# Databases and Information Systems CS303

Recovery Management
06-10-2023

## Recap

- Stable Storage implementation
- Log Files
- Undo / Redo
- Checkpoint operation

## Recovery Algorithm

- We have identified which transactions need undo / redo
- Is there an efficient algorithm to do this?
- Assume that a data item that has been updated by an uncommitted transaction cannot be modified by any other transaction, until the first transaction has either committed or aborted.
    - Recoverable schedules

# Recovery Algorithm : Transaction Rollback

- The log is scanned backward, and for each log record of $T_i$ of the form $<T_i , X , V_1 , V_2 >$ that is found:
    - The value $V_1$ is written to data item X
    - A special undo-only log record $<T_i , X , V_1 >$ is written to the log,
        - $V_1$ is the value being restored to data item X during the rollback.

- Once the log record $<T_i$ start$>$ is found the backward scan is stopped, and a log record $<T_i$ abort$>$ is written to the log.

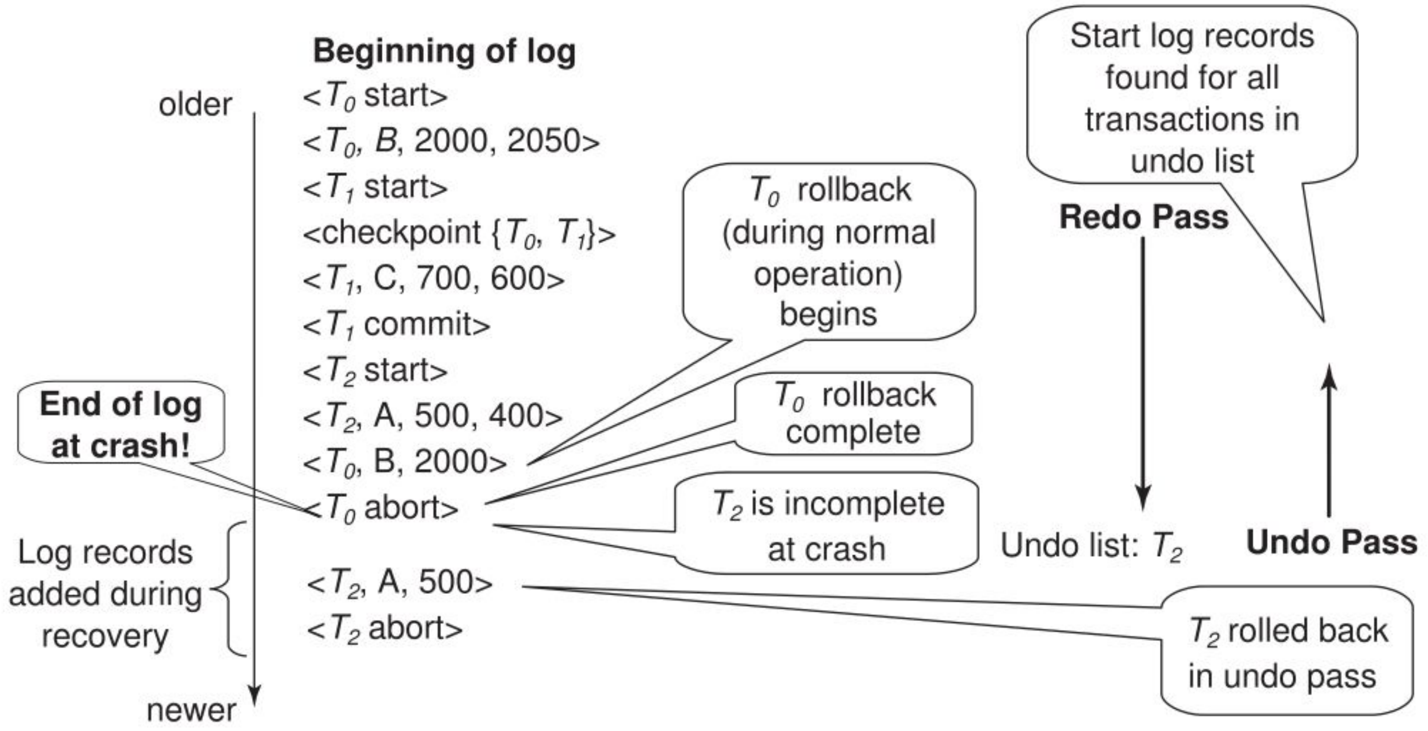
# Recovery Algorithm : After System Crash

