Unit - 2

Distributed Database Recovery

In order to recuperate from database failure, dbms resort to a number of recovery management techniques.

The typical strategies for database recovery are :-

* In case of soft failures that result in inconsistency of database, recovery strategy includes transaction undo or rollback.
* In case of hard failures resulting in extensive damage to database, recovery strategies encompass restoring a past copy of database from archival backup.

1) Recovery from Power Failure :-

Power Failure causes loss of information in the non-persistent memory. When power is restored, the operating system and the database management restart. Recovery manager initiates recovery from transaction logs. In case of immediate update mode, the recovery manager takes the following actions :-

1) Transactions which are in active list and failed list are undone and written on abort list.
2) Transactions which are in before-commit list are redone.
3) No action is taken for transaction in commit or abort lists.

2) Recovery from Disk failure :-

A disk failure or hard crash causes a total database loss. To recover from this hard crash, a new disk is prepared, then the operating system is restored, and finally the database is recovered using the database backup and transaction log. The recovery method is same for both immediate and defered update modes.

The recovery manager takes the following actions :-

1) The transactions in the commit list and before commit list are redone and written onto the commit list in transaction log.
2) The transactions in the active list and failed list are undone and written onto the abort list in transaction log.

3) Checkpointing :-

Checkpoint is a point of time at which a record is written onto the database from the buffers. As a consequence, in case of system crash, the recovery manager does not have to redo the transactions that have been committed before checkpoint. Periodical checkpointing shortens the recovery process.

The two types of checkpointing techniques are :-

1) Consistent checkpointing
2) Fuzzy checkpointing

i) Consistent Checkpointing :-

Consistent checkpointing creates a consistent image of database at checkpoint. During recovery, only those transactions which are on right side of last checkpoint are undone or redone. The transactions to left side of last consistent checkpoint are already committed and needn't be processed again.

The actions taken for check pointing are :-

* Active transactions are suspended temporarily
* All changes in main-memory buffers are written onto the disk.
* A "check point" record is written in transaction log.
* The transaction log is written to disk.
* The suspended transactions are resumed.

ii) Fuzzy Checkpointing

At time of check point, all active transactions are written in log. In case of power failure, the recovery manager processes only those transactions that were active during checkpoint and later. The transaction that have been committed before checkpoint are written to the disk and hence need not be redone.

4) Transaction Recovery using UNDO/REDO :-

Transaction recovery is done to eliminate the adverse effects of faulty transactions rather than to recover from a failure. Faulty transactions include all transactions that have changed the database into undesired state and transaction that have used values written by faulty transactions.

Transaction recovery in these cases is a two-step process:-

* UNDO all faulty transactions and transactions that may be affected by faulty transactions.
* REDO all transactions that are not faulty but have been undone due to faulty transactions.

→ Steps for UNDO operation :-

* If the faulty transaction has done INSERT, the recovery manager deletes the data item(s) inserted.
* If the faulty transaction has done DELETE, the recovery manager delete the data item(s) from the log.
* If the faulty transactions has done UPDATE, the recovery manager eliminates the value by writing the before-update value from log.

→ Steps for REDO operation :-

* If the transaction has done INSERT, the recovery manager generates an Insert from log.
* If the transaction has done DELETE, the recovery manager generates a delete from log.
* If the transaction has done UPDATE, the recovery manager generates an update from log.

2) Cryptography Algorithms :-

Cryptography is the science of encoding information before sending via unreliable communication paths so that only an authorized receiver can decode and use it.

* Encryption is the process of encoding a message in such a way as to hide its contents. Modern cryptography includes several secure algorithms for encrypting and decrypting message. They are all based on use of secrets called keys.
* A cryptographic key is a parameter used in an encryption algorithm in such a way that the encryption cannot be reversed without knowledge of the key.
* The first uses shared secret keys - The sender and the recipient must share a knowledge of the key and it must not be revealed to anyone else. The second class of encryption algorithms uses public/private key pairs.

