Chapter 6

Database Recovery Techniques

Adapted from the slides of “Fundamentals of Database Systems” (Elmasri et al., 2003)

Outline

Databases Recovery

1. Purpose of Database Recovery
2. Types of Failure
3. Transaction Log
4. Data Updates
5. Data Caching
6. Transaction Roll-back (Undo) and Roll-Forward
7. Checkpointing
8. Recovery schemes
9. ARIES Recovery Scheme
10. Recovery in Multidatabase System

Database Recovery

1 Purpose of Database Recovery

- To bring the database into the last consistent state, which existed prior to the failure.
- To preserve transaction properties (Atomicity, Consistency, Isolation and Durability).

Example: If the system crashes before a fund transfer transaction completes its execution, then either one or both accounts may have incorrect value. Thus, the database must be restored to the state before the transaction modified any of the accounts.

Database Recovery

2 Types of Failure

The database may become unavailable for use due to

- Transaction failure: Transactions may fail because of incorrect input, deadlock, incorrect synchronization.
- System failure: System may fail because of addressing error, application error, operating system fault, RAM failure, etc.
- Media failure: Disk head crash, power disruption, etc.

Database Recovery

3 Transaction Log

For recovery from any type of failure data values prior to modification (BFIM - BeFore Image) and the new value after modification (AFIM – AFter Image) are required. These values and other information are stored in a sequential file called ***Transaction log***. A sample log is given below. **Back P** and **Next P** point to the previous and next log records of the same transaction.

T ID	Back P	Next P	Operation	Data item	BFIM	AFIM
T1	0	1	Begin			
T1	1	4	Write	X	X = 100	X = 200
T2	0	8	Begin			
T1	2	5	W	Y	Y = 50	Y = 100
T1	4	7	R	M	M = 200	M = 200
T3	0	9	R	N	N = 400	N = 400
T1	5	nil	End			

Database Recovery

4 Data Update

- **Immediate Update**: As soon as a data item is modified in cache, the disk copy is updated.
- **Deferred Update**: All modified data items in the cache are written either after a transaction ends its execution or after a fixed number of transactions have completed their execution.
- **Shadow update**: The modified version of a data item does not overwrite its disk copy but is written at a separate disk location.
- **In-place update**: The disk version of the data item is overwritten by the cache version.

- ***Immediate update*** and ***deferred update*** are two main techniques for recovery

Database Recovery

5 Data Caching

Data items to be modified are first stored into *database cache* by the Cache Manager (CM) and after modification they are flushed (written) to the disk. The flushing is controlled by **Modified** and **Pin-Unpin** bits.

Pin-Unpin: Instructs the operating system not to flush the data item.

Modified: Indicates the AFIM of the data item.

Database Recovery

6 Transaction Roll-back (Undo) and Roll-Forward (Redo)

To maintain atomicity, a transaction's operations are **redone** or **undone**.

Undo: Restore all BFIMs on to disk (Remove all AFIMs).

Redo: Restore all AFIMs on to disk.

Database recovery is achieved either by performing only Undos or only Redos or by a combination of the two. These operations are recorded in the log as they happen.

Database Recovery

Roll-back

We show the process of roll-back with the help of the following three transactions T_1, T_2 and T_3.

T_1	T_2	T_3
read_item (A)	read_item (B)	read_item (C)
read_item (D)	write_item (B)	write_item (B)
write_item (D)	read_item (D)	read_item (A)
	write_item (D)	write_item (A)

Database Recovery

Roll-back: One execution of T_1, T_2 and T_3 as recorded in the log.

