443 Database Management Systems
Fall 2014 PL3

g1mihai, Mihai Nicolae - 998584367
g1dameng, Qingsong Meng - 998383093
g8buihuy, Huy Bui - 996594421

1. B+tree Concurrency Control

(a) A number of special locking protocols have been proposed for B+ trees. What problem do they address (e.g., why consider special locking protocols for B+ tree)?

Locks are proposed to solve the problem of contention when a large number of transactions try to search and modify a B+ tree simultaneously. In particular, it prevents the phantom problem and preserve serializability.

(b) Would the same problem be better or worse if the system used an optimistic concurrency control method rather than locking? Explain your answer.

Optimistic concurrency control assumes that contention is rare. The problem will be worse if there are a lot of transactions that try to modify the same object. In this case, these transactions will be repeatedly restarted and thus will degrade performance. However, the locking approach, say strict 2PL, also decreases throughput since the competing transactions will have to wait for locks on the common objects that they want to be released. The choice of concurrency control is determined by the contention rate.

(c) Describe one B+tree locking protocol other than the one presented in 17.5.2 in the textbook.

Section 17.5.2 describes a protocol with lock-coupling using S lock for read transactions and lock-upgrading for insert/delete transactions. Another simple way to lock B+ tree is to use separate locks for read and modify transactions. Specifically, for read transactions, we use lock-coupling with S lock. For inserts and deletes, we use X locks.

(d) Consider an index that is implemented with linear hashing rather than by a B+ tree. Would you need a special purpose locking protocol for this data structure? If so, outline such a locking protocol. If not, explain why not.

Yes. A linear hashing index still needs concurrency control. If two transactions are hashed into the same bucket, we lock the bucket. If the records are in one or more overflow pages, we lock all related overflow pages before modifying records.

2. This question relates to the design of a transaction manager.
(a) Specify the design (data structures and algorithms) of a 2PL lock manager with deadlock detection that implements the full S/X/IS/IX/SIX hierarchical locking protocol. Since it is extremely important to be able to set and release locks very quickly, your design should be as efficient as possible. In particular, a transaction should be able to release all of its locks with a single call to the lock manager. (Be sure to illustrate your data structure design via a small example.)

For deadlock detection, use a waits-for graph. Nodes represent active transactions and there exists a directed edge from T_i to T_j if T_i is waiting for T_j to release a lock. When locks requests are queued the lock manager adds edges to this graph. Similarly, when lock requests are granted the lock manager removes edges from this graph. The lock manager periodically checks for cycles, where a cycle indicates a deadlock. Because deadlocks are rare in practice and typically involve very few transactions it as acceptable to use this approach and not deadlock prevention. A deadlock is resolved by aborting a transaction. The transaction to be aborted is chosen is the one that has the least amount of work.

The database has a hierarchical structure, from highest to lowest in the structure we have database, tables, files, records. Each element in the structure is modelled as a node in a DAG. For each node, we maintain the a queue of lock requests. For each element of the queue we maintain the transactionID associated with it. Lock requests are added to the queue in first-in first-out (FIFO) order. There are 6 types of locks: IS, SIX, IX, S, X. Locks are acquired top-to-bottom from the highest node, and released from bottom-to-top. The last acquired node is of type S or X.

When a lock on the object is released we grant the next lock request in the queue, sending a signal to the associated transaction. We then iterate over the next elements of the queue and grant the locks that are compatible and stop at the first lock that is not compatible with the currently granted locks.

When a transaction completes it calls an interface of the lock manager to release all its associated locks. The lock manager then takes care of releasing the locks in a bottom-to-top fashion as described above. This abstracts away the details of lock management for the transaction.

Lastly, regarding handling the phantom problem. The phantom problem is completely avoided using this design. If the transaction uses an index search key, the data entry pertaining to the query is locked. If the transaction does not use an index search key, the entire table is locked.

(b) What extensions/changes would be necessary to extend this lock manager for use in a distributed DBMS?

In a distributed database, data is distributed and can be replicated across multiple physical locations. A transaction most likely has to be executed in multiple database nodes. Thus we need a coordinated effort to make this happen. In particular, a database node must communicate its intended transactions with a centralized transaction manager. The transaction manager will coordinate this intent with the local transaction managers (at these nodes). The coordinator can broadcast the transaction detail to other database nodes. The local transaction managers (at these nodes) will decide whether the transaction can be done and inform the coordinator. If it receives an "agree" message from all nodes, it will issue a unique global transaction ID and broadcast this ID in a "go ahead" message. Upon receiving this message, the local nodes will attach this global ID to all log records related to to the the transaction. A log record will have both a local LSN and this global ID.

For crash recovery purposes, the central transaction manager is synced across a coordinator node and a backup node. The backup node is periodically synced with the coordinator node but does not make any decisions on behalf of the coordinator node. If the coordinator node crashes, the backup node takes over as the central transaction manager. This minimizes the impact of the crash. The backup node may not have all the information up to date because of the periodical synchronization with the coordinator node, but this is better than continuous synchronization given that crashes occur rarely. Lastly, when the former coordinator node is back up, it becomes the new backup node.

3. Consider the Aries' page-oriented logging scheme for recovery where data is updated in place. In this question, you are asked to explain checkpointing and why a log, by itself, is not sufficient to make recovery work.

(a) Briefly explain why checkpoints are needed in Aries.

Checkpointing helps reduce the time needed to recover from a crash. A checkpoint is created, the database consistency is guaranteed up to that point. Thus when a crash happens, the recovery process only needs to start examining log records from the last checkpoint and not from the beginning of the log.

(b) Describe two possible checkpointing strategies. One should be optimized for an environment where failures are rare, and the other for an environment where failures are common. For each strategy, explain what should be logged, what should be forced to disk, and when, both during normal system operation and at checkpoint time.

In an environment where crashes happen frequently, the recovery process should take place faster. The checkpointing algorithm can assist this by:

1. During normal system operation, increase the frequency of periodic dirty page forcing.
2. At checkpoint time, force all dirty pages to disk before creating a checkpoint. This helps reducing the size of the dirty page table at recovery time.

In an environment where crashes happen rarely, we will use the no-force, steal approach of ARIES. In such a context,, recovery preparations can be minimized by reducing the frequency of checkpoints. The recovery process takes longer, but that is acceptable since crashes happen rarely.