CHAPTER 16

Recovery System

This chapter covers failure models and a variety of failure recovery techniques. Recovery in a real-life database systems supporting concurrent transactions is rather complicated. To help the student understand concepts better, the chapter presents recovery models in increasing degree of complexity.

The coverage of recovery in the 6th edition has changed significantly from the previous editions. In particular, the number of different algorithms presented has been reduced from earlier, avoiding the confusion between algorithms that perform undo first, then redo, versus algorithms that perform redo first, then undo; we have standardized on the latter in this edition.

Coverage of recovery algorithms now follows the sequence below:

- In Section 16.3 we describe a number of key concepts in recovery, before presenting the basic recovery algorithm in Section 16.4; the algorithm in this section supports concurrency with strict two-phase locking.
- Sections 16.5 and 16.6 discuss how to extend the recovery algorithm to deal with issues in buffer management and failure of non-volatile storage.
- Section 16.7 discusses how to handle early lock release by using logical undo operations; the algorithm described here was referred to as the "Advanced Recovery Algorithm" in the 5th edition. The coverage here has been extended, with more examples.
- Section 16.8 presents the ARIES recovery algorithm, variants of which are widely used in practice. ARIES includes some support for early lock release, along with several optimizations that speed up recovery greatly.

Exercises

16.12 Explain the difference between the three storage types—volatile, non-volatile, and stable—in terms of I/O cost.

Answer: Volatile storage is storage which fails when there is a power failure. Cache, main memory, and registers are examples of volatile storage.

Non-volatile storage is storage which retains its content despite power failures. An example is magnetic disk. Stable storage is storage which theoretically survives any kind of failure (short of a complete disaster!). This type of storage can only be approximated by replicating data.

In terms of I/O cost, volatile memory is the fastest and non-volatile storage is typically several times slower. Stable storage is slower than non-volatile storage because of the cost of data replication.

16.13 Stable storage cannot be implemented.

a. Explain why it cannot be.

b. Explain how database systems deal with this problem.

Answer:

a. Stable storage cannot really be implemented because all storage devices are made of hardware, and all hardware is vulnerable to mechanical or electronic device failures.

b. Database systems approximate stable storage by writing data to multiple storage devices simultaneously. Even if one of the devices crashes, the data will still be available on a different device. Thus data loss becomes extremely unlikely.

16.14 Explain how the database may become inconsistent if some log records pertaining to a block are not output to stable storage before the block is output to disk.

Answer: Consider a banking scheme and a transaction which transfers \$50 from account A to account B. The transaction has the following steps:

a. **read**(A,a_1)

b. $a_1 := a_1 - 50$

c. **write**(A,a_1)

d. **read**(B,b_1)

e. $b_1 := b_1 + 50$

f. **write**(B,b_1)

Suppose the system crashes after the transaction commits, but before its log records are flushed to stable storage. Further assume that at the time of the crash the update of A in the third step alone had actually been propagated to disk whereas the buffer page containing B was not yet written to disk. When the system comes up it is in an inconsistent state, but recovery is not possible because there are no log records corresponding to this transaction in stable storage.

16.15 Outline the drawbacks of the no-steal and force buffer management policies.

Answer: Drawback of the no-steal policy: The no-steal policy does not work with transactions that perform a large number of updates, since the buffer may get filled with updated pages that cannot be evicted to disk, and the transaction cannot then proceed.
Drawback of the force policy: The force policy might slow down the commit of transactions as it forces all modified blocks to be flushed to disk before commit. Also, the number of output operations is more in the case of force policy. This is because frequently updated blocks are output multiple times, once for each transaction. Further, the policy results in more seeks, since the blocks written out are not likely to be consecutive on disk.

16.16 Physiological redo logging can reduce logging overheads significantly, especially with a slotted page record organization. Explain why.
Answer: If a slotted page record organization is used, the deletion of a record from a page may result in many other records in the page being shifted. With physical redo logging, all bytes of the page affected by the shifting of records must be logged. On the other hand, if physiological redo logging is used, only the deletion operation is logged. This results in a much smaller log record. Redo of the deletion operation would delete the record and shift other records as required.

