Recovery System

Recovery Alogrithm

- **Recovery Algorithm** : Algorithm that ensure atomicity and durability of transaction
    - **Atomicity** : All or nothing
    - **Durability** : Safe save
- 2 parts
    - **Analysis** : phase *during normal transaction* - Check which transaction need to be redone or undone and log enough information for recovery
    - **Redo** : phase *after failure* - Redo all transaction that need to be redone to ensure atomicity, durability and *consistency*

Log-based Recovery

- Assumption
    - serial transaction
    - log is stored in stable storage directly
- **Log** : Sequence of **log records**
    - log must be stored in *stable storage*
    -
- **Log record** : Record that contain information about transaction
    - Start of transaction : $<T_i \text{ Start}>$
    - Before $<T_i>$ executes write(X) : $<T_i, X, V_{old}, V_{new}>$
    - Finish of last statement : $<T_i \text{ Commit}>$
    - Rollback : $<T_i \text{ Abort}>$

Immediate Database Modification

- Reflect all update to database immediately **during transaction executed**
    - Log record is written **before** data is updated
- Logging for Immediate Database Modification
    - Transaction start : $<T_i \text{ Start}>$
    - Write(X) opeartion results in
        - write log : $<T_i, X, V_{old}, V_{new}>$
        - update data(log is *prior* to data writing)
    - When T_i
- Example : (B_X denotes write block containing X from buffer to disk)

Log	Write	Output/Remarks
<T_0 Start>		
<T_0, A, 1000, 2000>		
	A = 2000	Update A in *bufferd* block
<T_0, B, 2000, 2050>		
	B = 2050	Update B in *bufferd* block
<T_0 Commit>		Commit but not write to disk
<T_1 Start>		
<T_1, C, 700, 600>		
	C = 600	Update C in *bufferd* block
		B_B, B_C
T_1 Commit		
		B_A

- Note that **disk write sequence is not same as log sequence**
  - Because log is written before data is updated
  - So, log sequence is not same as data update sequence

Redo and Undo : Based on Log

- **Redo** : Re-execute transaction that is not completed(to enusre durability)
  - **Redo** is needed when transaction is committed/aborted but not written to disk
  - Start from the **first** log record of transaction
- **Undo** : Rollback transaction that is not completed(to ensure atomicity)
  - **Undo** is needed when transaction is not committed/aborted
  - Start from the **last** log record of transaction
- Both **Redo** and **Undo** must be idempotent
  - **Idempotent** : Operation that can be applied multiple times without changing result
  - **Redo** and **Undo** must be idempotent because
    - **Redo** and **Undo** may be applied multiple times
    - Becuase of failure during **Redo** and **Undo** is possible
- Example

Case1	Case2	Case3
<T_0 Start>	<T_0 Start>	<T_0 Start>

Case1	Case2	Case3
<T_0, A, 1000, 950>	<T_0, A, 1000, 950>	<T_0, A, 1000, 950>
<T_0, B, 2000, 2050>	<T_0, B, 2000, 2050>	<T_0, B, 2000, 2050>
	<T_0 Commit>	<T_0 Commit>
	<T_1 Start>	<T_1 Start>
	<T_1, C, 700, 600>	<T_1, C, 700, 600>
		<T_1 Commit>

- Case1 : Fail during T_0
  - Undo <T_0
- Case2 : Commit T_0 but fail during T_1
  - Redo <T_0
  - Undo <T_1
- Case3 : Commit T_0 and T_1
  - Redo <T_0, <T_1>

Checkpoint

- Problem of **Log-based Recovery**
  - **Redo** and **Undo** is time consuming
  - **Log** is large
    - searching entire log is time consuming
    - storing entire log is space consuming
  - To solve this problem, **Checkpoint** is used
- **Checkpoint** : Point in time that all buffered data(include **log**) is written to disk
  - Sequence
    - Write all log reocrds in main memory to disk
    - Write all modified buffer block to disk
    - Write **Checkpoint** record to log
  - Checkpoint is used to reduce **Redo** and **Undo** time
  - Checkpoint is used to reduce **log size**

Transaction Rollback - during normal execution

- Rollback is Undo of transaction
- Rollback of transaction T_i
  - Search log from the end(for undo)
  - For each log record <T_i, X, V_{old}, V_{new}>
    - Write V_{old} to X

        - Write <T_i, X, V_{old}> : *redo-only log record*
    - Search until <T_i Start> is found
        - Write <T_i Abort> to log
        - T_i enters **abort** state
            - Note that abort need **redo** operation not undo

