CSE2004 – Database Management Systems

Text Books :

1.R. Elmasri & S. B. Navathe, Fundamentals of Database Systems, Addison Wesley, 7 th Edition, 2015

2.Raghu Ramakrishnan, Database Management Systems,Mcgraw-Hill,4th edition,2015

Reference Book

3.A. Silberschatz, H. F. Korth & S. Sudershan, Database System Concepts, McGraw Hill, 6th Edition 2010

4.Thomas Connolly, Carolyn Begg," Database Systems : A Practical Approach to Design, Implementation and Management",6th Edition,2012

5.Shashank Tiwari ,"Professional NoSql",Wiley ,2011

VIT UNIVERSITY
(Estd. u/s 3 of UGC Act 1956)
www.vit.ac.in
Vellore ▪ Chennai
CHENNAI CAMPUS
Vandalur - Kelambakkam Road, Chennai - 600127

Unit – 5 : Concurrency Control and Recovery Techniques

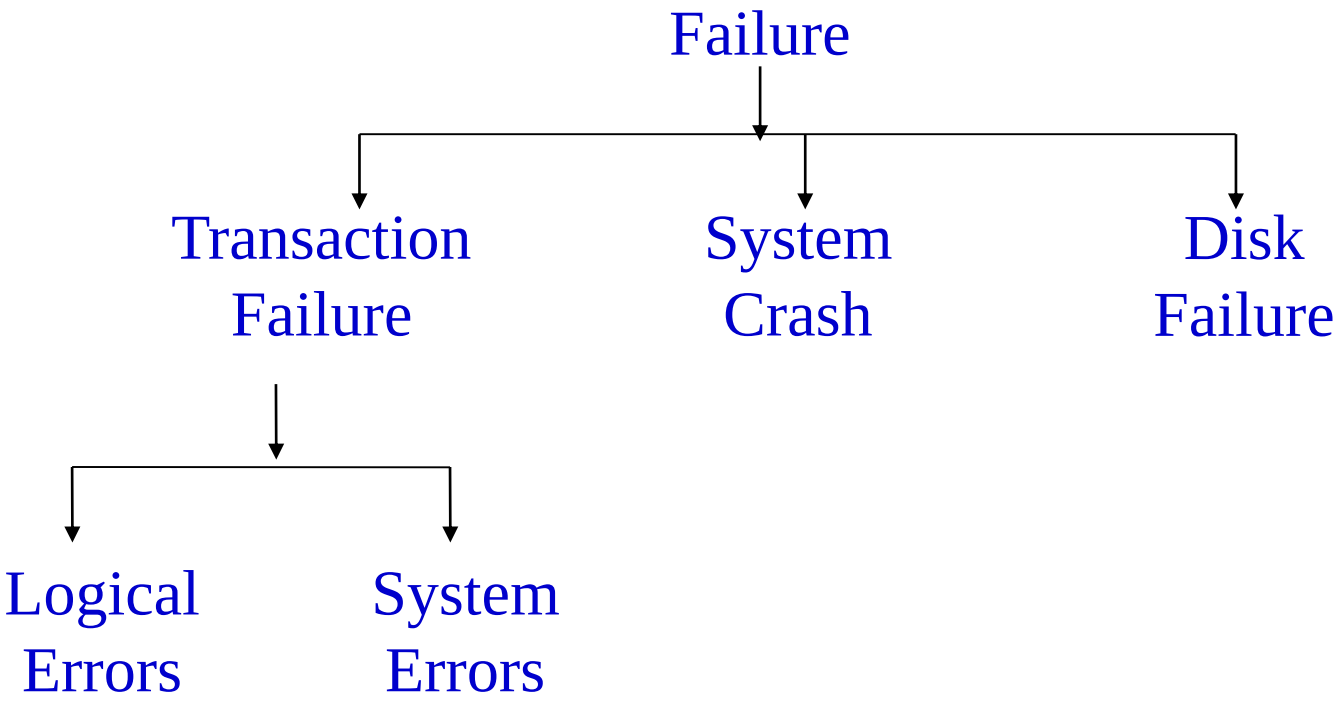
- ✓ **Introduction to Recovery Concepts**
  - ➢ **Recovery based on deferred update**
  - ➢ **Recovery based on immediate update**
  - ➢ **Shadow paging**

Recovery in DBMS

Recovery Means… (Crash Recovery)

- A database is a very huge system with lots of data and transaction.
- The transaction in the database is executed at each seconds of time and is very critical to the database.
- If there is any failure or crash while executing the transaction, then it expected that no data is lost.
- It is necessary to revert the changes of transaction to previously committed point (Check / Restore point).
- There are various techniques to recover the data depending on the type of failure or crash.

