RECOVERY & SECURITY

Lecture 1

TYPES OF FAILURE (1/3)

- **Transaction failure**
  - Logical error
    - Transaction can no longer continue with its normal execution because of some internal condition, like bad i/p, data not found, overflow or resource limit exceeded
  - System error
    - System has entered an undesirable state (eg. Deadlock) as a result of which a transaction cannot continue with its normal execution.
    - Transaction can however be executed at a later time

TYPES OF FAILURE (2/3)

- **System crash**
  - Hardware malfunction or bug in database software or OS, that causes the loss of content of volatile storage and brings transaction processing to a halt
  - Content of non-volatile storage remains intact, and is not corrupted
- **Disk failure**
  - Disk block loses contents as a result of either a head crash or failure during a data-transfer operation
  - Copies of other disks, or archival backups on tertiary media, like tapes, are used to recover failure

TYPES OF FAILURE (3/3)

- Two algorithms proposed:

1. Actions taken during normal transaction processing to ensure that enough information exists to allow recovery from failures

2. Actions taken after failure to recover database contents to a state that ensures database consistency, transaction atomicity and durability

TYPES OF STORAGE (1/3)

- **Volatile storage**
  - Information does not usually survive system crashes
  - Eg. Main memory, cache memory
  - Fast access speed because of the speed of the memory access itself, and also it is possible to access any data item in volatile storage directly

TYPES OF STORAGE (2/3)

- **Non volatile storage**
  - Information survives system crashes
  - Eg. Disk and magnetic tapes
  - Disks are used for on line storage, tapes for archival storage
  - Slower access speed than volatile storage since tapes and disks are electromechanical rather than based entirely on chips

TYPES OF STORAGE (3/3)

- **Stable storage**
  - Information is never lost
  - Theoretically impossible to obtain, but it can be closely approximated by techniques that make data loss extremely unlikely

STABLE-STORAGE IMPLEMENTATION (1/4)

- Need to **replicate** the needed information in several non-volatile storage media (disks)
- **Use RAID technology**
- Simplest and fastest form of RAID is **mirrored disk**, which keeps two copies of each block, on separate disks

STABLE-STORAGE IMPLEMENTATION (2/4)

- Block transfer between memory and disk storage can result in
  - **Successful completion** – The transferred information arrived safely at its destination
  - **Partial failure** – A failure occurred in the midst of transfer, and the destination block has incorrect information
  - **Total failure** – The failure occurred sufficiently early during the transfer that the destination block remains intact

STABLE-STORAGE IMPLEMENTATION (3/4)

- System must maintain two physical blocks for each logical database block

- An output operation is executed as follows
  1. Write information onto the first physical block
  2. When the first write completes successfully, write the same information onto the second physical block
  3. The output is completed only after the second write completes successfully

STABLE-STORAGE IMPLEMENTATION (4/4)

- During recovery, system examines each pair of physical blocks
  - If both are same and no detectable errors exists, no further actions are necessary
  - If system detects error in one block, then it replaces its content with the content of the other block
  - If both blocks contain no detectable error, but differ in content, then system replaces the content of first block with the value of second

RECOVERY & ATOMICITY (1/2)

- Suppose, T_i that transfers Rs.50 from account A to B, with initial values of A and B being Rs.1000 and Rs.2000 respectively
- Suppose system crash occurred during execution of T_i, after Rs. 50 have been deducted from A, but before transferred to B
- Invoke one of the two possible recovery procedures:
  - **Reexecute T_i** : Value of A becomes Rs.900, rather than Rs.950. Thus, system enters inconsistent state
  - **Do not reexecute T_i** : Value of A becomes Rs.950 and B becomes Rs.2000. Thus, system enters inconsistent state

RECOVERY & ATOMICITY (2/2)

- Reason – we have modify the database without having assurance that the transaction will indeed commit
- To achieve atomicity, we first output the information describing the modifications to stable storage, without modifying database itself
- Assume transactions are executed serially i.e. only a single transaction is active at a time