16.17 Explain why logical undo logging is used widely, whereas logical redo logging (other than physiological redo logging) is rarely used.
Answer: The class of operations which release locks early are called logical operations. Once such lower level locks are released, such operations cannot be undone by jusing the old values of updated data items. Instead, they must be undone by executing a compensating operation called a logical undo operation. In order to allow logical undo of operations, special log records are necessary to store the necessary logical undo information. Thus logical undo logging is used widely.
Redo operations are performed exclusively using physical log records. This is because the state of the database after a system failure may reflect some updates of an operation and not of other operations, depending on what buffer blocks had been written to disk before the failure. The database state on disk might not be in an *operation consistent* state, i.e., it might have partial effects of operations. Logical undo or redo operations cannot be performed on an inconsistent data structure.

16.18 Consider the log in Figure 16.5. Suppose there is a crash just before the $< T_0$ abort$>$ log record is written out. Explain what would happen during recovery.
Answer: Recovery would happen as follows:
Redo phase:

a. Undo-List = T_0, T_1

b. Start from the checkpoint entry and perform the redo operation.

c. C = 600

d. T_1 is removed from the Undo-list as there is a commit record.

e. T_2 is added to the Undo list on encountering the $< T_2$ start $>$ record.

f. A = 400

g. B = 2000

Undo phase:

a. Undo-List = T_0, T_2

b. Scan the log backwards from the end.

c. A = 500; output the redo-only record $< T_2$, A, 500 $>$

d. output $< T_2$ abort $>$

e. B = 2000; output the redo-only record $< T_0$, B, 2000 $>$

f. output $< T_0$ abort $>$

At the end of the recovery process, the state of the system is as follows:

A = 500
B = 2000
C = 600

The log records added during recovery are:

$< T_2$, A, 500 $>$
$< T_2$ abort $>$
$< T_0$, B, 2000 $>$
$< T_0$ abort $>$

Observe that B is set to 2000 by two log records, one created during normal rollback of T_0, and the other created during recovery, when the abort of T_0 is completed. Clearly the second one is redundant, although not incorrect. Optimizations described in the ARIES algorithm (and equivalent optimizations described in Section 16.7 for the case of logical operations) can help avoid carrying out redundant operations, which create such redundant log records.

16.19 Suppose there is a transaction that has been running for a very long time, but has performed very few updates.

a. What effect would the transaction have on recovery time with the recovery algorithm of Section 16.4, and with the ARIES recovery algorithm.

b. What effect would the transaction have on deletion of old log records?

Answer:

a. If a transaction has been running for a very long time, with few updates, it means that during recovery, the undo phase will have to scan the log backwards till the beginning of this transaction. This will increase the recovery time in the case of the recovery algorithm of Section 16.4. However, in the case of ARIES the effect is not that bad as ARIES considers the LastLSN and PrevLSN values of transactions in the undo list during its backward scan, allowing it to skip intermediate records belonging to completed transactions.

b. A long running transaction implies that no log records which are written after it started, can be deleted till it either commits or aborts. This might lead to a very large log file being generated, though most of the transactions in the log file have completed. This transaction becomes a bottleneck for deletion of old log records.

16.20 Consider the log in Figure 16.6. Suppose there is a crash during recovery, just before the operation abort log record is written for operation O_1. Explain what would happen when the system recovers again.

Answer: **Errata note:** The question above in the book has the text ".. just before after the operation abort"; the word "after" should be deleted from the text.

There is no checkpoint in the log, so recovery starts from the beginning of the log, and replays each action that is found in the log.

The redo phase would add the following log records:

$< T_0$, B, 2050 $>$
$< T_0$, C, 600 $>$
$< T_1$, C, 400 $>$
$< T_0$, C, 500 $>$

At the end of the redo phase, the undo list contains transactions T_0 and T_1, since their start records are found, but not their end of abort records.

During the undo phase, scanning backwards in the log, the following events happen:

$< T_0$, C, 400 $>$
$< T_1$, C, 600 $>$ /* The operation end of $T_1.O_2$ is found, */
/* and logical undo adds +200 to the current value of C. */
/* Other log records of T_1 are skipped till $T_1.O_2$ operation */
/* begin is found. Log records of other txns would be */
/* processed, but there are none here. */
$< T_1, O_2$, operation-abort $>$ /* On finding $T_1.O_2$ operation begin */

< T_1, abort > /* On finding T_1 start */
/* Next, the operation end of $T_0.O_1$ is found, and */
/* logical undo adds +100 to the current value of C.*/
< T_0, C, 700 >
/* Other operations of T_1 till $T_0.O_1$ begin are skipped */
/* And when $T_0.O_1$ operation begin is found: */
< T_0, O_1, operation-abort >
< T_0, B, 2000 >
< T_0, abort >

Finally the values of data items B and C would be 2000, and 700, which, which were their original values before T_0 or T_1 started.

