Lecture 48
Log Based Recovery

Recovery System

- A computer system, like any other device, is subject to failure from a variety of causes : disk crash, power outage, software error, a fire in the machine room, even sabotage.
- In any failure, information may be lost. Therefore, the database system must take actions in advance to ensure that the atomicity and durability properties of transactions, are preserved.
- An integral part of a database system is a recovery scheme that can restore the database to the consistent state that existed before the failure.

Recovery System

- The recovery scheme must also support **high availability**, that is, the database should be usable for a very high percentage of time.
- To support high availability in the face of machine failure (as also planned machine shutdowns for hardware/software upgrades and maintenance), the recovery scheme must support the ability to keep a backup copy of the database synchronized with the current contents of the primary copy of the database.
- If the machine with the primary copy fails, transaction processing can continue on the backup copy.

Failure Classification

- There are various types of failure that may occur in a system, each of which needs to be dealt with in a different manner.
- We shall consider only the following types of failure :
  a) **Transaction failure** - There are two types of errors that may cause a transaction to fail :
     a) **Logical error** - The transaction can no longer continue with its normal execution because of some internal condition, such as bad input, data not found, overflow, or resource limit exceeded.
     b) **System error** - The system has entered an undesirable state (e.g., deadlock), as a result of which a transaction cannot continue with its normal execution. The transaction, however, can be reexecuted at a later time.

Failure Classification

c) **System crash -** There is a hardware malfunction, or a bug in the database software or the operating system, that causes the loss of the content of volatile storage and brings transaction processing to a halt. The content of non-volatile storage remains intact and is not corrupted.

d) **Disk failure -** A disk block loses its content as a result of either a head crash or failure during a data-transfer operation. Copies of the data on other disks, or archival backups on tertiary media, such as DVD or tapes, are used to recover from the failure.

Failure Classification

- To determine how the system should recover from failures, we need to identify the failure modes of those devices used for storing data.
- Next, we must consider how these failure modes affect the contents of the database.
- We can then propose algorithms to ensure database consistency and transaction atomicity despite failures.
- These algorithms, known as recovery algorithms, have two parts :
  - Actions taken during normal transaction processing to ensure that enough information exists to allow recovery from failures.
  - Actions taken after a failure to recover the database contents to a state that ensures database consistency, transaction atomicity, and durability.

Log Records

- The most widely used structure for recording database modifications is the 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:
  1. **Transaction identifier,** which is the unique identifier of the transaction that performed the write operation.
  2. **Data-item identifier,** which is the unique identifier of the data item written. Typically, it is the location on disk of the data item, consisting of the block identifier of the block on which the data item resides and an offset within the block.
  3. **Old value,** which is the value of the data item prior to the write.
  4. **New value,** which is the value that the data item will have after the write.

Log Records

- We represent an update log record as <*Ti*, *Xj*, *V*1, *V*2>, indicating that transaction *Ti* has performed a write on data item *Xj*. *Xj* had value *V*1 before the write and has value *V*2 after the write.
- Other special log records exist to record significant events during transaction processing, such as the start of a transaction and the commit or abort of a transaction.
- Among the types of log records are :
  1. **<*Ti* start>.** Transaction *Ti* has started.
  2. **<*Ti* commit>.** Transaction *Ti* has committed.
  3. **<*Ti* abort>.** Transaction *Ti* has aborted.

Log Records

- Whenever a transaction performs a write, it is essential that the log record for that write be created and added to the log, before the database is modified.
- Once a log record exists, we can output the modification to the database if that is desirable. Also, we have the ability to *undo* a modification that has already been output to the database. We undo it by using the old-value field in log records.
- For log records to be useful for recovery from system and disk failures, the log must reside in stable storage. For now, we assume that every log record is written to the end of the log on stable storage as soon as it is created.

Database Modification

- A transaction creates a log record prior to modifying the database.
- The log records allow the system to undo changes made by a transaction in the event that the transaction must be aborted; they allow the system also to redo changes made by a transaction if the transaction has committed but the system crashed before those changes could be stored in the database on disk.
- In order for us to understand the role of these log records in recovery, we need to consider the steps a transaction takes in modifying a data item :
  1. The transaction performs some computations in its own private part of main memory.
  2. The transaction modifies the data block in the disk buffer in main memory holding the data item.
  3. The database system executes the output operation that writes the data block to disk.

