RECOVERY & SECURITY

Lecture 2

LOG-BASED RECOVERY (1/4)

- Log is a sequence of log records, recording all update activities in the database

- Three fields
  - **Transaction identifier** : Unique for the transaction that performed the write operation
  - **Data-item identifier** : Unique for data item written. It is location on disk of data item
  - **Old value** : Value of data item before the write
  - **New value** : Value that the data item will have after the write

LOG-BASED RECOVERY (2/4)

- Various types of log records are denoted as
  - **<T_i start>** : Transaction T_i has started
  - **<T_i, X_j, V_1, V_2>** : Transaction T_i has performed a write on data item X_j. X_j had value V_1 before the write and will have value V_2 after the write
  - **<T_i commit>** : Transaction T_i has committed
  - **<T_i abort>** : Transaction T_i has aborted

LOG-BASED RECOVERY (3/4)

- Whenever a transaction performs a write, the log record for that write must be created before the database is modified
- Once a log record exists, we can output the modification to the database
- Can also undo the modifications that has already been output to the database by using the old value field in log records
- For log records to be useful for recovery from system and disk failures, the **log must reside in stable storage**

LOG-BASED RECOVERY (4/4)

- Assume that every log record is written to the end of the log on stable storage as soon as it is created

- Two techniques using log-based:
  - Deferred database modification
  - Immediate database modification

DEFERRED DATABASE MODIFICATION (1/13)

- **Records all database modifications in the log, but deferring (delayed) the execution of all write operations of a transaction until transaction partially commits**
- Assumes that transactions are executed serially
- When a transaction partially commits, information on the log associated with the transaction is used in executing the deferred writes
- If system crashes before transaction completes its execution, or it aborts, then information on log is ignored

DEFERRED DATABASE MODIFICATION (2/13)

- The execution of transaction T_i proceeds as follows
  - Before T_i starts its execution, a record <T_i start> is written to the log
  - A write(X) operation by T_i results in the writing of a new record to the log
  - Finally, when T_i partially commits, a record <T_i commit> is written to the log

DEFERRED DATABASE MODIFICATION (3/13)

- When T_i partially commits, the records associated with it in the log are used in executing the deferred writes
- Since failure may occur while this updating is taking place, before the start of these updates, all the log records are written out to stable storage
- Once they have been written, the actual updating takes place and the transaction enters the committed state

DEFERRED DATABASE MODIFICATION (4/13)

- **Only the new value of the data item is required by the deferred database modification technique**

- Eg. Let T_0 be a transaction that transfers Rs.50 from account A to B

  T_0 :

  ```
  read(A);
  A:=A-50;
  write(A);
  read(B);
  B:=B+50;
  write(B);
  ```

DEFERRED DATABASE MODIFICATION (5/13)

- Let T_1 be transaction that withdraws Rs.100 from account C

  T_1 : read(C);
  C:=C-100;
  write(C);

- Suppose these transactions are executed serially in the order T_0 followed by T_1, and the values of A,B,C before execution were Rs.1000, Rs.2000 and Rs.700 respectively

- Portion of the log containing relevant information on these two transactions is as shown

DEFERRED DATABASE MODIFICATION (6/13)

<T_0 start>
<T_0, A, 950>
<T_0, B, 2050>
<T_0 commit>
<T_1 start>
<T_1, C, 600>
<T_1 commit>

- Here, value of A is changed in database only after the record <T_0, A, 950> has been placed in the log

DEFERRED DATABASE MODIFICATION (7/13)

Log	Database
<T_0 start>	
<T_0, A, 950>	
<T_0, B, 2050>	
<T_0 commit>	
	A=950
	B=2050
<T_1 start>	
<T_1, C, 600>	
<T_1 commit>	
	C=600

DEFERRED DATABASE MODIFICATION (8/13)

- Recovery scheme uses the following recovery procedure
  - **redo(T_i)** – sets the value of all data items updated by T_i to the new values

- The redo operation must be **idempotent** i.e. executing it several times must be equivalent to executing it once

DEFERRED DATABASE MODIFICATION (9/13)

- **T_i needs to be redone** if and only if the **log contains both** record **<T_i start> and <T_i commit>**

- If system crashes after transaction completes its execution, the recovery scheme uses the information in the log to restore the system to a previous consistent state after the transaction had completed

- Example : Let us suppose that the system crashes before the completion of the transactions

DEFERRED DATABASE MODIFICATION (10/13)

- The log at that time is as shown below
  <T_0 start>
  <T_0, A, 950>
  <T_0, B, 2050>

- **No redo** actions need to be taken, since no commit statement

- Value of A and B remain Rs.1000 and Rs.2000 respectively

- Log records of the incomplete transaction T_0 can be deleted from the log

DEFERRED DATABASE MODIFICATION (11/13)

- The log at the time of crash is as shown
  <T_0 start>
  <T_0, A, 950>
  <T_0, B, 2050>
  <T_0 commit>
  <T_1 start>
  <T_1, C, 600>

