Chapter 22

Introduction to Database Recovery Protocols

Database Recovery

1 Purpose of Database Recovery

- To bring the database into the last consistent state, which existed prior to the failure.
- To preserve transaction properties (Atomicity, Consistency, Isolation and Durability).

Example: If the system crashes before a fund transfer transaction completes its execution, then either one or both accounts may have incorrect value. Thus, the database must be restored to the state before the transaction modified any of the accounts.

Database Recovery

2 Types of Failure

The database may become unavailable for use due to

- Transaction failure: Transactions may fail because of incorrect input, deadlock, incorrect synchronization.
- System failure: System may fail because of addressing error, application error, operating system fault, RAM failure, etc.
- Media failure: Disk head crash, power disruption, etc.

Database Recovery

3 Transaction Log

For recovery from any type of failure data values prior to modification (BFIM - BeFore Image) and the new value after modification (AFIM – AFter Image) are required. These values and other information is stored in a sequential file called Transaction log. A sample log is given below. **Back P** and **Next P** point to the previous and next log records of the same transaction.

T ID	**Back P**	**Next P**	**Operation**	**Data item**	**BFIM**	**AFIM**
T1	0	1	Begin			
T1	1	4	Write	X	X = 100	X = 200
T2	0	8	Begin			
T1	2	5	W	Y	Y = 50	Y = 100
T1	4	7	R	M	M = 200	M = 200
T3	0	9	R	N	N = 400	N = 400
T1	5	nil	End			

Database Recovery

4 **Data Update**

- **Immediate Update**: As soon as a data item is modified in cache, the disk copy is updated.
- **Deferred Update**: All modified data items in the cache is written either after a transaction ends its execution or after a fixed number of transactions have completed their execution.

Database Recovery

5 Data Caching

Data items to be modified are first stored into database cache by the Cache Manager (CM) and after modification they are flushed (written) to the disk. The flushing is controlled by m**odified (dirty)** and **pin-unpin** bits.

pin-Unpin: Instructs the operating system not to flush the data item.

Modified (dirty): Indicates the AFIM of the data item.

- **Shadow update**: The modified version of a data item does not overwrite its disk copy but is written at a separate disk location.
- **In-place update**: The disk version of the data item is overwritten by the cache version.

Database Recovery

6 Transaction Roll-back (Undo) and Roll-Forward (Redo)

To maintain atomicity, a transaction's operations are **redone** or **undone**.

Undo: Restore all BFIMs on to disk (Remove all AFIMs).

Redo: Restore all AFIMs on to disk.

Database recovery is achieved either by performing only Undos or only Redos or by a combination of the two. These operations are recorded in the log as they happen.

Database Recovery

Roll-back

We show the process of roll-back with the help of the following three transactions T1, and T2 and T3.

T1	T2	T3
read_item (A)	read_item (B)	read_item (C)
read_item (D)	write_item (B)	write_item (B)
write_item (D)	read_item (D)	read_item (A)
	write_item (D)	write_item (A)

Database Recovery

Roll-back: **One execution of T1, T2 and T3 as recorded in the log.**

		A	B	C	D
		30	15	40	20
	[start_transaction, T3]				
	[read_item, T3, C]				
*	[write_item, T3, B, 15, 12]		12		
	[start_transaction,T2]				
	[read_item, T2, B]				
**	[write_item, T2, B, 12, 18]		18		
	[start_transaction,T1]				
	[read_item, T1, A]				
	[read_item, T1, D]				
	[write_item, T1, D, 20, 25]				25
	[read_item, T2, D]				
**	[write_item, T2, D, 25, 26]				26
	[read_item, T3, A]				

