Lecture 11: Database Recovery Techniques

CS3402 Database Systems

Recovery Concepts

- Recovery from transaction failures usually means that the database is restored to the most recent consistent state before the time of failure.
- To do this, the system must keep information about the changes that were applied to data items by the various transactions.
- This information is typically kept in the **system log**.
- The system keeps track of all transaction operations that affect the values of database items, as well as other transaction information that may be needed to permit recovery from failures.

System Log

- The log is a sequential, append-only file that is kept on disk, so it is not affected by any type of failure except for disk failure.
- Typically, one (or more) main memory buffers, called the **log buffers**, hold the last part of the log file, so that log entries are first added to the log main memory buffer. When the **log buffer** is filled, or when certain other conditions occur, the log buffer is appended to the end of the log file on disk.
- In addition, the log file from disk is periodically backed up to archival storage (tape) to guard against disk failure.

Log Records

- The following are the types of entries—called **log records**—that are written to the log file and the corresponding action for each log record.
- In these entries, T refers to a unique **transaction-id** that is generated automatically by the system for each transaction and that is used to identify each transaction:
  - **[**start_transaction**, T].** Indicates that transaction T has started execution.
  - **[**write_item**, T, X, old_value, new_value].** Indicates that transaction T has changed the value of database item X from old_value to new_value.
  - **[**read_item**, T, X].** Indicates that transaction T has read the value of database item **X**.
  - **[**commit**, T].** Indicates that transaction T has completed successfully, and affirms that its effect can be committed (recorded permanently) to the database.
  - **[**abort**, T].** Indicates that transaction T has been aborted.

Checkpoint

- Another type of entry in the log is called a **checkpoint**. A [checkpoint, list of active transactions] record is written into the log periodically at that point **when the system writes out to the database on disk all DBMS buffers that have been modified.**
- As a consequence of this, all transactions that have their [commit, T] entries in the log before a [checkpoint] entry do not need to have their WRITE operations redone in case of a system crash, since all their updates will be recorded in the database on disk during checkpointing.
- As part of checkpointing, the list of transaction ids for active transactions at the time of the checkpoint is included in the checkpoint record, so that these transactions can be easily identified during recovery.
- The recovery manager of a DBMS must decide at what intervals to take a checkpoint.

Recovery Strategies

- We can distinguish two main strategies for recovery from transaction failures: **deferred update** and **immediate update**.
- The recovery strategy is to identify any changes that may cause an inconsistency in the database.
- For example, a transaction that has updated some database items on disk but has not been committed needs to have its changes reversed by **undoing** its write operations.
- It may also be necessary to **redo** some operations in order to restore a consistent state of the database; for example, if a transaction has committed but some of its write operations have not yet been written to disk.

Immediate Update

- In the **immediate update** techniques, the database may be updated by some operations of a transaction **before the transaction reaches its commit point**.
- However, these operations must also be recorded in the log on disk by force-writing before they are applied to the database on disk, making recovery still possible.
- For the algorithm where all updates are required to be recorded in the database on disk before a transaction commits requires undo only, so it is known as the **UNDO/NO-REDO algorithm**.

Undo-Logging Rules

- U1: if transaction T modifies database element X (**current value** is v), the log record of the form <T, X, v> must be written to the disk BEFORE the new value of X is written to disk
- U2: if a transaction commits, its COMMIT log record must be written to disk ONLY after all database elements changed by the transaction have been written to disk
- **A transaction must be written to disk in the following order**
  1. **The log records indicating changed database elements**
  2. **The changed database elements themselves**
  3. **The COMMIT log record**

Undo-Logging Rules and Actions

Action	t	A (MEM)	B (MEM)	A (DISK)	B (DISK)	Log
						<START T>
Read(A, t)	8	8		8	8	
t:= t * 2;	16	8		8	8	
Write(A, t);	16	16		8	8	<T, A, 8>
Read(B, t);	8	16	8	8	8	
t:= t * 2;	16	16	8	8	8	
Write(B, t)	16	16	16	8	8	<T, B, 8>
FLUSH LOG						(Order #1)
Output(A)	16	16	16	16	8	(Order #2)
Output(B)	16	16	16	16	16	(Order #2)
						<COMMIT T> (Order #3)
FLUSH LOG						

Recovery Using Undo-Logging (1/2)

- Suppose a system failure occurs while a transaction is committing. It is possible certain changes made by the transaction were written to disk while others changes never reached the disk
- Recovery manager has to REMOVE the partial changes of a transaction
- If there is a log record <COMMIT T>, by undo rule U2, all changes made by T were previously written to disk
- If we find <START T> on the log but no <COMMIT T> record, T may be incomplete and must be undone
  - Rule U1 assures that if T changed X on disk before the crash, there will be a <T, X, v> record on the log, and that record will have been copied to disk before the crash. During recovery, we must write the value v for X

