B.Tech II year II Semester IT Department

Subject : Database Management Systems

UNIT - IV PART - II

Topics: Recovery System - failure classification, Storage, Recovery and Atomicity, Recovery Algorithm, Buffer Management, failure with loss of nonvolatile storage, Early lock Release and logical undo operations, Remote Backup Systems.

Recovery System

→ When the system recovers from failure, it can restore the latest dump.

→ It can maintain redo-list and undo-list as in checkpoints.

→ It can recover the system by consulting undo-redo lists to restore the state of all transactions upto last checkpoint.

failure classification

To see where the problem has occured we generalize the failure into various categories, as follows:

1) Transaction failure
2) logical errors
3) system errors
4) system crash
5) Disk failure.

Storage structure

The storage structure can be divided in various categories:

1) Volatile storage: This storage does not survive system crashes and mostly placed very closed to CPU by embedding them onto the chipset itself.
for Eg: main memory, cache memory.

2) Non-volatile storage: These memories are made to survive system crashes. They are huge in data storage capacity but slower in accessibility.
for Eg: Hard disks, magnetic tapes, flash memory..

Recovery and Atomicity

When a system crashes, it may have several transactions being executed and various files opened for them to modifying data items. Transactions are made of various operations, which are atomic in nature

→ It should check the states of all transactions, which were being executed.

→ A transaction may be in the middle of some operation; DBMS must ensure the atomicity of transaction in this case.

→ It should check whether the transaction can be completed now or needs to be rolled back.

→ No transaction would be allowed to left DBMS in inconsistent state.

contd" There are 2 types of techniques, which can help DBMS in recovering as well as maintaining the atomicity of transaction:

i) Maintaining the logs of each transaction, and writing them onto some stable storage before actually modifying the db.

ii) Maintaining shadow paging, where are the changes are done on a volatile memory and later the actual db is updated.

Log-Based Recovery

log is a sequence of records, which maintains the records of actions performed by a transaction.

Log-Based recovery works as follows:

→ The log file is kept on stable storage media.

→ when a transaction enters the system and starts execution, it writes a log about it $< T_n, start >$.

→ when the transaction modifies an item x, it writes logs as follows: $< T_n, X, V1, V2 >$

It reads T_n has changed the value of x, from V_1 to V_2

→ when transaction finishes, it logs: $< T_n$ Commit$>$

Database can be modified using two approaches:

1) Defered database modification: All logs are written on to the stable storage & db is updated when transaction commits.

2) Immediate database modification: Each log follows an actual db modification. That is, db is modified immediately after every operation.

Buffer Management

Db buffers are generally implemented in virtual memory in spite of some drawbacks:

a) when operating system needs to evict a page that has been modified, the page is written to swap space on disk.

b) when db decides to write buffer page to disk, buffer page may be in swap space, and may have to be read from swap space on disk & o/p to the db on disk, resulting in extra I/O. (known as dual paging problem).

c) Ideally when os needs to evict a page from the buffer, it should pass control to db, which in turn should

→ o/p the page to db instead of to swap space, if it is modified.

→ Release the page from the buffer, for the os to use.

failure with loss of Non-Volatile storage

→ All transaction, which are being executed are kept in main memory. All active logs, disk buffers and related data is stored in non-volatile storage.

→ when storage like RAM fails, it takes away all the logs and active copy of db. It ~~may~~ makes recovery almost impossible as everything to help recover is also lost.

following techniques may be adopted in case of loss of non-volatile storage:

i) A mechanism like checkpoint can be adopted which makes the entire content of db be saved periodically.

ii) State of active db in non-volatile memory can be dumped onto stable storage periodically, which may also contains logs and active transactions and buffer blocks.

iii) <dump> can be marked on log file whenever the db contents are dumped from non-volatile memory to a stable one.

Recovery Algorithm

→ Consider transaction T_i that transfers $50 from account A to account B.

→ Two updates: Subtract 50 from A and adds 50 to B.

→ Transaction T_i requires updates to A and B to be output to the db.

→ A failure may occur after one of these modifications have been made but before both of them are made.

- Modifying the db without ensuring that the transaction will commit may leave the db in an inconsistent state.
- Not modifying the db may result in lost updates if failure occurs just after transaction commits.

→ Recovery Algorithms have 2 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 db contents to a state that ensures atomicity, consistency and durability.

Early lock Release and logical undo operations:

→ Support for high-concurrency locking techniques, such as those used for B^+ tree concurrency control, which release locks early
- supports "logical undo".

→ Recovery based on "repeating history", whereby recovery executes exactly the same actions as normal processing.

→ operations like B^+ tree insertions and deletions release locks early.
- They cannot be undone by restoring old values (physical undo), since once a lock is released, other transactions may have updated the B^+ tree.

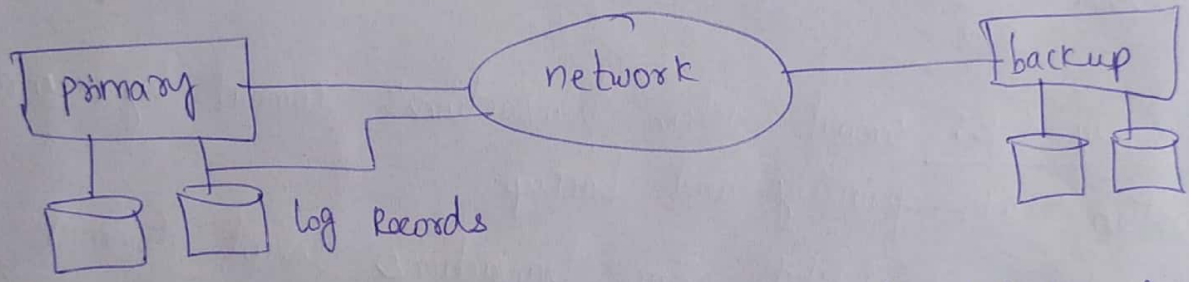
Contd.

→ Instead, insertions are undone by executing a deletion operation (known as logical undo).

→ for such operations, undo log records should contain the undo operation to be executed — such logging is called logical undo logging.

Remote Backup Systems

→ Remote Backup systems provide high availability by allowing transaction processing to continue even if the primary site is destroyed.



→ Detection of failure : Backup site must detect when primary site has failed.

→ Transfer of Control : - To take over control backup site first perform recovery using its copy of the db and all the log records it has received from the primary.
- when the backup site takes over processing it becomes the new primary.
- To transfer control back to old primary when it recovers old primary must receive redo logs from the old backup & apply all updates locally.

→ Time to Recover: To reduce delay in takeover, backup site periodically processes the redo log records performs a checkpoint, and can then delete earlier parts of the log.

→ Hot-Spare configuration permits very fast takeover.

→ Ensure durability of updates by delaying transaction commit until update is logged at backup; avoid this delay by permitting lower degrees of durability.

→ One-Safe: Commit as soon as transaction's commit log record is written at primary.
- problem: updates may not arrive at backup before it takes over.

→ Two-Very-Safe: Commit when transaction's commit log record is written at primary and backup.
- Reduces availability since transactions cannot commit if either site fails.

→ Two-Safe: proceed as in two-very-safe if both primary & backup are active. If only the primary is active, the transaction commits as soon as is commit log record is written at the primary.
- Better availability than two-very-safe; avoids problems of lost transactions in one-safe.