- Has Two phases
  - Redo phase : Updates all transactions by scanning the log forward from the last checkpoint
    - The list of transactions to be rolled back, undo-list, is initially set to the list L in the <checkpoint L> log record.
    - Whenever a normal log record of the form $<T_i , X , V_1 , V_2 >$, or a redo-only log record of the form $<T_i , X , V_2 >$ is encountered, the operation is redone;
      - The value $V_2$ is written to data item $X$
    - Whenever a log record of the form $<T_i$ start> is found, $T_i$ is added to undo-list.
    - Whenever a log record of the form $<T_i$ abort> or $<T_i$ commit> is found, $T_i$ is removed from undo-list.
  - At the end of Redo phase, undo-list contains list of transactions that are incomplete.
    - They need undo

## Recovery Algorithm : After System Crash

- Has Two phases
    - Undo phase : the system rolls back all transactions in the undo-list.
    - It performs rollback by scanning the log backward from the end.
        - Whenever it finds a log record belonging to a transaction in the undo-list, it performs undo actions just as if the log record had been found during the rollback of a failed transaction.
        - When the system finds a <$T_i$ start> log record for a transaction $T_i$ in undo-list, it writes a <$T_i$ abort> log record to the log, and removes $T_i$ from undo-list.
    - The undo phase terminates once undo-list becomes empty,
        - Which means, the system has found <$T_i$ start> log records for all transactions that were initially in undo-list.

# Recovery Algorithm : After System Crash



## Buffer management

- Log-record buffering
  - We have assumed that log records are written to stable storage as soon as they are created
  - Not practical : Time consuming
  - Maintain log buffer in main memory (multiple log records can be output in a single operation)
  - But buffers do not survive system crash.

  - So impose additional requirements on recovery techniques to ensure atomicity:
    - Transaction T enters the commit state after the <T commit> log record has been output to stable storage.
    - Before the <T commit> log record can be output to stable storage, all log records pertaining to transaction T must have been output to stable storage.
    - Before a block of data in main memory can be output to the database (in nonvolatile storage), all log records pertaining to data in that block must have been output to stable storage.
      - Called Write-Ahead Logging (WAL)

## Buffer management

- Database buffering:
    - Force policy : Transactions would force-output all modified blocks to disk when they commit.
    - NoForce policy : Allows a transaction to commit even if it has modified some blocks that have not yet been written back to disk.

    - Recovery algorithms that we have discussed works in both cases

    - No-steal policy: Blocks modified by a transaction that is still active should not be written to disk.
    - Steal policy : Allows system to write modified blocks to disk even if the transactions that made those modifications have not all committed.

    - Recovery algorithms that we have discussed works in both cases (with Write Ahead Logging)
        - Exercise : Why is WAL required in steal policy to ensure proper recovery ?

## Buffer management

- Database buffering:
  - If block B is output to disk, all log records pertaining to data in B should be output to disk before B is output
  - No write to B is permitted when B is being output to disk.
    - Ensured by using locks

  - Transaction acquires lock on block B (in buffer) before it performs a write on a data item
    - The lock is released immediately after the update has been performed.

  - When a block is output to the disk:
    - Obtain an exclusive lock on the block, to ensure that no transaction is performing a write on the block.
    - Output log records to stable storage until all log records pertaining to block have been output.
    - Output the block to disk.
    - Release the lock once the block output has completed.

## Buffer management

- Database buffering:
    - Database systems usually have a process that continually cycles through the buffer blocks, outputting modified buffer blocks back to disk.
    - The number of dirty blocks in the buffer is eventually minimized
        - Blocks that have been modified in the buffer but have not been output.

## Buffer management

- Role of OS in buffer management:
  - Approach 1 : The database system reserves part of main memory to serve as a buffer and manages it
    - The database system manages data-block transfer
  - Drawbacks
    - The buffer must be kept small enough that other applications have sufficient main memory available for their needs.
    - So even when the other applications are not running, the database will not be able to make use of all the available memory.
    - Conversely, a non-database applications may not use that part of main memory reserved for the database buffer, even if some part of the database buffer are not being used.

# Buffer management

- Role of OS in buffer management:
  - Approach 2 : The database system implements its buffer within the virtual memory provided by the operating system.
    - Since the operating system knows about the memory requirements of all processes in the system, it can decide what buffer blocks must be force-output to disk, and when.
    - But, OS should ensure write-ahead logging requirements
    - So transfer of database files must be handled by the database system to enforce WAL

## Buffer management

- Fuzzy checkpoints:
  - Checkpoint operation might take long time to finish
  - Can result in interruption of transaction processing
  - Fuzzy checkpoint : Permit transaction updates to resume as soon as checkpoint is recorded in buffer (but before the modified blocks are transferred to disk)
- Possible that the system could crash before all blocks are written.
  - Checkpoint on disk may be incomplete.
  - Write out the last checkpoint that has been successfully completed in a fixed location on the disk.
    - Update this only after the data output is complete

## Failure of Non-volatile storages

- Dumps are like checkpoints for non-volatile storages

- Most systems support SQL dump
  - Writes Data Definition Language and insert statements to a file
  - Can be executed to recreate a database
  - Useful for migration

- In case of partial failure of storage, only the lost blocks are restored

- Simple dumps are costly since it copies entire database
- Transaction processing is halted during data transfer
  - Fuzzy dumps allow transactions to be active while dump is in progress

## ARIES

- State of the art recovery method
- Algorithms for Recovery and Isolation Exploiting Semantics
- Reduces the time taken for recovery and the overhead of checkpointing
  - Avoid redoing many logged operations that have already been applied and to reduce the amount of information logged.