Recovery Using Undo-Logging (2/2)

- The recovery manager scans the log from the end and remembers all those transactions T for which it has a <COMMIT T> record or an <ABORT T> record
- If it sees a record <T, X, v>
  - If T is a transaction whose COMMIT record has been seen, do nothing. T is committed and must not be undone
  - Otherwise, T is an incomplete transaction, or an aborted transaction. The recovery manager must change the value of X in the database to v in case X had been altered just before the crash
  - After making the changes, the recovery manager must write a log record <ABORT T> for each incomplete transaction that was not previously aborted and the flush the log

Deferred Update

- The **deferred update** techniques do not physically update the database on disk until after a transaction commits; then the updates are recorded in the database. Before reaching commit, all transaction updates are recorded in the local transaction workspace or in the main memory buffers that the DBMS maintains.
- Before commit**, the updates are recorded persistently in the log file on disk, and then after commit**, the updates are written to the database from the main memory buffers.
- If a transaction fails before reaching its commit point, it will not have changed the database on disk in any way, so UNDO is not needed. It may be necessary to REDO the effect of the operations of a committed transaction from the log, because their effect may not yet have been recorded in the database on disk.
- Hence, deferred update is also known as the **NO-UNDO/REDO algorithm**.

Redo-Logging for Recovery

- Undo logging has a potential problem that we cannot commit a transaction without first writing all changed data to disk (flush $\rightarrow$ commit)
- Differences between redo logging (deferred update) and undo logging (immediate update)
  - While undo logging cancels the effect of incomplete transactions and ignores committed ones during recovery, redo logging ignores incomplete transactions and repeats the changes made by committed transactions
  - While undo logging requires us to write changed database elements to disk before the COMMIT log records reaches disk, redo logging requires that the COMMIT record appears on disk before any changed values reach disk

Redo-Logging Rules

- R1: before modifying any database element X on disk, it is necessary that all log records pertaining to this modification of X, including both the update record <T, X, v> (v is the **new** value) and <COMMIT T> record, must appear on disk
- **The order in which material associated with one transaction gets to written to disk is:**
  1. **The log record indicating changed database elements**
  2. **The COMMIT log record**
  3. **The changed database elements themselves**
- **Undo-logging (immediate update): Log → change → COMMIT**
- **Redo-logging (deferred update): Log → COMMIT → change**

Redo-Logging Rules and Actions

Action	t	A (MEM)	B (MEM)	A (DISK)	B (DISK)	Log
						<START T>
Read(A, t)	8	8		8	8	
t:= t * 2;	16	8		8	8	
Write(A, t);	16	16		8	8	<T, A, 16>
Read(B, t);	8	16	8	8	8	
t:= t * 2;	16	16	8	8	8	
Write(B, t)	16	16	16	8	8	<T, B, 16>
						<COMMIT T>
FLUSH LOG						
Output(A)	16	16	16	16	8	
Output(B)	16	16	16	16	16	

Recovery Using Redo-Logging

- To recover using a redo log, after a system crash:
  - Identify the committed transactions
  - Scan the log forward from the beginning. For each log record <T, X, v> encountered:
    - If T is not a committed transaction, do nothing
    - If T is a committed transaction, write value v for database element X
  - For each incomplete transaction T, write an <ABORT T> record to the log and flush the log

Recovery Using Redo-Logging: Example (1/2)

- When the checkpoint was taken at time t_1, transaction T_1 had committed, whereas transactions T_3 and T_4 had not. Before the system crash at time t_2, T_3 and T_2 were committed but not T_4 and T_5.
- There is no need to redo the write_item operations of transaction T_1—or any transactions committed before the last checkpoint time t_1.