- Need to **redo(T_0)**, since the record <T_0 commit> appears in the log on disk

- Hence, values of A = Rs.950 and B = Rs.2050, C remains Rs.700

- The log records of incomplete transaction T_1 can be deleted from the log

DEFERRED DATABASE MODIFICATION (12/13)

- Log at the time of the crash is as shown

  <T_0 start>
  <T_0, A, 950>
  <T_0, B, 2050>
  <T_0 commit>
  <T_1 start>
  <T_1, C, 600>
  <T_1 commit>

- Operations **redo(T_0)** and **redo(T_1)** are performed

- Hence, values of A,B,C are Rs.950, Rs.2050 and Rs.600 respectively

DEFERRED DATABASE MODIFICATION (13/13)

- Suppose there is a second system crash occurs during recovery from the first crash
- Some changes may have been made to the database as a result of redo operation, but all changes may not have been made
- When system comes up after second crash, recovery proceeds exactly as in the preceding examples
- For each commit record <T_i commit> found in the log, the system performs the redo(T_i) operation
- **It restarts recovery actions from the beginning**

IMMEDIATE DATABASE MODIFICATION (1/9)

- **Allows database modification to be output to the database while transaction is still in active state**

- Data modifications written by active transactions are called **uncommitted modifications**

- In the event of crash, system **must use the old value field** of the log records

IMMEDIATE DATABASE MODIFICATION (2/9)

- Before transaction T_i starts execution, system writes the record <T_i start> to the log

- During execution, any write(X) operation by T_i is preceded by the writing of appropriate new update record to the log

- When T_i commits, the system writes the record <T_i commit> to the log

IMMEDIATE DATABASE MODIFICATION (3/9)

- Portion of the log containing relevant information

  <T_0 start>

  <T_0, A, 1000, 950>

  <T_0, B, 2000, 2050>

  <T_0 commit>

  <T_1 start>

  <T_1, C, 700, 600>

  <T_1 commit>

IMMEDIATE DATABASE MODIFICATION (4/9)

Log	Database
<T_0 start>	
<T_0, A, 1000, 950>	
<T_0, B, 2000, 2050>	
	A=950
	B=2050
<T_0 commit>	
<T_1 start>	
<T_1, C, 700, 600>	
	C=600
<T_1 commit>	

IMMEDIATE DATABASE MODIFICATION (5/9)

- Recovery scheme uses two recovery procedures
  - **undo(T_i) –** restores the value of all data items updated by T_i to the old values
  - **redo(T_i) –** sets the value of all data items updated by T_i to new values
- **Undo and redo operations must be idempotent**

IMMEDIATE DATABASE MODIFICATION (6/9)

- After failure has occurred, recovery scheme consults the log to determine which transactions need to be redone, and which need to be undone
  - T_i needs to be **undone** if log contains the record ***<T_i start>*, but *does not* contain record *<T_i commit>***
  - T_i needs to be **redone** if log contains **both the record *<T_i start> and* the record *<T_i commit>***

IMMEDIATE DATABASE MODIFICATION (7/9)

- Values of A and B (on disk) are restored to Rs.1000 and Rs.2000 respectively

  <T_0 start>
  <T_0, A, 1000, 950>
  <T_0, B, 2000, 2050>

- **undo(T_0)** operation is performed, since no commit record in log

- Values of A, B and C are still the same

IMMEDIATE DATABASE MODIFICATION (8/9)

<T_0 start>

<T_0, A, 1000, 950>

<T_0, B, 2000, 2050>

<T_0 commit>

<T_1 start>

<T_1, C, 700, 600>

- Operations : **undo(T_1)** and **redo(T_0)** are performed
- Values of A,B,C are Rs.950, Rs.2050 and Rs.700 respectively

IMMEDIATE DATABASE MODIFICATION (9/9)

- Values of A,B,C are Rs.950, Rs.2050 and Rs.600 respectively

  <T_0 start>

  <T_0, A, 1000, 950>

  <T_0, B, 2000, 2050>

  <T_0 commit>

  <T_1 start>

  <T_1, C, 700, 600>

  <T_1 commit>

- Operations : **redo(T_0)** and **redo(T_1)** are performed

- Values are : A = 950, B = 2050, C = 600

CHECKPOINTS (1/8)

- When a system failure occurs, we must consult the log to determine those transactions that need to be redone and those that need to be undone

- **Need to search the entire log**

- Two major difficulties
  - Search is time-consuming, therefore, longer recovery time
  - Overhead - since most of the transactions that need to be redone have already written their updates into the database

CHECKPOINTS (2/8)

- To reduce overheads, introduce checkpoints
- During execution, the system maintains the log, and in addition it periodically performs checkpoints
- These require the following **sequence of actions** to take place:
  1. Output onto stable storage all log records currently residing in main memory
  2. Output to the disk all modified buffer blocks
  3. Output onto stable storage a log record **<checkpoint>**

CHECKPOINTS (3/8)