Database Modification

- We say a transaction ***modifies the database*** if it performs an update on a disk buffer, or on the disk itself; updates to the private part of main memory do not count as database modifications.
- If a transaction does not modify the database until it has committed, it is said to use the **deferred-modification technique**.
- If database modifications occur while the transaction is still active, the transaction is said to use the **immediate modification technique.**
- Deferred modification has the overhead that transactions need to make local copies of all updated data items; further, if a transaction reads a data item that it has updated, it must read the value from its local copy.

Database Modification

- A recovery algorithm must take into account a variety of factors, including:
  - The possibility that a transaction may have committed although some of its database modifications exist only in the disk buffer in main memory and not in the database on disk.
  - The possibility that a transaction may have modified the database while in the active state and, as a result of a subsequent failure, may need to abort.
- Because all database modifications must be preceded by the creation of a log record, the system has available both the old value prior to the modification of the data item and the new value that is to be written for the data item. This allows the system to perform *undo* and *redo* operations as appropriate.
  - The **undo** operation using a log record sets the data item specified in the log record to the old value contained in the log record.
  - The **redo** operation using a log record sets the data item specified in the log record to the new value contained in the log record.

Example

- Consider our simplified banking system.
- 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*).

Example : deferred-modification technique

T0	T1	Log	Database
Read(A)		<T0, start>	A=1000, B = 2000, C = 700
A : = A – 50;			
Write(A)		<T0, A, 1000, 950>	A=1000, B = 2000, C = 700
Read(B)			
B := B + 50;			
Write(B)		<T0, B, 2000, 2050>	A=1000, B = 2000, C = 700
Commit;		<T0, commit>	A=950, B = 2050, C = 700
	Read(C)	<T1, start>	A=950, B = 2050, C = 700
	C := C – 100;		
	Write(C)	<T1, C, 700, 600>	A=950, B = 2050, C = 700
	Commit;	<T1, commit>	A=950, B = 2050, C = 600

Example : deferred-modification technique

T0	T1	Log	Database
Read(A)		<T0, start>	A=1000, B = 2000, C = 700
A : = A – 50;			
Write(A)		<T0, A, 1000, 950>	A=1000, B = 2000, C = 700
Read(B)			
B := B + 50;			
Write(B)		<T0, B, 2000, 2050>	A=1000, B = 2000, C = 700

Recovery Procedure : Ignore the Log Records

Example : deferred-modification technique

T0	T1	Log	Database
Read(A)		<T0, start>	A=1000, B = 2000, C = 700
A : = A – 50;			
Write(A)		<T0, A, 1000, 950>	A=1000, B = 2000, C = 700
Read(B)			
B := B + 50;			
Write(B)		<T0, B, 2000, 2050>	A=1000, B = 2000, C = 700
Commit;		<T0, commit>	A=950, B = 2050, C = 700
	Read(C)	<T1, start>	A=950, B = 2050, C = 700
	C := C – 100;		
	Write(C)	<T1, C, 700, 600>	A=950, B = 2050, C = 700

Recovery Procedure : Ignore the Log Records of T1 and Perform Redo (T0)

Example : deferred-modification technique

T0	T1	Log	Database
Read(A)		<T0, start>	A=1000, B = 2000, C = 700
A : = A – 50;			
Write(A)		<T0, A, 1000, 950>	A=1000, B = 2000, C = 700
Read(B)			
B := B + 50;			
Write(B)		<T0, B, 2000, 2050>	A=1000, B = 2000, C = 700
Commit;		<T0, commit>	A=950, B = 2050, C = 700
	Read(C)	<T1, start>	A=950, B = 2050, C = 700
	C := C – 100;		
	Write(C)	<T1, C, 700, 600>	A=950, B = 2050, C = 700
	Commit;	<T1, commit>	A=950, B = 2050, C = 600

Recovery Procedure : Perform Redo(T1) as well as Redo (T0)

Example : Immediate-modification technique

T0	T1	Log	Database
Read(A)		<T0, start>	A=1000, B = 2000, C = 700
A : = A – 50;			
Write(A)		<T0, A, 1000, 950>	A=950, B = 2000, C = 700
Read(B)			
B := B + 50;			
Write(B)		<T0, B, 2000, 2050>	A=950, B = 2050, C = 700
Commit;		<T0, commit>	A=950, B = 2050, C = 700
	Read(C)	<T1, start>	A=950, B = 2050, C = 700
	C := C – 100;		
	Write(C)	<T1, C, 700, 600>	A=950, B = 2050, C = 600
	Commit;	<T1, commit>	A=950, B = 2050, C = 600