Classification of Failure



Recovery in DBMS

Recovery and Atomicity :

When a DBMS recovers from a crash, it should maintain the following −

- It should check the states of all the transactions, which were being executed.
- A transaction may be in the middle of some operation; the DBMS must ensure the atomicity of the transaction in this case.
- It should check whether the transaction can be completed now or it needs to be rolled back.
- No transactions would be allowed to leave the DBMS in an inconsistent state.

Recovery and Atomicity :

For Eg :

Consider transaction T_i that transfers \$50 from account A to account B; goal is either to perform all database modifications made by T_i or none at all.

Several output operations may be required for T_i (to output A and B). A failure may occur after one of these modifications have been made but before all of them are made

Recovery Algorithms in DBMS

Recovery algorithms are techniques to ensure database consistency and transaction atomicity and durability despite of failures

Recovery algorithms have two parts :

1. Actions taken during normal transaction processing to ensure enough information exists to recover from failures
2. Actions taken after a failure to recover the database contents to a state that ensures atomicity, consistency and durability

Recovery in DBMS

There are two types of techniques, which can help a DBMS in recovering as well as maintaining the atomicity of a transaction

- Maintaining the logs of each transaction, and writing them onto some stable storage before actually modifying the database(Log-Based Recovery).
- Maintaining shadow paging, where the changes are done on a volatile memory, and later, the actual database is updated.

Storage Structure

There are 3 types of Storage Structures :

1. **Volatile** (RAM/Cache) : Doesn't Survive System crashes
2. **Non-Volatile** (Disks/ROM) : Survives System Crashes
3. **Stable** : Mythical form of storage, survives all failures
   - **It** is a classification of computer data storage technology that guarantees ***atomicity*** for any given write operation and allows software to be written that is robust against some ***hardware*** and ***power*** failure

Stable Storage Implementation

Stable : Mythical form of storage, survives all failures

- Maintains multiple copies of each block on Non-Volatile
- Copies can be at remote sites to protect against disasters
- ***Failure during data transfer can result in inconsistent copies***
  - Successful completion
  - Partial/Total Failure
- ***Protection from failure during data transfer*** (One Solution)
  - Execute output operation as follows (assuming 2 copies of each block)
    - ❖ Write the information onto 1st physical block
    - ❖ When first write is successful, do the same on 2nd physical block
    - ❖ The output is completed only after 2nd write is successful

Stable Storage Implementation

Stable : Mythical form of storage, survives all failures

Copies of a block may differ due to failure during output operation.

To recover from failure:

- First find inconsistent blocks:
  - Expensive solution : Compare the two copies of every disk block.
  - Better solution ::
    - Record in-progress disk writes on non-volatile storage (Non-volatile RAM or special area of disk).
    - Use this information during recovery to find blocks that may be inconsistent, and only compare copies of these.
    - Used in hardware RAID systems
- If either copy of an inconsistent block is detected to have an error (bad checksum), overwrite it by the other copy.
- If both have no error, but are different, overwrite the second block by the first block.

Recovery Algorithms

Log Based Recovery : A log is kept on stable storage.

i.e., a sequence of log records, which maintains a record of update activities on database.

- When Transaction T_i starts, it registers itself by writing <T_i , Start>
- Before T_i executes write(x), a log record <**T_i, X, V_1, V_2**> is written, Where V_1 is the value of X before the write, and V_2 is the value to be written to X
- When T_i finishes it last statement, the log record < **T_i , commit**> is written
- We assume for now that log records are written directly to stable storage (that is, they are not buffered)

Two approaches

- Deferred database modification
- Immediate database modification.

Deferred database Modification:

- It records all modifications to the log, but defers all the writes to after partial commit.
- Assume that transactions execute serially
- Transaction starts by writing <T_i , **start**> record to log.
- A write(X) operation results in a log record <T_i, **X, V**> being written, where V is the new value for X
- The write is not performed on X at this time, but is deferred.
- When T_i partially commits, <T_i , **commit**> is written to the log
- Finally, the log records are read and used to actually execute the previously deferred writes.
- During recovery after a crash, a transaction needs to be redone iff both <T_i , **start**> and <T_i , **commit**> are there in the log.

Deferred database Modification:

- Redoing a transaction T_i (redo T_i) sets the value of all data items updated by the transaction to the new values.
- Crashes can occur while
  - The transaction is executing the original updates, or
  - While recovery action is being taken

For Eg : Consider the Transaction T_0 and T_1 (T_0 executes before T_1)

T_0
read (A)
A:=A-50
write(A)
read(B)
B:=B+50
write(B)

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

Deferred database Modification:

For Eg : Consider the Transaction T_0 and T_1 (T_0 executes before T_1)

T_0	T_1
read (A)	read (C)
A:=A-50	C:=C-100
write(A)	write(C)
read(B)	
B:=B+50	
write(B)	

Log Files

(a)	(b)	(c)
< T_0, Start >	< T_0, Start >	< T_0, Start >
< T_0, A, 950 >	< T_0, A, 950 >	< T_0, A, 950 >
< T_0, B, 2040 >	< T_0, B, 2040 >	< T_0, B, 2040 >
	< T_0, commit >	< T_0, commit >
	< T_1, start >	< T_1, start >
	<T_1, C, 600 >	<T_1, C, 600 >
		<T_1, commit>

If log on stable storage at time of crash is as in case:

(a) No redo actions need to be taken

(b) redo(T_0) must be performed since <T_0 ,commit> is present

(c) redo(T_0) must be performed followed by redo(T_1) since <T_0, commit > and < T_1, commit> are present.

