LECTURE 4

BUFFER MANAGEMENT (1/6)

- Log-record Buffering
  - Every log record is output to the stable storage at the time it is created
  - High overhead
  - Output to stable storage is in units of blocks
  - **Log record is much smaller than a block**

BUFFER MANAGEMENT (2/6)

- Cost of outputting a block to stable storage is sufficiently high that it is desirable to output multiple log records at once
- Hence, **write log records to log buffer in main memory**, where they **stay temporarily until they are output to stable storage**
- Multiple log records can be gathered in log buffer, and **output to stable storage** in a **single output operation**

BUFFER MANAGEMENT (3/6)

- **Order of log records** in stable storage must be **exactly same** as order in which they were written to log buffer

- Since log records are lost if system crashes, additional requirements on recovery techniques must be imposed
  - T_i enters commit state after <T_i commit> log record has been output to stable storage
  - Before <T_i commit> log record can be output to stable storage, all log records pertaining to transaction T_i must have been output to stable storage

BUFFER MANAGEMENT (4/6)

- Before a block of data in main memory can be output to database, all log records pertaining to data in that block must have been output to stable storage

- This rule is called **write-ahead logging (WAL) rule**

- When system finds it necessary to output a log record to stable storage, it outputs entire block of log records, if there are enough log records in main memory to fill a block

BUFFER MANAGEMENT (5/6)

- If not enough, all log records in main memory are combined into a partially full block, and are output to stable storage
- **Writing the buffered log to disk is sometimes referred to as log force**

LOGICAL UNDO LOGGING (1/6)

- For operations that locks are released early, we cannot perform the undo actions by simply writing back the old value of the data items
- Suppose we have transaction T that inserts an entry into a B^+ tree, and following the B^+ tree concurrency-control protocol, releases some locks after the insertion operation completes, but before transaction commits
- After the locks are released, other transactions may perform further insertions and deletions, thereby causing further changes to B^+ tree nodes

LOGICAL UNDO LOGGING (2/6)

- Even though operation release some locks early, it must retain enough locks to ensure that no other transactions is allowed to execute any conflicting operation
- For this reason, the B^+ tree concurrency-control protocol **holds locks on the leaf level of the B^+ tree until the end of the transaction**
- Consider how to perform transaction rollback

LOGICAL UNDO LOGGING (3/6)

- If **physical undo** is used, i.e. **the old values of the internal B^+ tree nodes** (before insertion operation was executed) **are written back during transaction rollback**, some of the updates performed by later insertion or deletion operations executed by other transactions could be lost
- Instead, **insertion has to be undone by a logical undo** i.e. **execution of a delete operation**
- When insertion operation completes, before it releases any locks, it writes a log record **$<T_i, O_j,$ operation-end, U>**, where **U** denotes **undo information** and O_j denotes a **unique identifier** for the operation

LOGICAL UNDO LOGGING (4/6)

- Eg. If operation inserted an entry in a B^+ tree, the undo information *U would indicate that a deletion operation is to be performed*, and would identify the B^+ tree and what to delete from the tree
- These type of logging of information about operations is called **logical logging**
- In contrast, **logging of old-value and new-value information is called physical logging**, and corresponding log records are called **physical log records**

LOGICAL UNDO LOGGING (5/6)

- **Insertion and deletion operations** are examples of a class of operations that require logical undo operations since they release locks early
- These operations are called **logical operations**
- Before logical operation begins, it writes a log record **<T_i, O_j, operation-begin>**, where O_j is unique identifier for the operation

LOGICAL UNDO LOGGING (6/6)

- While system is executing the operation, it does physical logging in the normal fashion for all updates performed by the operation
- The usual old-value and new-value information is written out for each update
- When operation finishes, it writes an **operation-end** log record

FUZZY CHECKPOINTING (1/4)

- A normal checkpoint requires that all updates to database be temporarily suspended while checkpointing is in progress
- If number of pages in buffer is large, a checkpoint may take long time to finish, which can result in an unacceptable interruption in processing of transactions
- To avoid such interruptions, the checkpointing technique can be modified to **permit updates to start once the checkpoint record has been written**, **but before modified buffer blocks are written to disk**

FUZZY CHECKPOINTING (2/4)

- The checkpoint thus generated is a **fuzzy checkpoint**
- Since pages are output to disk only after the checkpoint record has been written, it is possible that the system could crash before all pages are written
- Thus, a checkpoint on disk may be incomplete
- **Last-checkpoint** - The location in the log of the last completed checkpoint is stored in a fixed position

FUZZY CHECKPOINTING (3/4)

- System does not update this information when it writes the checkpoint record
- Instead, before it writes the checkpoint record, it creates a list of all modified buffer blocks
- The last-checkpoint information is updated only after all buffer blocks in the list of modified buffer blocks have been output to disk

FUZZY CHECKPOINTING (4/4)

- Even with fuzzy checkpointing, a **buffer block must not be updated while it is being output to disk**, although **other buffer blocks may be updated concurrently**

- The **write-ahead log protocol** must be followed so that log records pertaining to a block are on stable storage before block is output

RESTART RECOVERY

- **SELF STUDY**

- There will be two phase
  - Undo phase
  - Redo phase