# ARIES

- Data Structures used:
  - Each log record in ARIES has a log sequence number (LSN) that uniquely identifies the record.
  - The number is conceptually just a logical identifier whose value is greater for log records that occur later in the log.

  - ARIES splits a log into multiple log files, each of which has a file number.
  - When a log file grows to some limit, ARIES appends further log records to a new log file;
    - The new log file has a file number that is higher by 1 than the previous log file.
  - The LSN then consists of a file number and an offset within the file.

  - Whenever an update operation on a block occurs, we store the LSN of its log record for the block as BlockLSN
  - During the redo phase of recovery, any log records with LSN less than or equal to the BlockLSN need not be executed on the block, since their actions are already reflected

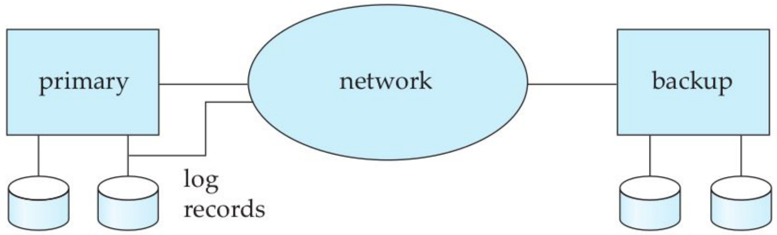
  - This helps avoid reading many blocks for which logged operations are already reflected on disk.
  - Thereby, recovery time is reduced significantly.

## ARIES

- ARIES Algorithm:
  - Analysis Pass:  Determines which transactions to undo, which blocks were dirty at the time of the crash, and the LSN from which the redo pass should start.
  - Redo Pass: Starts from a position determined during analysis, and performs a redo, to bring the database to a state it was in before the crash.
  - Undo Pass: This pass rolls back all transactions that were incomplete at the time of crash.

# Remote backup systems

- High availability cannot be ensured by centralized database systems
- Have a primary database and a remote backup where all data from primary source is replicated
  - In a physically distant location
- Remote site is kept synchronized with Primary site
  - By sending all log files from primary to remote
- In case of crash of primary, remote starts accepting the transactions
  - Availability is greatly increased



## Remote backup systems

- Detection of failures :
  - Remote should detect that primary has failed.
    - By maintaining several independent communication channels

- Transfer of control :
  - When primary fails, the backup site takes over processing and becomes the new primary.
  - When the original primary site recovers, it can either play the role of remote backup or take over the role of primary site again.
    - In both cases, old primary must receive a log of updates carried out by the backup site while the old primary was down.

## Remote backup systems

- Time to recovery :
  - If the log at the remote backup grows large, recovery will take a long time.
  - The remote backup site can periodically process the redo log records that it has received and can perform a checkpoint
    - so that earlier parts of the log can be deleted.
    - The delay before the remote backup takes over can be significantly reduced as a result.

  - A hot-spare configuration can ensure that the takeover by the backup site almost instantaneous.
    - In this configuration, the remote backup site continually processes redo log records as they arrive, applying the updates locally.
    - As soon as the failure of the primary is detected, the backup site completes recovery by rolling back incomplete transactions

## Remote backup systems

- Time to commit : To ensure that the updates of a committed transaction are durable
  - a transaction must not be declared committed until its log records have reached the backup site.
  - This delay can result in a longer wait to commit a transaction

- Degrees of durability :
  - One-safe : A transaction commits as soon as its commit log record is written to stable storage at the primary site.

  - Two-very-safe : A transaction commits as soon as its commit log record is written to stable storage at the primary and the backup site.
    - If one copy fails, this protocol cannot proceed
    - Should wait until the both copies are restored

  - Two-safe : Same as two-very-safe if both primary and backup sites are active.
    - If only the primary is active, the transaction is allowed to commit as soon as its commit log record is written to stable storage at the primary site.

- Alternative way to achieve high availability is to use distributed database

## Reference:

Database System Concepts by Silberschatz, Korth and Sudarshan

(6th edition)

Chapter 16