Database Management System (DBMS)

Lecture-46

Dharmendra Kumar

November 19, 2022

Log-Based Recovery

Log-Based Recovery

Log

The log is a sequence of log records, recording all the update activities in the database. There are several types of log records. An update log record describes a single database write. It has these fields:

- **Transaction identifier** is the unique identifier of the transaction that performed the write operation.
- **Data-item identifier** is the unique identifier of the data item written. Typically, it is the location on disk of the data item.
- **Old value** is the value of the data item prior to the write.
- **New value** is the value that the data item will have after the write.

Types of Log Records

- $< T_i \ \ start >$. Transaction T_i has started.
- $< T_i, \ X_j, \ V_1, \ V_2 >$. Transaction T_i has performed a write on data item X_j . X_j had value V_1 before the write, and will have value V_2 after the write.
- $< T_i \ \ commit >$. Transaction T_i has committed.
- $< T_i \ \ abort >$. Transaction T_i has aborted.

Note: For log records to be useful for recovery from system and disk failures, the log must reside in stable storage.

Type of Log-based Recovery Techniques

There are two techniques for using the log to ensure transaction atomicity despite failures.

1. Deferred Database Modification
2. Immediate Database Modification

Deferred Database Modification

- The deferred-modification technique ensures transaction atomicity by recording all database modifications in the log, but deferring the execution of all write operations of a transaction until the transaction partially commits.

- When a transaction partially commits, the information on the log associated with the transaction is used in executing the deferred writes. If the system crashes before the transaction completes its execution, or if the transaction aborts, then the information on the log is simply ignored.

Log-Based Recovery

The execution of transaction T_i proceeds as follows.

- Before T_i starts its execution, a record $< T_i \ \ start >$ is written to the log.
- A write(X) operation by T_i results in the writing of a new record to the log.
- Finally, when T_i partially commits, a record $< T_i \ \ commit >$ is written to the log.

Note: Here, only the new value of the data item is required by the deferred-modification technique.

Log-Based Recovery

Example: .

Let T_0 be a transaction that transfers \$50 from account A to account B:

T_0: read(A);

A := A 50;

write(A);

read(B);

B := B + 50;

write(B).

Let T_1 be a transaction that withdraws \$100 from account C:

T_1: read(C);

C := C 100;

write(C).

Log-Based Recovery

Suppose that these transactions are executed serially, in the order T_0 followed by T_1, and that the values of accounts A, B, and C before the execution took place were \$1000, \$2000, and \$700, respectively. The portion of the log containing the relevant information on these two transactions appears as the following:-

$< T_0 \;\; start >$
$< T_0,\; A,\; 950 >$
$< T_0,\; B,\; 2050 >$
$< T_0 \;\; commit >$
$< T_1 \;\; start >$
$< T_1,\; C,\; 600 >$
$< T_1 \;\; commit >$

Log-Based Recovery

There are various orders in which the actual outputs can take place to both the database system and the log as a result of the execution of T_0 and T_1. One such order is the following:-

Log	**Database**
$< T_0 \ \ \mathit{start} >$	
$< T_0,\ A,\ 950 >$	
$< T_0,\ B,\ 2050 >$	
	A $=$ 950
	B $=$ 2050
$< T_0 \ \ \mathit{commit} >$	
$< T_1 \ \ \mathit{start} >$	
$< T_1,\ C,\ 600 >$	
$< T_1 \ \ \mathit{commit} >$	
	C $=$ 600

Log-Based Recovery

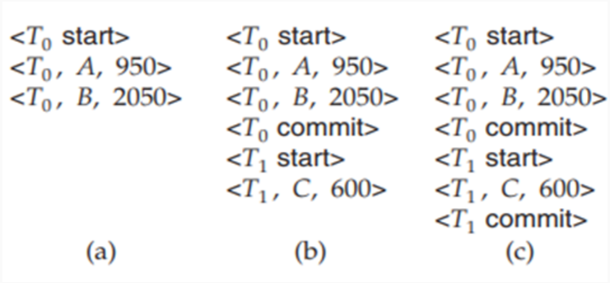
Using the log, the system can handle any failure that results in the loss of information on volatile storage. The recovery scheme uses the following recovery procedure:

- redo(T_i) sets the value of all data items updated by transaction T_i to the new values.