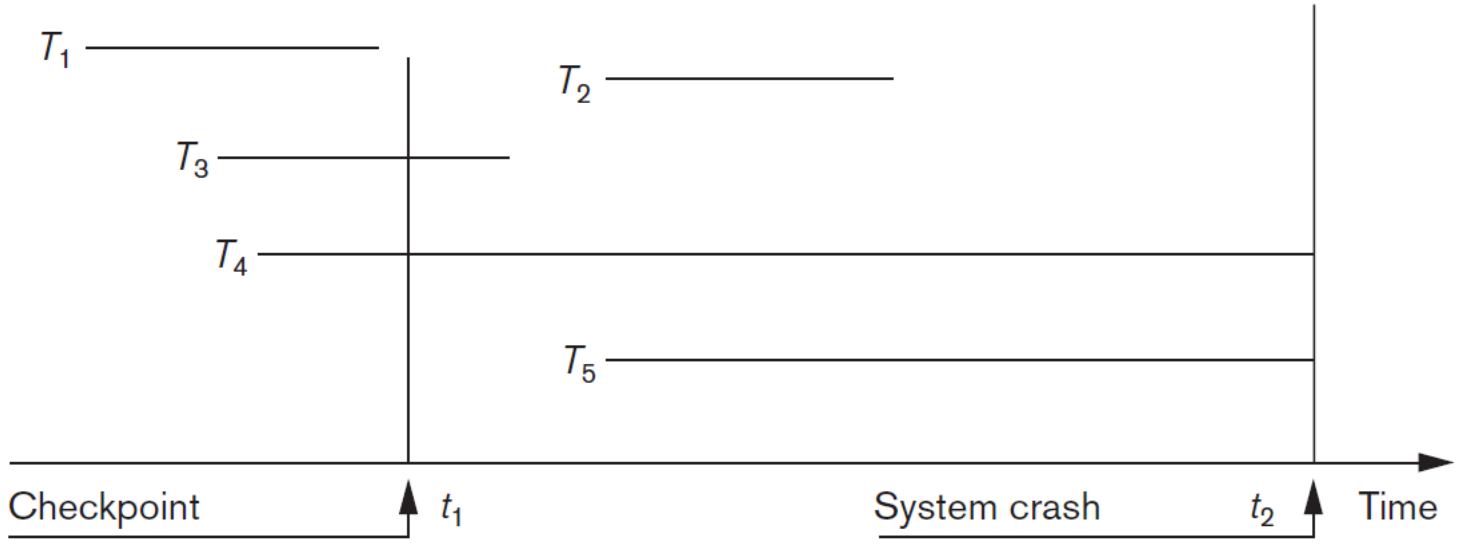

Figure 22.2
An example of a recovery timeline to illustrate the effect of checkpointing.

Recovery Using Redo-Logging Example (2/2)

- The write_item operations of T_2 and T_3 must be redone, however, because both transactions reached their commit points after the last checkpoint. Recall that the log is force-written before committing a transaction.
- Transactions T_4 and T_5 are ignored: They are effectively canceled or rolled back because none of their write_item operations were recorded in the database on disk under the deferred update protocol.

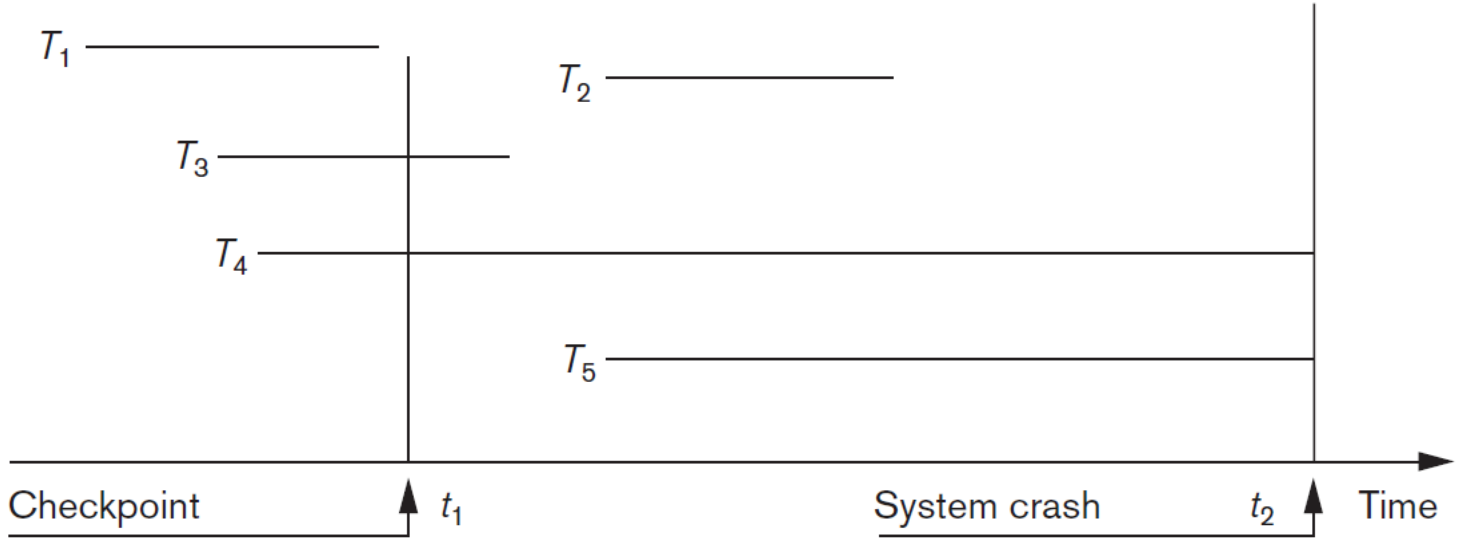

Figure 22.2
An example of a recovery timeline to illustrate the effect of checkpointing.