abort, the DBMS may provide checkpointing - a way to commit changes in the midst of a txn without terminating the txn.

Why concurrency?

- Large database systems are typically multi-user systems; that is, they are systems that allow a large number of txns to access the data in a database at the same time.
- In principle, it is possible to at any given time allow only a single txn to execute, but this will not give satisfactory performance.
- The txn throughput will be too slow because a txn typically spends most of its lifetime waiting for input/output events to compete as it accesses items of data on disk.

Total Order Implementation

- By interleaving txns, we can get better utilization of the computer hardware.
- The price we pay is more complexity in managing the activity in the database management system.

Lost Update Problem

- Txn A retrieves record R at time t1.
- Txn B retrieves record R at time t2.
- Txn A updates its copy of R at time t3.
- Txn B updates its copy of R at time t4.

- Txn A's update is lost because txn B overwrites it.

Uncommitted Dependency Problem

- Txn A retrieves and updates record R at time t1.
- Txn B retrieves the version of record R, as updated by A, at time t2.
- Txn A is rolled back at time t3.

- Txn B saw data, which was never permanently recorded.

Inconsistent Analysis Problem

- Txn A is summing account balances.
- Txn B transfers a sum of money from one account to another whilst txn A is in the middle of computing the sum.
- Txn A may see an inconsistent state of the database and this led it to perform an inconsistent analysis.

Locking

- A locking mechanism can solve all of the above problems.
- When a txn requires some assurance that the contents of a database item will not change whilst the txn is performing its work, the txn acquires a lock on the record.
- This means that other txns are `locked out' of the record and, in particular, are prevented from changing the item.

Lock types

- Exclusive locks (X locks) and shared locks (S locks).
- If txn A holds an X lock on a record, R, then a request from another txn, B, for a lock on R will cause txn B to go into a wait state until A releases its lock.
- If txn A holds an S lock then txn B can also be granted an S lock, but B will enter a wait state if it requests an X lock.

Concurrency policy

- In general, user programs will often attempt to update the same pieces of information at the same time.
- Doing so creates a contention for the data.
- The concurrency control mechanism mediates these conflicts.
- It does so by instituting policies that dictate how read and write conflicts will be handled.

Conservative Policy

- The most conservative way to enforce serialisation is to make a txn lock all necessary objects at the start of the transaction and to release the locks when the txn terminates.
- However, by distinguishing between reading the data and acquiring it to modify (write) it, greater concurrency can be provided.
- We do this by choosing an appropriate lock to put on the data --- ”read only” or ”update”.
- This allows an object to have many concurrent readers but only one writer.

The actions of txn

- A program must start a txn before it accesses persistent data.
- While the txn is in progress, the program's actions can include reads and writes to persistent objects.
- The program can then either commit or abort the txn at any time.
- By committing a txn, changes made to persistent data during the txn are made permanent in the database and visible to other processes.

The actions of txn / 2

- Changes to persistent data are ``undone'' or ``rolled back'' if the txn in which they were made is aborted.
- So txns do two things:
  - they mark off program segments whose effects can be ``undone'', and
  - they mark off program segments that, from the point of view of other processes, execute either all at once or not at all – other processes don't see the intermediate results.

Recovery

- Once the txn has completed, the DBMS must ensure that either
  - (a) all the changes to the data are recorded permanently in the database, or
  - (b) the txn has no effect at all on the database or on any other txns.
- We must avoid the situation in which some of the changes are applied to the database while others are not.
- The database would not necessarily be left in a consistent state if only some of the txn's changes are made permanent.

Recovery / 2

- Problems might arise if there is some sort of failure during the lifetime of the txn.
- There are several types of possible failure:
  - A computer failure (due to a hardware or software error) during the execution of the txn.
  - A txn error. This could be, for example, because the user interrupted the execution with a control-C.
  - A condition, such as insufficient authorization, might cause the system to cancel the txn.
  - The system may abort the txn, e.g. to break a deadlock.
  - Physical problems. Disk failure, corrupted disk blocks, power failure, etc.

Recovery log

- In order to recover from txn failures, the system maintains a log, which keeps track of all txn operations affecting the database item values.
- The log is kept on disk, so it is not affected by any of the failures except disk failure.
- Periodically, the log is backed up to archive tape, in order to guard against failures.
- For each txn, the log will contain information about the fact that the txn started, the granules that it wrote and read and whether or not it completed successfully.