Example : Immediate-modification technique

T0	T1	Log	Database
Read(A)		<T0, start>	A=1000, B = 2000, C = 700
A : = A – 50;			
Write(A)		<T0, A, 1000, 950>	A=950, B = 2000, C = 700
Read(B)			
B := B + 50;			
Write(B)		<T0, B, 2000, 2050>	A=950, B = 2050, C = 700

Recovery Procedure : Perform Undo (T0)

Example : Immediate-modification technique

T0	T1	Log	Database
Read(A)		<T0, start>	A=1000, B = 2000, C = 700
A : = A – 50;			
Write(A)		<T0, A, 1000, 950>	A=950, B = 2000, C = 700
Read(B)			
B := B + 50;			
Write(B)		<T0, B, 2000, 2050>	A=950, B = 2050, C = 700
Commit;		<T0, commit>	A=950, B = 2050, C = 700
	Read(C)	<T1, start>	A=950, B = 2050, C = 700
	C := C – 100;		
	Write(C)	<T1, C, 700, 600>	A=950, B = 2050, C = 600

Recovery Procedure : Perform Undo (T1) and Redo (T0)

Example : Immediate-modification technique

T0	T1	Log	Database
Read(A)		<T0, start>	A=1000, B = 2000, C = 700
A : = A – 50;			
Write(A)		<T0, A, 1000, 950>	A=1000, B = 2000, C = 700
Read(B)			
B := B + 50;			
Write(B)		<T0, B, 2000, 2050>	A=1000, B = 2000, C = 700
Commit;		<T0, commit>	A=950, B = 2050, C = 700
	Read(C)	<T1, start>	A=950, B = 2050, C = 700
	C := C – 100;		
	Write(C)	<T1, C, 700, 600>	A=950, B = 2050, C = 700
	Commit;	<T1, commit>	A=950, B = 2050, C = 600

Recovery Procedure : Perform Redo (T1) and Redo (T0)

CheckPoints

- When a system crash occurs, we must consult the log to determine those transactions that need to be redone and those that need to be undone. In principle, we need to search the entire log to determine this information. There are two major difficulties with this approach:
  1. The search process is time-consuming.
  2. Most of the transactions that, according to our algorithm, need to be redone have already written their updates into the database. Although redoing them will cause no harm, it will nevertheless cause recovery to take longer.
- To reduce these types of overhead, we introduce checkpoints.

CheckPoints

- A checkpoint is performed as follows :
  1. Output onto stable storage all log records currently residing in main memory.
  2. Output to the disk all modified buffer blocks.
  3. Output onto stable storage a log record of the form <checkpoint *L*>, where *L* is a list of transactions active at the time of the checkpoint.
- Transactions are not allowed to perform any update actions, such as writing to a buffer block or writing a log record, while a checkpoint is in progress.

CheckPoints

- The presence of a <checkpoint *L*> record in the log allows the system to streamline its recovery procedure. Consider a transaction *Ti* that completed prior to the checkpoint.
- For such a transaction, the <*Ti* commit> record (or < *Ti* abort> record) appears in the log before the <checkpoint> record.
- Any database modifications made by *Ti* must have been written to the database either prior to the checkpoint or as part of the checkpoint itself.
- Thus, at recovery time, there is no need to perform a redo operation on *Ti*.

CheckPoints

- After a system crash has occurred, the system examines the log to find the last <checkpoint *L*> record (this can be done by searching the log backward, from the end of the log, until the first <checkpoint *L*> record is found).
- The redo or undo operations need to be applied only to transactions in *L*, and to all transactions that started execution after the <checkpoint *L*> record was written to the log.
- Let us denote this set of transactions as *T*.
  1. For all transactions *Tk* in *T* that have no <*Tk* commit> record or <*Tk* abort> record in the log, execute undo(*Tk*).
  2. For all transactions *Tk* in *T* such that either the record <*Tk* commit> or the record <*Tk* abort> appears in the log, execute redo(*Tk*).