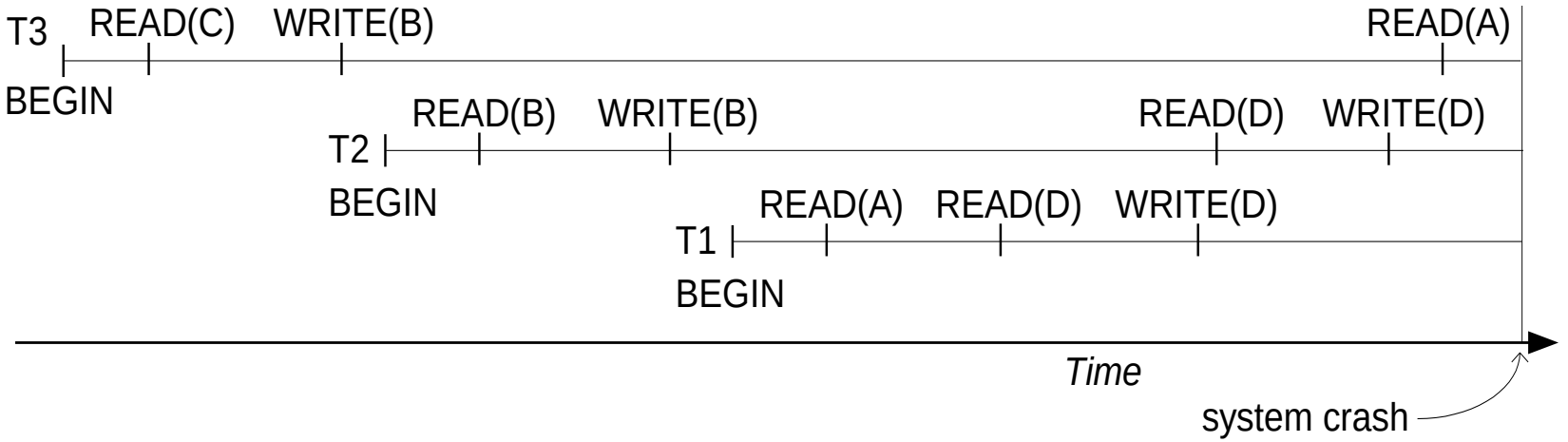
		A	B	C	D
		30	15	40	20
	[start_transaction, T_3]				
	[read_item, T_3, C]				
*	[write_item, T_3, B, 15, 12]		12		
	[start_transaction,T_2]				
	[read_item, T_2, B]				
**	[write_item, T_2, B, 12, 18]		18		
	[start_transaction,T_1]				
	[read_item, T_1, A]				
	[read_item, T_1, D]				
	[write_item, T_1, D, 20, 25]				25
	[read_item, T_2, D]				
**	[write_item, T_2, D, 25, 26]				26
	[read_item, T_3, A]				
	---- system crash ----				

* T_3 is rolled back because it did not reach its commit point.

** T_2 is rolled back because it reads the value of item B written by T_3.

Database Recovery

Roll-back: One execution of T_1, T_2 and T_3 as recorded in the log.



Illustrating cascading roll-back

Database Recovery

Write-Ahead Logging

When **in-place** update (immediate or deferred) is used then log is necessary for recovery and it must be available to recovery manager. This is achieved by **Write-Ahead Logging** (WAL) protocol. WAL states that

For Undo: Before a data item's AFIM is flushed to the database disk (overwriting the BFIM) its BFIM must be written to the log and the log must be saved on a stable store (log disk).

For Redo: Before a transaction executes its commit operation, all its AFIMs must be written to the log and the log must be saved on a stable store.

Database Recovery

Steal/No-Steal and Force/No-Force

Possible ways for flushing database cache to database disk:

Steal: Cache can be flushed before transaction commits.

No-Steal: Cache cannot be flushed before transaction commit.

Force: Cache is immediately flushed (forced) to disk when transaction commits.

No-Force: Cache is deferred when transaction commits.

These give rise to four different ways for handling recovery:

Steal/No-Force (Undo/Redo), Steal/Force (Undo/No-redo), No-Steal/No-Force (No-undo/Redo) and No-Steal/Force (No-undo/No-redo).

Database Recovery

7 Checkpointing

From time to time (randomly or under some criteria) the database flushes its buffer to database disk to minimize the task of recovery. The following steps defines a *checkpoint* operation:

1. Suspend execution of transactions temporarily.
2. Force write modified buffer data to disk.
3. Write a [*checkpoint*] record to the log, save the log to disk.
4. Resume normal transaction execution.

During recovery **redo** or **undo** is required to transactions appearing after [*checkpoint*] record.