16.21 Compare log-based recovery with the shadow-copy scheme in terms of their overheads, for the case when data is being added to newly allocated disk pages (in other words, there is no old value to be restored in case the transaction aborts).

Answer: In general, with logging each byte that is written is actually written twice, once as part of the page to which it is written, and once as part of the log record for the update. In contrast, shadow-copy schemes avoids log writes, at the expense of increased IO operations, and lower concurrency.

In the case of data added to newly allocated pages:

a. There is no old value, so there is no need to manage a shadow copy for recovery, which benefits the shadow-copy scheme.

b. Log-based recovery system would unnecessarily write old value, unless optimizations to skip the old value part are implemented for such newly allocated pages. Without such an optimization, logging has a higher overhead.

c. Newly allocated pages are often formatted, for example by zeroing out all bytes, and then setting bytes corresponding to data structures in a slotted-page architecture. These operations also have to be logged, adding to the overhead of logging.

d. However, the benefit of shadow-copy over logging is really significant only if the page is filled with a significant amount of data before it is written, since a lot of logging is avoided in this case.

16.22 In the ARIES recovery algorithm:

a. If at the beginning of the analysis pass, a page is not in the checkpoint dirty page table, will we need to apply any redo records to it? Why?

b. What is RecLSN, and how is it used to minimize unnecessary redos?

Answer:

a. If a page is not in the checkpoint dirty page table at the beginning of the analysis pass, redo records prior to the checkpoint record need not be applied to it as it means that the page has been flushed to disk and been removed from the DirtyPageTable before the checkpoint. However, the page may have been updated after the checkpoint, which means it will appear in the dirty page table at the end of the analysis pass.
For pages that appear in the checkpoint dirty page table, redo records prior to the checkpoint may also need to be applied.

b. The RecLSN is an entry in the DirtyPageTable, which reflects the LSN at the end of the log when the page was added to DirtyPageTable. During the redo pass of the ARIES algorithm, if the LSN of the update log record encountered, is less than the RecLSN of the page in DirtyPageTable, then that record is not redone but skipped. Further, the redo pass starts at RedoLSN, which is the earliest of the RecLSNs among the entries in the checkpoint DirtyPageTable, since earlier log records would certainly not need to be redone. (If there are no dirty pages in the checkpoint, the RedoLSN is set to the LSN of the checkpoint log record.)

16.23 Explain the difference between a system crash and a "disaster."

Answer: In a system crash, the CPU goes down, and disk may also crash. But stable-storage at the site is assumed to survive system crashes. In a "disaster", *everything* at a site is destroyed. Stable storage needs to be distributed to survive disasters.

16.24 For each of the following requirements, identify the best choice of degree of durability in a remote backup system:

a. Data loss must be avoided but some loss of availability may be tolerated.

b. Transaction commit must be accomplished quickly, even at the cost of loss of some committed transactions in a disaster.

c. A high degree of availability and durability is required, but a longer running time for the transaction commit protocol is acceptable.

Answer:

a. Two very safe is suitable here because it guarantees durability of updates by committed transactions, though it can proceed only if both primary and backup sites are up. Availability is low, but it is mentioned that this is acceptable.

b. One safe committing is fast as it does not have to wait for the logs to reach the backup site. Since data loss can be tolerated, this is the best option.

c. With two safe committing, the probability of data loss is quite low, and also commits can proceed as long as at least the primary site is up. Thus availability is high. Commits take more time than in the one safe protocol, but that is mentioned as acceptable.

16.25 The Oracle database system uses undo log records to provide a snapshot view of the database, under snapshot-isolation. The snapshot view seen by transaction T_i reflects updates of all transactions that had committed when T_i started, and the updates of T_i; updates of all other transactions are not visible to T_i.

Describe a scheme for buffer handling whereby transactions are given a snapshot view of pages in the buffer. Include details of how to use the log to generate the snapshot view. You can assume that operations as well as their undo actions affect only one page.

Answer: First, determine if a transaction is currently modifying the buffer. If not, then return the current contents of the buffer. Otherwise, examine the records in the undo log pertaining to this buffer. Make a copy of the buffer, then for each relevant operation in the undo log, apply the operation to the buffer copy starting with the most recent operation and working backwards until the point at which the modifying transaction began. Finally, return the buffer copy as the snapshot buffer.