Immediate database Modification:

- It allows database updates of an uncommitted transaction to be made as the writes are issued.
  - since undoing may be needed, update logs must have both old value and new value.
- update log record must be written before database item is written.
  - We assume that the log record is output directly to stable storage
  - Can be extended to postpone log record output, so long as prior to execution of an output(B) operation for a data block B, all log records corresponding to items B must be flushed to stable storage
- Output of updated blocks can take place at any time before or after transaction commit
- Order in which blocks are output can be different from the order in which they are written

Recovery Algorithms

Log Based Recovery

Immediate database Modification:

Log	Write	Output
$<T_0$ **start>**		
$<T_0$, A, 1000, 950>		
T_o, B, 2000, 2050		
	$A = 950$ $B = 2050$	
$<T_0$ **commit>**		
$<T_1$ **start>**		
$<T_1$, C, 700, 600>		
	$C = 600$	
		B_B, B_C
$<T_1$ **commit>**		
		B_A

- Note: B_X denotes block containing X.

Immediate database Modification:

- Recovery procedure has two operations instead of one:
  - ✓ undo(T_i) restores the value of all data items updated by T_i to their old values, going backwards from the last log record for T_i
  - ✓ redo(T_i) sets the value of all data items updated by T_i to the new values, going forward from the first log record for T_i
- Both operations must be idempotent
  - ✓ i.e., even if the operation is executed multiple times the effect is the same as if it is executed once
    - **Needed since operations may get re-executed during recovery**
- When recovering after failure:
  - ✓ Transaction Ti needs to be undone if the log contains the record <Ti, start>, but does not contain the record <Ti, commit>.
  - ✓ Transaction Ti needs to be redone if the log contains both the record <Ti, start> and the record <Ti , commit>.
- Undo operations are performed first, then redo operations.

Immediate database Modification:

Below we show the log as it appears at three instances of time.

(a)	(b)	(c)
<T_0 start>	<T_0 start>	<T_0 start>
<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>

Recovery actions in each case above are:

(a) undo (T_0): B is restored to 2000 and A to 1000.

(b) undo (T_1) and redo (T_0): C is restored to 700, and then *A* and *B* are set to 950 and 2050 respectively.

(c) redo (T_0) and redo (T_1): A and B are set to 950 and 2050 respectively. Then *C* is set to 600

Shadow Paging Recovery :

It is an alternative to log-based recovery techniques, which has both advantages and disadvantages.

It may require fewer disk accesses, but it is hard to extend paging to allow multiple concurrent transactions.

The paging is very similar to paging schemes used by the operating system for memory management.

Shadow Paging Recovery :

- The idea is to maintain two page tables during the life of a transaction: the *current* page table and the *shadow* page table.
- When the transaction starts, both tables are identical.
- The shadow page is never changed during the life of the transaction.
- The current page is updated with each **write** operation.
- Each ***table entry points to a page on the disk***.
- When the transaction is committed, the shadow page entry becomes a copy of the current page table entry and the disk block with the old data is released.
- If the shadow is stored in nonvolatile memory and a system crash occurs, then the shadow page table is copied to the current page table.
- This guarantees that the shadow page table will point to the database pages corresponding to the state of the database prior to any transaction that was active at the time of the crash, making aborts automatic.

Shadow Paging Recovery :

There are drawbacks to the shadow-page technique:

- **Commit overhead.** The commit of a single transaction using shadow paging requires **multiple blocks to be output.(i.e.,** *the current page table, the actual data and the disk address of the current page table*). Log-based schemes need to output only the log records.

- **Data fragmentation.** Shadow paging causes database pages to change locations (therefore, no longer contiguous).

- **Garbage collection.** Each time that a transaction commits, the database pages containing the old version of data changed by the transactions must become inaccessible. Such pages are considered to be *garbage* since they are not part of the free space and do not contain any usable information. Periodically, need to find all such pages and add them to the list of free pages. This process is called *garbage collection* and imposes additional overhead and complexity on the system.

Shadow Paging Recovery :

Advantages :

- No Overhead for writing log records.
- No Undo / No Redo algorithm.
- Recovery is faster.

Disadvantages

- Data gets fragmented or scattered.
- After every transaction completion database pages containing old version of modified data need to be garbage collected.
- Hard to extend algorithm to allow transaction to run concurrently


Queries