After a failure, the recovery subsystem consults the log to determine which transactions need to be redone. Transaction T_i needs to be redone if and only if the log contains both the record $< T_i \; start >$ and the record $< T_i \; commit >$.

Log-Based Recovery

Example: Consider above transactions T_0 and T_1 executed one after the other in the order T_0 followed by T_1. The log that results from the complete execution of T_0 and T_1 is shown above. Suppose that the system crashes before the completion of the transactions, so that we can see how the recovery technique restores the database to a consistent state.



<T_0 start>	<T_0 start>	<T_0 start>
<T_0, A, 950>	<T_0, A, 950>	<T_0, A, 950>
<T_0, B, 2050>	<T_0, B, 2050>	<T_0, B, 2050>
	<T_0 commit>	<T_0 commit>
	<T_1 start>	<T_1 start>
	<T_1, C, 600>	<T_1, C, 600>
		<T_1 commit>
(a)	(b)	(c)

Figure 1: The above log is shown at three different times.

Log-Based Recovery

(1) Assume that the crash occurs just after the log record for the step

write(B)

of transaction T_0 has been written to stable storage.

- The log at the time of the crash appears in Figure 1-a.
- When the system comes back up, no redo actions need to be taken, since no commit record appears in the log.
- The values of accounts A and B remain \$1000 and \$2000, respectively.
- The log records of the incomplete transaction T_0 can be deleted from the log.

Log-Based Recovery

(2) Now, let us assume the crash comes just after the log record for the step

write(C)

of transaction T_1 has been written to stable storage.

- In this case, the log at the time of the crash is as in Figure 1-b.
- When the system comes back up, the operation redo(T_0) is performed, since the record

  $$< T_0 \;\; commit >$$

  appears in the log on the disk.
- After this operation is executed, the values of accounts A and B are \$950 and \$2050, respectively. The value of account C remains \$700.

Log-Based Recovery

(3) Finally, assume that a crash occurs just after the log record

$$< T_1 \quad commit >$$

is written to stable storage.

- The log at the time of this crash is as in Figure 1-c.
- When the system comes back up, two commit records are in the log: one for T_0 and one for T_1. Therefore, the system must perform operations redo(T_0) and redo(T_1), in the order in which their commit records appear in the log.
- After the system executes these operations, the values of accounts A, B, and C are \$950, \$2050, and \$600, respectively.

Immediate Database Modification

- The immediate-modification technique allows database modifications to be output to the database while the transaction is still in the active state.

- In the event of a crash or a transaction failure, the system must use the old-value field of the log records to restore the modified data items to the value they had prior to the start of the transaction. The undo operation accomplishes this restoration.

Log-Based Recovery

- Before a transaction T_i starts its execution, the system writes the record $< T_i \;\; \mathit{start} >$ to the log.
- During its execution, any write(X) operation by T_i is preceded by the writing of the appropriate new update record to the log.
- When T_i partially commits, the system writes the record $< T_i \;\; \mathit{commit} >$ to the log.

Log-Based Recovery

Example: Consider transactions T_0 and T_1 used in previous topic, executed one after the other in the order T_0 followed by T_1. The portion of the log containing the relevant information concerning these two transactions appears as following:-

$< T_0 \ \ start >$
$< T_0, \ A, \ 1000, \ 950 >$
$< T_0, \ B, \ 2000, \ 2050 >$
$< T_0 \ \ commit >$
$< T_1 \ \ start >$
$< T_1, \ C, \ 700, \ 600 >$
$< T_1 \ \ commit >$

Log-Based Recovery

Following shows one possible order in which the actual outputs took place in both the database system and the log as a result of the execution of T_0 and T_1.