CheckPoints

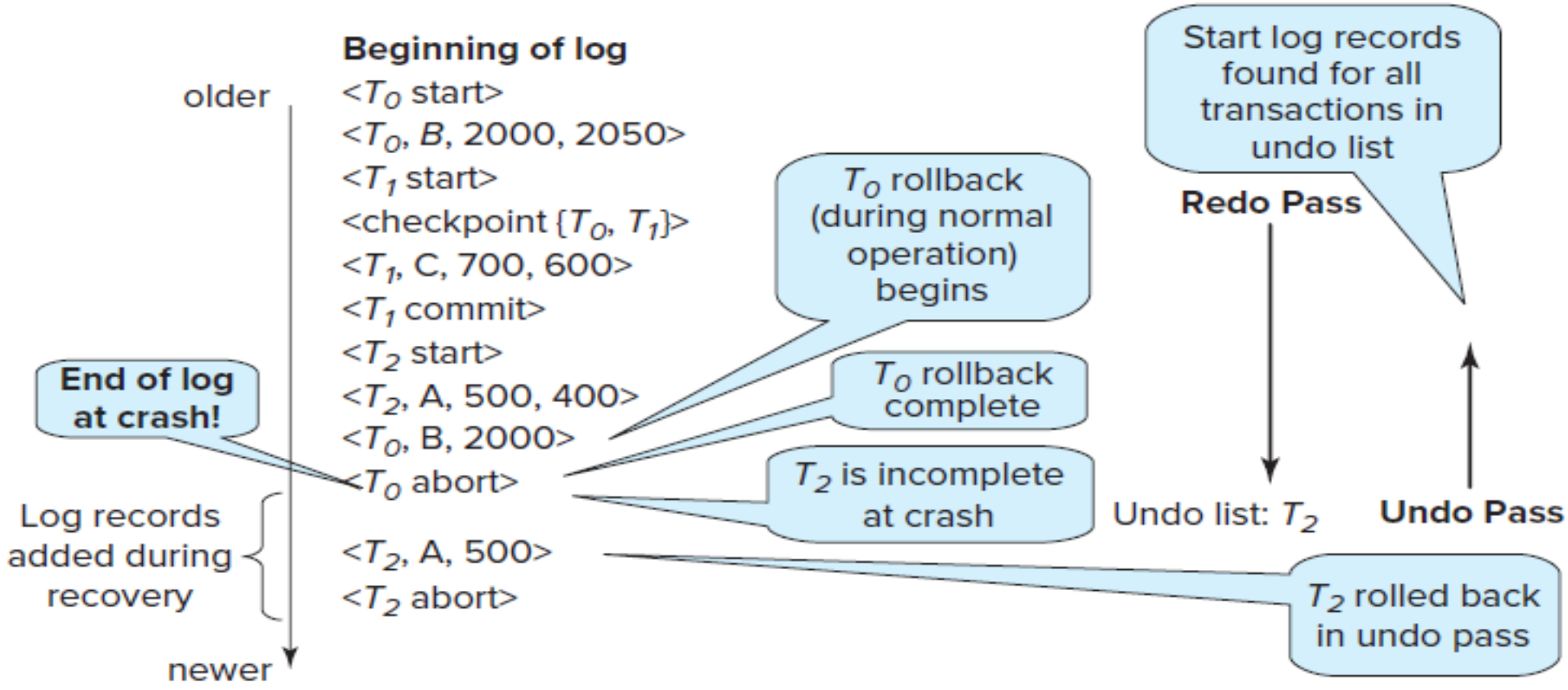

Figure 19.5 Example of logged actions and actions during recovery.

GATE Question

Consider the following log sequence of two transactions on a bank account, with initial balance 12000, that transfer 2000 to a mortgage payment and then apply a 5% interest.

1. T1 start
2. T1 B old=12000 new=10000
3. T1 M old=0 new=2000
4. T1 commit
5. T2 start
6. T2 B old=10000 new=10500
7. T2 commit

Suppose the database system crashes just before log record 7 is written. When the system is restarted, which one statement is true of the recovery procedure?

(A) We must redo log record 6 to set B to 10500

(B) We must undo log record 6 to set B to 10000 and then redo log records 2 and 3.

(C) We need not redo log records 2 and 3 because transaction T1 has committed.

(D) We can apply redo and undo operations in arbitrary order because they are idempotent

[GATE 2006]

GATE Question

Consider a simple checkpointing protocol and the following set of operations in the log.

(start, T4); (write, T4, y, 2, 3); (start, T1); (commit, T4); (write, T1, z, 5, 7);

(checkpoint);

(start, T2); (write, T2, x, 1, 9); (commit, T2); (start, T3); (write, T3, z, 7, 2);

If a crash happens now and the system tries to recover using both undo and redo operations, what are the contents of the undo list and the redo list?

(A) Undo: T3, T1; Redo: T2

(B) Undo: T3, T1; Redo: T2, T4

(C) Undo: none; Redo: T2, T4, T3, T1

(D) Undo: T3, T1, T4; Redo: T2

[GATE 2015]

For Video lecture on this topic please subscribe to my youtube channel.

The link for my youtube channel is

https://www.youtube.com/channel/UCRWGtE76JlTp1iim6aOTRuw?sub_confirmation=1