Recovery log / 2

- Some CC schemes require more extensive log information than others.
- It is considered to be advantageous when a CC scheme requires less log information.

Serializability

- For performance reasons, we allow executions of txns that have the same effect as serial executions, even though they may involve interleaving the execution of the operations in the txns.
- An execution is serializable if it produces the same output and has the same effect on the database as some serial execution of the same txns.
- Any serial execution is assumed to be correct and since a serializable execution has the same effect as one of the possible serial executions, serializable executions may be assumed to be correct, too.

Scheduler

- We assume that the DBMS has a scheduler, i.e. a program that controls the concurrent execution of txns.
- It restricts the order in which the Reads, Writes, Commits and Aborts of different txns are executed.
- It orders these operations so that the resulting schedule is serializable.

Scheduler / 2

- After receiving the details of an operation from the txn, the scheduler can take one of the following three actions:
- Execute: The scheduler will be informed when the operation has been executed.
- Reject: The scheduler may tell the txn that the operation has been rejected. This would cause the txn to be aborted.
- Delay: The scheduler can place the operation into a queue. Later, the scheduler can make a decision as to whether to execute it or reject it.

Schedule

- A schedule, say S, of a set of n txns, T1 , T2 , ... , Tn, is an ordering of the operations of the txns, subject to the constraint that, for each txn, say Ti,
- that participates in S, the ordering of the operations in Ti must be respected in S.
- Of course, operations from some other txn, say Tj, can be interleaved with the operations of Tj in S.

Conflicting operations

- Two operations in a schedule conflict, if
  - they belong to different txns,
  - they access the same data item, say x, and
  - one or both of the operations is a write.
- Let S1 and S2 be two schedules over the same set T1 , T2 ,... , T_n, of txns.
- We say that S1 and S2 are {\it conflict equivalent} if the order of any two {\it conflicting} operations is the same in both schedules.
- A schedule is {\it serializable} if it represents a serializable execution.

Conflicting operations / 2

- A schedule, S is conflict serializable if it is conflict equivalent to some serial schedule, S'.
- In this case, we could (in principle) re-order the non-conflicting operations in S so as to obtain the schedule $S’

Conflicting operations / 2

- Most CC methods do not explicitly test for serializability.
- Rather, the scheduler is designed to operate according to a protocol which guarantees that the schedule produced by the scheduler will be serializable.
- In general, checking for serializability is tricky.
- Txns are continuously starting, finishing and rolling back, and each txn is continuously submitting operations to be scheduled.

2PL and serializability

- Most commercial DBMS's CC facilities are based on the use of the strict two-phase locking protocol.
- When the txns adhered to 2PL, the resulting schedule is always serializable.

2PL

- A txn adheres to the two-phase locking (2PL) protocol if all locking operations are carried out before any of the unlocking operations.
- If a txn adheres to the 2PL protocol, we can divide its execution into two phases: (1) a growing phase, during which locks on granules are obtained and no lock is released, and (2) a shrinking phase during which existing locks can be released but no new locks can be acquired.
- Some DBMSs allow a read lock to be upgraded to an exclusive lock.
- Our definition of 2PL covers this case.

2PL

- The advantage of 2PL is that if every txn in a schedule follows the 2PL protocol, the schedule is guaranteed to be serializable.
- 2PL severely limits the amount of concurrency that can occur in a schedule. A long-running txn, T may not need to keep a lock on a granule, say X, even though T has finished reading or writing, because T may later need to lock some granule.
- Another problem is that T may need to lock a granule, say X a long time before it really needs to, merely so that it can release a lock on a `popular' granule, Y, so that other txns can access Y.

Strict schedules

- Strict 2PL guarantees so-called strict schedules i.e. schedules in which a txn, T1, can neither read nor write a granule,X, until all txns that have previously written X have committed or aborted.
- Strict schedules simplify recovery because you just have to restore the `before' image of X, i.e.the value that X had before the aborted write.
- In strict 2PL, a txn, T, does not release any of its locks until after it commits or aborts.
- Thus no other txn can read or write an granule that is written by T unless T has committed.
- Strict 2PL is not deadlock-free unless it is combined with conservative 2PL.