Log	**Database**
$< T_0 \;\; start >$	
$< T_0,\; A,\; 1000,\; 950 >$	
$< T_0,\; B,\; 2000,\; 2050 >$	
	A=950
	B=2050
$< T_0 \;\; commit >$	
$< T_1 \;\; start >$	
$< T_1,\; C,\; 700,\; 600 >$	
	C=600
$< T_1 \;\; commit >$	

Log-Based Recovery

Using the log, the system can handle any failure that does not result in the loss of information in nonvolatile storage. The recovery scheme uses two recovery procedures:

- undo(T_i) restores the value of all data items updated by transaction T_i to the old values.
- redo(T_i) sets the value of all data items updated by transaction T_i to the new values

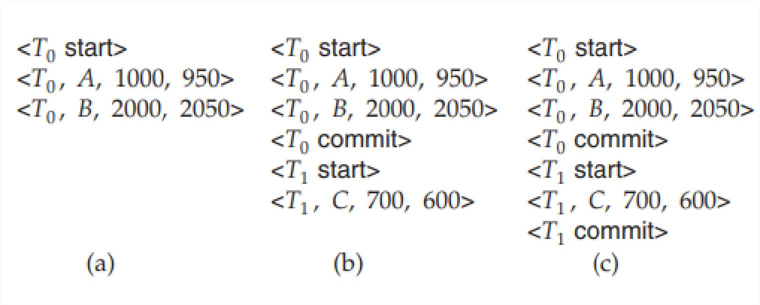
Log-Based Recovery

After a failure has occurred, the recovery scheme consults the log to determine which transactions need to be redone, and which need to be undone:

- Transaction T_i needs to be undone if the log contains the record $< T_i \ \ start >$, but does not contain the record $< T_i \ \ commit >$.
- Transaction T_i needs to be redone if the log contains both the record $< T_i \ \ start >$ and the record $< T_i \ \ commit >$.

Log-Based Recovery

Example: Consider transactions T_0 and T_1 executed one after the other in the order T_0 followed by T_1.Suppose that the system crashes before the completion of the transactions. We shall consider three cases. The state of the logs for each of these cases appears in Figure 2.



<T_0 start>	<T_0 start>	<T_0 start>
<T_0, A, 1000, 950>	<T_0, A, 1000, 950>	<T_0, A, 1000, 950>
<T_0, B, 2000, 2050>	<T_0, B, 2000, 2050>	<T_0, B, 2000, 2050>
	<T_0 commit>	<T_0 commit>
	<T_1 start>	<T_1 start>
	<T_1, C, 700, 600>	<T_1, C, 700, 600>
		<T_1 commit>
(a)	(b)	(c)

Figure 2: The same log, shown at three different times.

Log-Based Recovery

(1) First, let us assume that the crash occurs just after the log record for the step

write(B)

of transaction T_0 has been written to stable storage shown in Figure 2-a.

- When the system comes back up, it finds the record $< T_0 \;\; \mathit{start} >$ in the log, but no corresponding $< T_0 \;\; \mathit{commit} >$ record. Thus, transaction T_0 must be undone, so an undo(T_0) is performed.
- As a result, the values in accounts A and B (on the disk) are restored to \$1000 and \$2000, respectively.

Log-Based Recovery

(2) Next, let us assume that the crash comes just after the log record for the step

write(C)

of transaction T_1 has been written to stable storage shown in Figure 2-b.

- When the system comes back up, two recovery actions need to be taken. The operation undo(T_1) must be performed, since the record $< T_1 \;\; \textit{start} >$ appears in the log, but there is no record $< T_1 \;\; \textit{commit} >$.
- The operation redo(T_0) must be performed, since the log contains both the record $< T_0 \;\; \textit{start} >$ and the record $< T_0 \;\; \textit{commit} >$.
- At the end of the entire recovery procedure, the values of accounts A, B, and C are \$950, \$2050, and \$700, respectively. Note that the undo(T_1) operation is performed before the redo(T_0).

Log-Based Recovery

(3) Finally, let us assume that the crash occurs just after the log record

$$< T_1 \quad commit >$$

has been written to stable storage shown in Figure 2-c.

- When the system comes back up, both $T_0$and T_1 need to be redone, since the records $< T_0 \quad start >$ and $< T_0 \quad commit >$ appear in the log, as do the records $< T_1 \quad start >$ and $< T_1 \quad commit >$.
- After the system performs the recovery procedures redo(T_0) and redo(T_1), the values in accounts A, B, and C are $950, $2050, and $600, respectively.