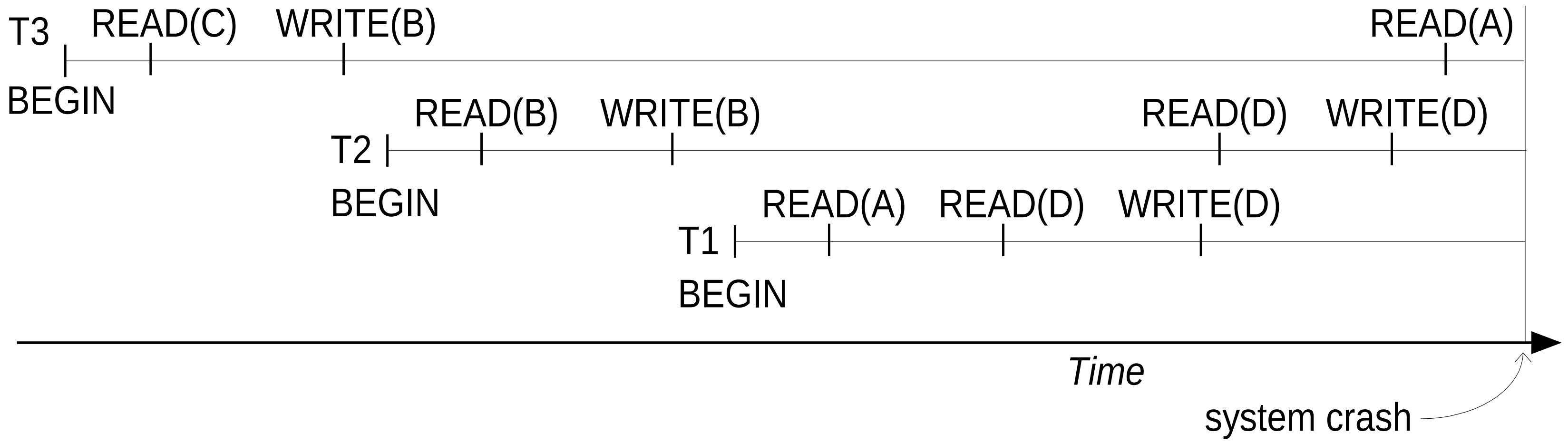
---- system crash ----

*** T3 is rolled back because it did not reach its commit point.**

**** T2 is rolled back because it reads the value of item B written by T3.**

Database Recovery

Roll-back: **One execution of T1, T2 and T3 as recorded in the log.**



Illustrating cascading roll-back

Database Recovery

Steal/No-Steal and Force/No-Force

Possible ways for flushing database cache to database disk:

Steal: Cache can be flushed before transaction commits.

No-Steal: Cache cannot be flushed before transaction commit.

Force: Cache is immediately flushed (forced) to disk.

No-Force: Cache is deferred until transaction commits.

These give rise to four different ways for handling recovery:

Steal/No-Force (Undo/Redo), Steal/Force (Undo/No-redo), No-Steal/No-Force (Redo/No-undo) and No-Steal/Force (No-undo/No-redo).

Database Recovery

Write-Ahead Logging

When **in-place** update (immediate or deferred) is used then log is necessary for recovery and it must be available to recovery manager. This is achieved by **Write-Ahead Logging** (WAL) protocol. WAL states that

For Undo: Before a data item's AFIM is flushed to the database disk (overwriting the BFIM) its BFIM must be written to the log and the log must be saved on a stable store (log disk).

For Redo: Before a transaction executes its commit operation, all its AFIMs must be written to the log and the log must be saved on a stable store.

Database Recovery

7 **Checkpointing**

Time to time (randomly or under some criteria) the database flushes its buffer to database disk to minimize the task of recovery. The following steps defines a checkpoint operation:

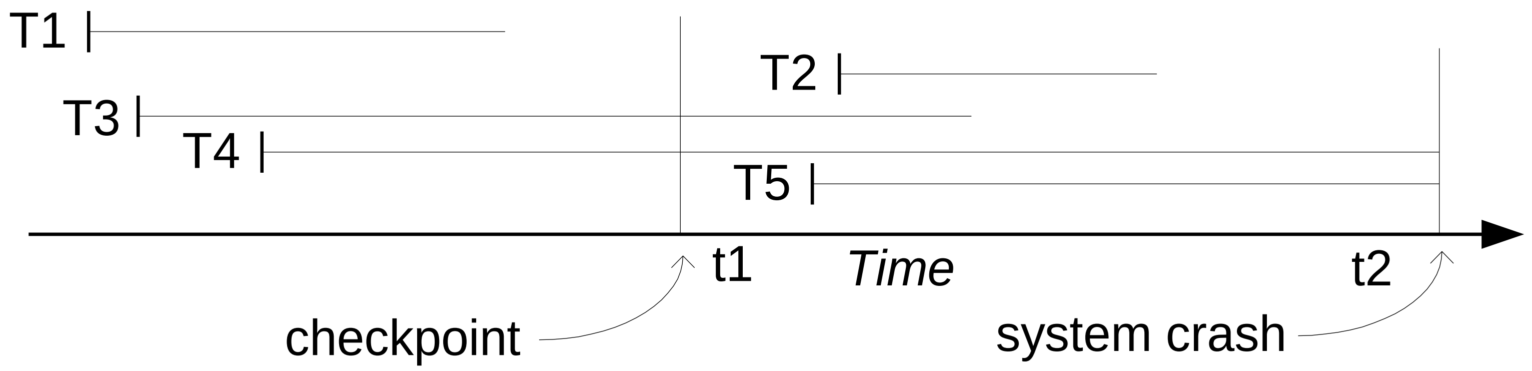
1. Suspend execution of transactions temporarily.
2. Force write modified buffer data to disk.
3. Write a [checkpoint] record to the log, save the log to disk.
4. Resume normal transaction execution.

During recovery **redo** or **undo** is required to transactions appearing after [checkpoint] record.

Database Recovery

Deferred Update with concurrent users

This environment requires some concurrency control mechanism to guarantee **isolation** property of transactions. In a system recovery transactions which were recorded in the log after the last checkpoint were **redone**. The recovery manager may scan some of the transactions recorded before the checkpoint to get the AFIMs.



Recovery in a concurrent users environment.