- **Transactions are not allowed to perform any update actions while the checkpoint is in progress**
- Presence of a <checkpoint> record in the log allows the system to streamline the recovery procedure
- Consider transaction T_i that committed prior to the checkpoint
- For such a transaction, the <T_i commit> record appears in the log before the <checkpoint> record

CHECKPOINTS (4/8)

- Any database modifications made by T_i must have been written to database either prior to the checkpoint or as part of the checkpoint itself
- Thus, at recovery time, there is no need to perform a redo operation on T_i
- After a failure has occurred, the recovery scheme examines the log to determine the most recent transaction T_i that started executing before the most recent checkpoint took place

CHECKPOINTS (5/8)

- **Search the log backward**, from the end of the log till it finds the final <checkpoint> record; then it continues to search backward until it finds the next <T_i start> record
- This record identifies a transaction T_i
- Let us denote these transactions by the set T
- Remainder of the log can be ignored, and erased whenever desired

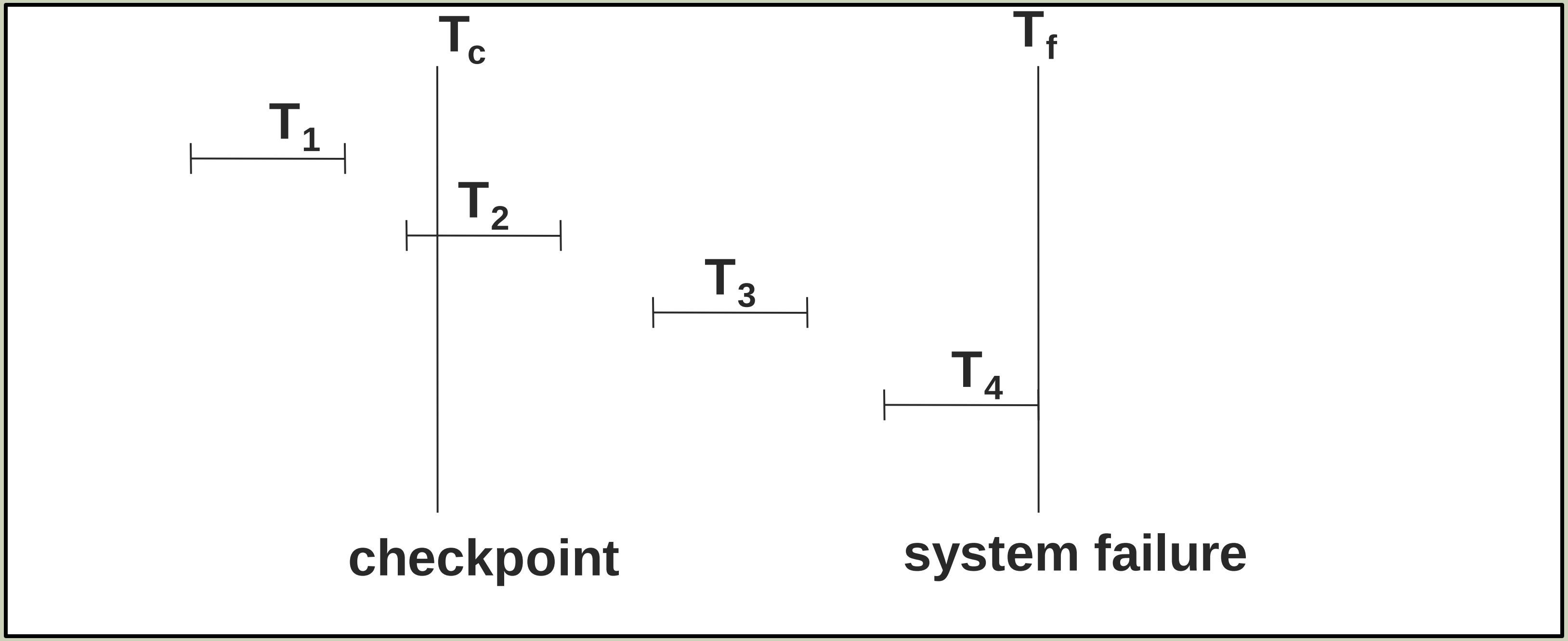
CHECKPOINTS (6/8)

- For immediate-modification technique, the recovery operations are
  - For all transaction T_k in T that have no <T_k commit> record in the log, execute undo(T_k)
  - For all transaction T_k in T such that the record <T_k commit> appears in the log, execute redo(T_k)
- The undo operation does not need to be applied when the deferred-modification technique is used

CHECKPOINTS (7/8)

- Example
  - If the set of transactions {T0, T1, …T10} are executed
  - Suppose that the most recent checkpoint took place during execution of T5
  - Only the transactions T5, T6, .. T10 need to be considered during recovery scheme
  - Each of them needs to be redone if it has committed otherwise it needs to be undone

CHECKPOINTS (8/8)



T_1 can be ignored (updates already output to disk due to checkpoint)

T_2 and T_3 redone

T_4 undone