Fuzzy checkpointing

- The time needed for force-write all modified buffers may delay transaction processing because of step 1. To reduce this delay, use ***fuzzy checkpointing***.
- In this technique, the system can resume transaction processing after the [*checkpoint*] record is written to the log without waiting for step 2 to finish.
- Until step 2 is completed, previous [*checkpoint*] record should remain valid. To accomplish this, the system maintain a pointer to the valid checkpoint, which continues to point to the previous [*checkpoint*] record in the log. Once step 2 is concluded, that pointer is changed to point to the new checkpoint in the log.

Database Recovery

8 Recovery Scheme

Deferred Update (No Undo/Redo)

The data update goes as follows:

1. A set of transactions records their updates in the log.
2. At commit point under WAL scheme these updates are saved on database disk.

After reboot from a failure the log is used to redo all the transactions affected by this failure. No undo is required because no AFIM is flushed to the disk before a transaction commits.

Database Recovery

Deferred Update in a single-user system

There is no concurrent data sharing in a single user system. The data update goes as follows:

The algorithm RDU_S uses a REDO procedure for redoing certain *write_item* operation:

PROCEDURE RDU_S: use two lists of transactions: the committed transactions since the last checkpoint, and the active transaction. Apply the REDO operation to all the *write_item* operations of the committed transactions from the log in the order in which they are written to the log. Restart the active transaction.

The REDO procedure:

REDO(WRITE_OP): Redoing a *write_item* operation WRITE_OP consisting of examining its log entry [*write_item, T, X, new_value*] and setting the value of *X* in the database to *new_value*, which is the after image (AFIM).

Database Recovery

Deferred Update in a single-user system

(a)

T_1	T_2
read_item (A)	read_item (B)
read_item (D)	write_item (B)
write_item (D)	read_item (D)
	write_item (D)

(b)

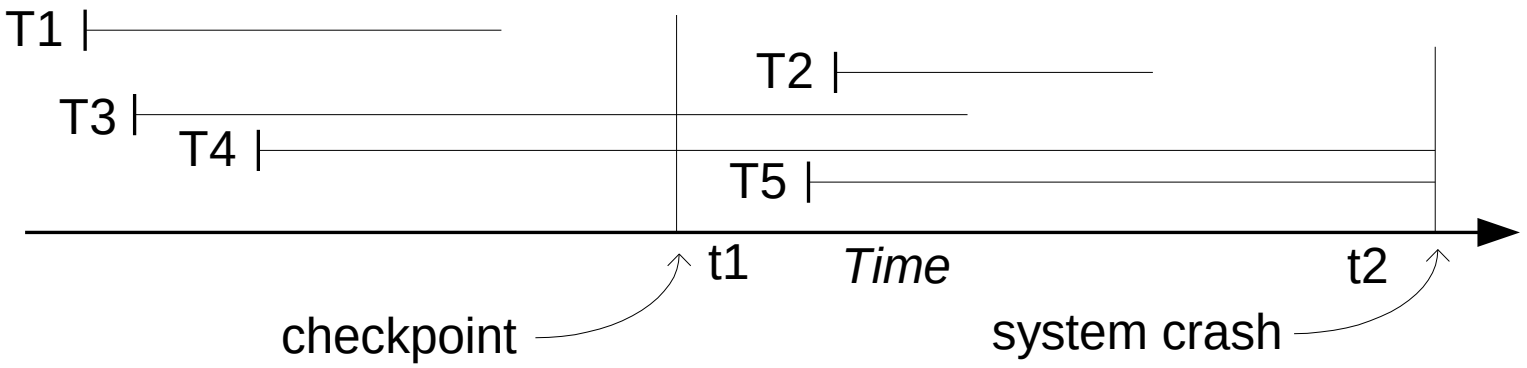
[start_transaction, T_1]
[write_item, T_1, D, 20]
[commit T_1]
[start_transaction, T_2]
[write_item, T_2, B, 10]
[write_item, T_2, D, 25] ← system crash

The [write_item, …] operations of T_1 are redone.
T_2 log entries are ignored by the recovery manager. (T_2 is not committed.)

Database Recovery

Deferred Update with concurrent users

This environment requires some concurrency control mechanism to guarantee **isolation** property of transactions. In a system recovery transactions which were recorded in the log after the last checkpoint were **redone**. The recovery manager may scan some of the transactions recorded before the checkpoint to get the AFIMs.



Recovery in a concurrent users environment.