Here the sender of a message uses a public key - one that has already been published by recipient - to encrypt the message. The recipient uses a corresponding private key to decrypt the message.

Uses of Cryptography

* Secrecy and Integrity :-

Cryptography is used to maintain the secrecy and integrity of information whenever it is exposed to potential attacks.

* Authentication :-

Cryptography is used in support of mechanism for authenticating communication between pair of principals. A principal who decrypts a message successfully using a particular key can assume that the message is authentic if it contains a correct checksum or some other expected value.

Algorithms :-

1) Symmetric key cryptography :-

It is an encryption system where the sender and receiver of message use a single common key to encrypt and decrypt message. Symmetric key systems are faster and simpler but the problem is that sender and receiver have to somehow exchange keys in a secure manner. The most popular symmetric key cryptography system is Data Encryption System (DES).

2) Hash Function :-

There is no usage of any key in this algorithm. A hash value with fixed length is calculated as per the plain text which makes it impossible for contents of plain text to be recovered. Many operating systems use hash functions to encrypt passwords.

(or)

It is a mathematical function used in cryptography. Typical hash functions takes inputs of variable lengths to return outputs of a fixed length.

3) Asymmetric key Cryptography :-

Under this system a pair of keys is used to encrypt and decrypt information. A public key is used for encryption and private key is used for decryption. Public key and Private key are different. Even if the public key is known by everyone the intended receiver can only decode it because he alone knows the private key.

3) Security Techniques (6M or 2M)

1) Interfaces are exposed :-

Distributed systems are composed of processes that offer services or share information. Their communication interfaces are necessarily open (to allow new clients to access them) - an attacker can send a message to any interface.

2) Networks are insecure :-

for ex, message source can be falsified.

3) Limit the lifetime and scope of each secret :-

When a secret key is first generated we can be confident that it has not been compromised. The use of secrets such as passwords and shared secret keys should be time-limited, and sharing should be restricted.

4) Algorithms and program code are available to attackers :-

Secret encryption algorithms are totally inadequate for today's large scale network environments. Best practise is to publish the algorithms used for encryption and authentication, relying only on the secrecy of cryptographic keys. This helps to ensure that the algorithms are strong by throwing them open to scrutiny by third parties.

5) Attackers may have access to large resources :-

The cost of computing power is rapidly decreasing. We should assume that attackers will have access to largest and most powerful computers projected in lifetime of a system, then add a few orders of magnitude to allow for unexpected developments.

6) Minimize the trusted base :

The portion of a system that are responsible for the implementation of its security, and all the hardware and software components upon which they rely, have to be trusted - this is often referred to as trusted computing base. Any defect or programming error in trusted base can produce security weakness, so we should aim to minimize its size.

Q5. Digital Signature →

Message authentication provides protection to two parties who exchange messages from any 3rd party

But it does not protect the two parties from eachother

A party 'A' can recieve an authenticated message from a party 'B'

Party 'A' can forge a different message and claim that it was sent by party 'B'. They would simply have to create a message and append the authentication code using the secret key that 'A' and 'B' share.

'B' can deny sending any message at all.

There is no way to prove that 'B' infact did send a message.

Both these scenarios are legitimate concerns

So something more than authentication is required.

Digital signature is a solution to this problem.

Properties of digital signature →

→ One can verify the author, and the date and time of the signature.

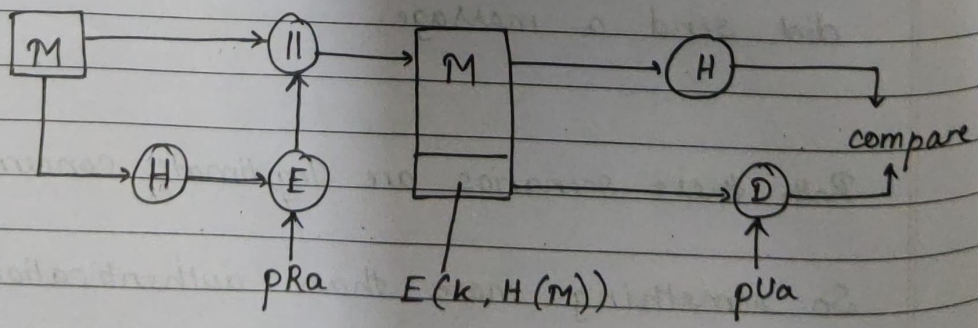
→ Can authenticate the contents at the time of signature

→ must be verifiable by 3rd parties to resolve disputes.