Recovery in Concurrent Transactions

- Extend the log-based recovery schemes
    - All transactions share a single disk buffer and a **single log**
    - log records of different transactions may be interspersed in the log
    - Buffer block may contain updated data of different transactions
- Assume using strict 2PL
    - **Strict 2PL** : Transaction holds all locks until it commits/aborts
    - **Strict 2PL** is used to ensure **serializability**
    - **Strict 2PL** is used to ensure **recoverability**

Checkpoint and Crash

- Decide Redo or Undo using Checkpoint time, Crash time, Committed time
    - Before Checkpoint : Nothing to do
    - After Checkpoint, Before Crash : Redo
    - After Crash : Undo

Recovery after System Crash

1. Redo Phase(repeating history)
    - Scan log *forward* from the beginning(last checkpoint)
    - Redo Operation
        - <T_i, X, V_{old}, V_{new}> : Write V_{new} to X
        - Write <T_i, X, V_{old}> : *redo-only log record*
    - Add to undo-list
        - <T_i, start>
    - Remove from undo-list
        - <T_i, abort> or <T_i, commit>
2. Undo Phase
    - Scan log *backward* from the end
    - Undo operation
        - Perform undo action and write redo-only log record(same as rollback)
    - Meet <T_i, start>
        - Write <T_i, abort> to log
        - remove from undo-list

- Terminate when undo-list is empty

Recovery after System Crash : Example

```
<T0, start>
<T0, B, 2000, 2050>
<T1, start>
<checkpoint {T0, T1}>
<T1, C, 700, 600>
<T1, commit>
<T2, start>
<T2, A, 500, 400>
<T0, B, 2000>
<T0, abort>
CRASH!
<T2, A, 500>
<T2, abort>
```

- Analysis
  - Rollback operation of T_0
    - Undo <T_0, B, 2000, 2050>
    - Write <T_0, B, 2000> : Redo-only log record
    - Write <T_0, abort>
  - Redo Phase : start from the check point
    - : add T_0, T_1 to undo-list
    - <T_1, C, 700, 600> : Redo
    - <T_1, commit> : Remove from undo-list
    - <T_2, start> : Add to undo-list
    - <T_2, A, 500, 400> : Redo
    - <T_0, B, 2000> : Redo
    - <T_0, abort> : Remove from undo-list
  - Undo Phase : start from the end(only check T_2)
    - <T_2, A, 500, 400> : Undo the redo operation
      - Write <T_2, A, 500> : Redo-only log record
    - <T_2, start> : Remove from undo-list
      - Write <T_2, abort>

Recovery after System Crash : Example2

```
<T0, start>
<T0, A, 0, 10>
<T0, commit>
<T1, start>
<T1, B, 0, 10>
```

```
<T2, start>
<T2, C, 0, 10>
<T2, C, 10, 20>
<checkpoint {T1, T2}>
<T3, start>
<T3, A, 10, 20>
<T4, start>
<T4, D, 0, 10>
<T3, commit>
CRASH!
```

- Analysis
  - Redo : start from the check point
    - : Add T_1, T_2 to undo-list
    - <T_3, start> : Add to undo-list
    - <T_3, A, 10, 20> : Redo
    - <T_4, start> : Add to undo-list
    - <T_4, D, 0, 10> : Redo
    - <T_3, commit> : Remove from undo-list
  - Undo : start from the end
    - Only T_4 in undo-list
    - <T_4, D, 0, 10> : Undo
      - Write <T_4, D, 0> : Redo-only log record
    - <T_4, start> : Remove from undo-list
      - Write <T_4, abort>
    - <T_2, C, 10, 20> : Undo
      - Write <T_2, C, 10> : Redo-only log record
    - <T_2, C, 0, 10> : Undo
      - Write <T_2, C, 0> : Redo-only log record
    - <T_2, start> : Remove from undo-list
      - Write <T_2, abort>
    - <T_1, B, 0, 10> : Undo
      - Write <T_1, B, 0> : Redo-only log record
    - <T_1, start> : Remove from undo-list
      - Write <T_1, abort>

Log Record Buffering

- **Log Record Buffering** : Buffer log records in main memory
  - Normally, not directly written to disk
- Write log records to disk when
  - Buffer is full
  - **Log force** : Write all log records in buffer to disk
    - After commit

Write-Ahead Logging(WAL)

- **Write-Ahead Logging(WAL)** : Write log records before writing data
    - Write log records to disk before writing data to disk
    - Guarantee atomicity and durability
- Rules
    1. Log record saved in stable storage in order in which they are written
    2. T_i can only enter to commit state after all log records for T_i have been written to stable storage
    3. Before a data item is written to disk, the log record for the write must be written to stable storage