Database Recovery

Deferred Update with concurrent users

(a)

T_1	T_2	T_3	T_4
read_item (A)	read_item (B)	read_item (A)	read_item (B)
read_item (D)	write_item (B)	write_item (A)	write_item (B)
write_item (D)	read_item (D)	read_item (C)	read_item (A)
	write_item (D)	write_item (C)	write_item (A)

(b)

[start_transaction, T_1]
[write_item, T_1, D, 20]
[commit, T_1]
[checkpoint]
[start_transaction, T_4]
[write_item, T_4, B, 15]
[write_item, T_4, A, 20]
[commit, T_4]
[start_transaction T_2]
[write_item, T_2, B, 12]
[start_transaction, T_3]
[write_item, T_3, A, 30]
[write_item, T_2, D, 25] ← system crash

T_2 and T_3 are ignored because they did not reach their commit points.
T_4 is redone because its commit point is after the last checkpoint.

Database Recovery

Deferred Update with concurrent users

Two tables are required for implementing this protocol:

1. **Active table**: All active transactions are entered in this table.
2. **Commit table**: Transactions to be committed are entered in this table.

During recovery, all transactions of the **commit** table are redone and all transactions of **active** tables are ignored since none of their AFIMs reached the database. It is possible that a **commit** table transaction may be **redone** twice but this does not create any inconsistency because a redone is "**idempotent**", that is, one redone for an AFIM is equivalent to multiple redone for the same AFIM.

Database Recovery

Recovery Techniques Based on Immediate Update

Undo/No-redo Algorithm

In this algorithm AFIMs of a transaction are flushed to the database disk under WAL before it commits. For this reason the recovery manager **undoes** all transactions during recovery. No transaction is **redone**. It is possible that a transaction might have completed execution and ready to commit but this transaction is also **undone**.

Database Recovery

Recovery Techniques Based on Immediate Update

Undo/Redo Algorithm (Single-user environment)

Recovery schemes of this category apply **undo** and also **redo** for recovery. In a single-user environment no concurrency control is required but a log is maintained under WAL. Note that at any time there will be one transaction in the system and it will be either in the commit table or in the active table.

The recovery manager performs:

1. **Undo** of a transaction if it is in the **active** table.
2. **Redo** of a transaction if it is in the **commit** table.

Database Recovery

Recovery Techniques Based on Immediate Update

Undo/Redo Algorithm (Concurrent execution)

Recovery schemes of this category applies undo and also redo to recover the database from failure. In concurrent execution environment a concurrency control is required and log is maintained under WAL. Commit table records transactions to be committed and active table records active transactions. To minimize the work of the recovery manager, checkpointing is used.

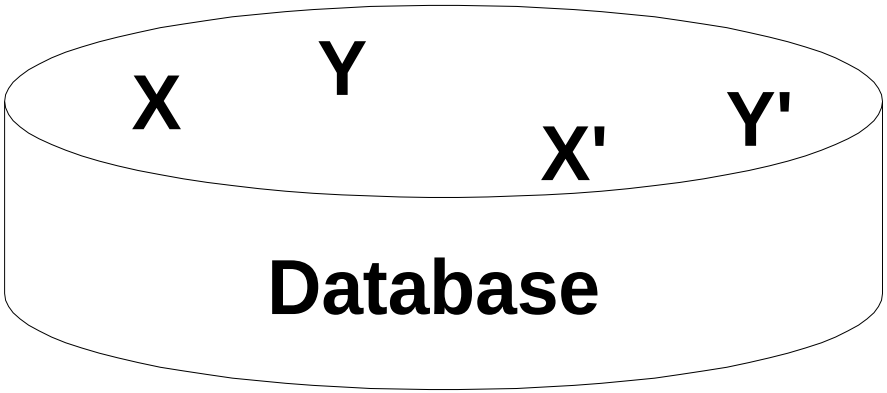
The recovery performs:

1. **Undo** of a transaction if it is in the active table.
2. **Redo** of a transaction if it is in the commit table.

Database Recovery

Shadow Paging

The AFIM does not overwrite its BFIM but recorded at another place on the disk. Thus, at any time a data item has AFIM and BFIM (Shadow copy of the data item) at two different places on the disk.