Digital signature can be classified into 2 types →

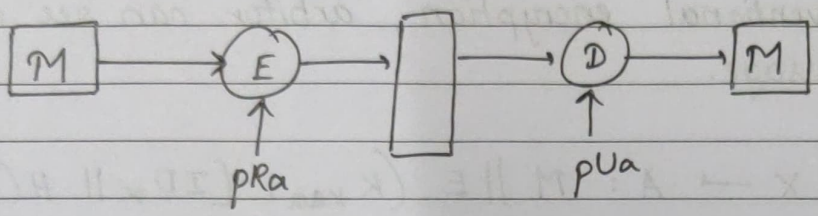
1. Direct digital signature
2. Arbitrated digital signature

Direct digital signature →

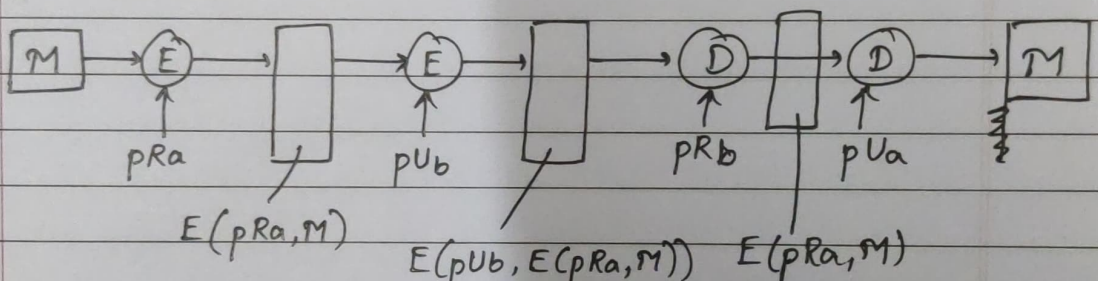


The hash function is encrypted

The message contains the message and the hash code



decrypting with the help of private key of a

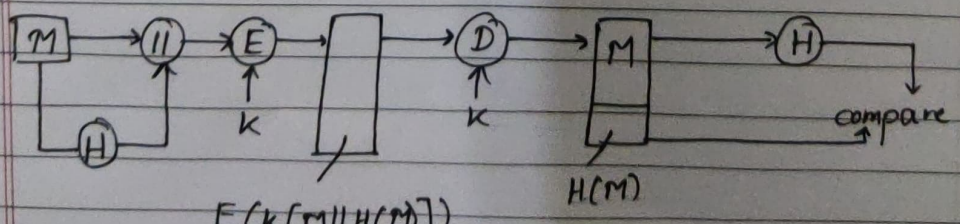


message is encrypted with the help of private key of a

message is encrypted again with the help of private key of b

message is then decrypted with the help of private key of b

message is then decrypted with the help of ~~pu~~ private key of a



Arbitrated Digital Signature →

conventional encryption, arbitor can see the message.

1. $X \rightarrow A : M \,||\, E\,(K_{xa}, [ID_x \,||\, H(M)])$

2. $A \rightarrow Y : E\,(K_{ay}, [ID_x \,||\, M \,||\, E\,(K_{xa}\,[ID_x \,||\, H(M)] \,||\, T])$

Distributed deadlocks

Deadlock is a state of a database system having two or more transactions, when each transaction is waiting for a data item that is being locked by some other transaction. A deadlock can be indicated by a cycle in the wait-for-graph. This is a directed graph in which the vertices denote transactions and the edges denote waits for data items.