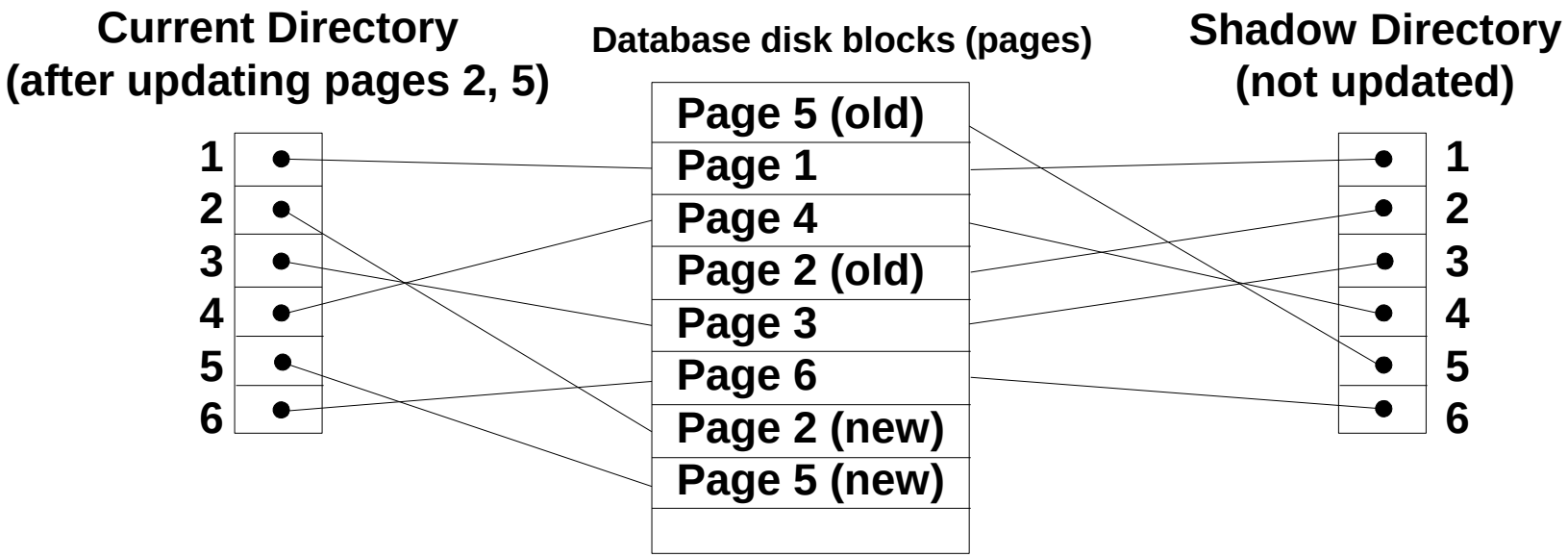


X and Y: Shadow copies of data items
X` and Y`: Current copies of data items

Database Recovery

Shadow Paging

To manage access of data items by concurrent transactions two directories (current and shadow) are used. The directory arrangement is illustrated below. Here a page is a data item.



Database Recovery

9 The ARIES Recovery Algorithm

The ARIES Recovery Algorithm is based on:

1. WAL (Write Ahead Logging)
2. Repeating history during redo: ARIES will retrace all actions of the database system prior to the crash to reconstruct the database state when the crash occurred.
3. Logging changes during undo: It will prevent ARIES from repeating the completed undo operations if a failure occurs during recovery, which causes a restart of the recovery process.

Database Recovery

The ARIES Recovery Algorithm

The ARIES recovery algorithm consists of three steps:

1. **Analysis**: step identifies the dirty (updated) pages in the buffer and the set of transactions active at the time of crash. The appropriate point in the log where redo is to start is also determined.
2. **Redo**: necessary redo operations are applied.
3. **Undo**: log is scanned backwards and the operations of transactions active at the time of crash are undone in reverse order.

Database Recovery

The ARIES Recovery Algorithm

The Log and Log Sequence Number (LSN)

A log record is written for (a) data update, (b) transaction commit, (c) transaction abort, (d) undo, and (e) transaction end. In the case of undo a compensating log record is written.

A unique LSN is associated with every log record. LSN increases monotonically and indicates the disk address of the log record it is associated with. In addition, each data page stores the LSN of the latest log record corresponding to a change for that page.

A log record stores (a) the previous LSN of that transaction, (b) the transaction ID, and (c) the type of log record.

Database Recovery

The ARIES Recovery Algorithm

The Log and Log Sequence Number (LSN)

A log record stores:

1. Previous LSN of that transaction: It links to the log record of each transaction. It is like a back pointer that points to the previous record of the same transaction.
2. Transaction ID
3. Type of log record.

For a *write* operation the following additional information is logged:

4. Page ID for the page that includes the item
5. Length of the updated item
6. Its offset from the beginning of the page
7. BFIM of the item
8. AFIM of the item

The ARIES Recovery Algorithm

- A log record is written for any of the following actions:
  - updating a page (write)
  - committing a transaction (commit)
  - aborting a transaction (abort)
  - undoing an update (undo)
  - ending a transaction (end)
- When an update is undone, *compensation log record* is written in the log.
- When a transaction ends, whether by committing or aborting, an *end log record* is written.

Database Recovery

The ARIES Recovery Algorithm

The Transaction table and the Dirty Page table

For efficient recovery following tables are also stored in the log during checkpointing:

Transaction table: Contains an entry for each active transaction, with information such as transaction ID, transaction status and the LSN of the most recent log record for the transaction.

Dirty Page table: Contains an entry for each dirty page in the buffer, which includes the page ID and the LSN corresponding to the earliest update to that page.

Database Recovery

The ARIES Recovery Algorithm

Checkpointing

A checkpointing does the following:

1. Writes a *begin_checkpoint* record in the log
2. Writes an *end_checkpoint* record in the log. With this record the contents of transaction table and dirty page table are appended to the end of the log.
3. Writes the LSN of the *begin_checkpoint* record to a special file. This special file is accessed during recovery to locate the last checkpoint information.

To reduce the cost of checkpointing and allow the system to continue to execute transactions, ARIES uses "fuzzy checkpointing".

Database Recovery

The ARIES Recovery Algorithm

The following steps are performed for recovery

1. **Analysis phase**: Start at the begin_checkpoint record and proceed to the end_checkpoint record. Access transaction table and dirty page table are appended to the end of the log. Note that during this phase some other log records may be written to the log and transaction table may be modified. The analysis phase compiles the set of redo and undo to be performed and ends.
2. **Redo phase**: Starts from the point in the log up to where all dirty pages have been flushed, and move forward to the end of the log. Any change that appears in the dirty page table is redone.
3. **Undo phase**: Starts from the end of the log and proceeds backward while performing appropriate undo. For each undo it writes a *compensating log record* in the log.

The recovery completes at the end of *undo* phase.

Database Recovery

An example of the working of ARIES scheme

(a)

LSN	LAST-LSN	TRAN-ID	TYPE	PAGE-ID	Other Info.
1	0	T1	update	C	-----
2	0	T2	update	B	-----
3	1	T1	commit		-----
4	begin checkpoint				
5	end checkpoint				
6	0	T3	update	A	-----
7	2	T2	update	C	-----
8	7	T2	commit		-----

At time of checkpoint

(b)

TRANSACTION TABLE

TRANSACTION ID	LAST LSN	STATUS
T1	3	commit
T2	2	in progress

DIRTY PAGE TABLE

PAGE ID	LSN
C	1
B	2

After the analysis phase

(c)

TRANSACTION TABLE

TRANSACTION ID	LAST LSN	STATUS
T1	3	commit
T2	8	commit
T3	6	in progress

DIRTY PAGE TABLE

PAGE ID	LSN
C	1
B	2
A	6

Database Recovery

10 Recovery in multidatabase system

A multidatabase system is a special distributed database system where one node may be running relational database system under Unix, another may be running object-oriented system under Window and so on. A transaction may run in a distributed fashion at multiple nodes. In this execution scenario the transaction commits only when all these multiple nodes agree to commit individually the part of the transaction they were executing.

This commit scheme is referred to as "*two-phase commit*" (2PC). If any one of these nodes fails or cannot commit the part of the transaction, then the transaction is aborted. Each node recovers the transaction under its own recovery protocol.