For example, in the following wait-for-graph, transaction T1 is waiting for data item X which is locked by T3. T3 is waiting for Y which is locked by T2 and T2 is waiting for Z which is locked by T1. Hence, a waiting cycle is formed, and none of the transactions can proceed executing shown in Fig 2.6.

Deadlock Handling in Centralized Systems

There are three classical approaches for deadlock handling, namely –

- Deadlock prevention.
- Deadlock avoidance.
- Deadlock detection and removal.

Deadlock Prevention

The deadlock prevention approach does not allow any transaction to acquire locks that will lead to deadlocks. The convention is that when more than one transactions request for locking the same data item, only one of them is granted the lock.
One of the most popular deadlock prevention methods is pre-acquisition of all the locks. In this method, a transaction acquires all the locks before starting to execute and retains the locks for the entire duration of transaction. If another transaction needs any of the already acquired locks, it has to wait until all the locks it needs are available. Using this approach, the system is prevented from being deadlocked since none of the waiting transactions are holding any lock.

Deadlock Avoidance

The deadlock avoidance approach handles deadlocks before they occur. It analyzes the transactions and the locks to determine whether or not waiting leads to a deadlock.
The method can be briefly stated as follows. Transactions start executing and request data items that they need to lock. The lock manager checks whether the lock is available. If it is available,the lock manager allocates the data item and the transaction acquires the lock. However, if the item is locked by some other transaction in incompatible mode, the lock manager runs an algorithm to test whether keeping the transaction in waiting state will cause a deadlock or not. Accordingly, the algorithm decides whether the transaction can wait or one of the transactions should be aborted.

•**Wait-Die** – If T1 is older than T2, T1 is allowed to wait. Otherwise, if T1 is younger than T2, T1 is aborted and later restarted.
•**Wound-Wait** – If T1 is older than T2, T2 is aborted and later restarted. Otherwise, if T1 is younger than T2, T1 is allowed to wait.

Deadlock Detection and Removal

The deadlock detection and removal approach runs a deadlock detection algorithm periodically and removes deadlock in case there is one. It does not check for deadlock when a transaction places a request for a lock. When a transaction requests a lock, the lock manager checks whether it is available. If it is available, the transaction is allowed to lock the data item; otherwise the transaction is allowed to wait.
Since there are no precautions while granting lock requests, some of the transactions may be deadlocked. To detect deadlocks, the lock manager periodically checks if the wait-forgraph has cycles. If the system is deadlocked, the lock manager chooses a victim transaction from each cycle. The victim is aborted and rolled back; and then restarted later. Some of the methods used for victim selection are –

- Choose the youngest transaction.
- Choose the transaction with fewest data items.
- Choose the transaction that has performed least number of updates.
- Choose the transaction having least restart overhead.

Distributed Deadlock Detection
Just like centralized deadlock detection approach, deadlocks are allowed to occur and are removed if detected. The system does not perform any checks when a transaction places a lock request. For implementation, global wait-for-graphs are created. Existence of a cycle in the global wait-for-graph indicates deadlocks. However, it is difficult to spot deadlocks since transaction waits for resources across the network.Alternatively, deadlock detection algorithms can use timers. Each transaction is associated with a timer which is set to a time period in which a transaction is expected to finish. If a transaction does not finish within this time period, the timer goes off, indicating a possible deadlock. Another tool used for deadlock handling is a deadlock detector. In a centralized system, there is one deadlock detector. In a distributed system, there can be more than one deadlock detectors. A deadlock detector can find deadlocks for the sites under its control. There are three alternatives for deadlock detection in a distributed system, namely.

- **Centralized Deadlock Detector –** One site is designated as the central deadlock detector.
- **Hierarchical Deadlock Detector –** A number of deadlock detectors are arranged in hierarchy.
- **Distributed Deadlock Detector –** All the sites participate in detecting deadlocks and removing them.