Unit

5 Concurrency Control Techniques

- When several transactions execute simultaneously in a database is called concurrent execution of the transaction.
- When many transactions execute concurrently in the database, however the isolation property may be failed, therefore the system must control the interaction among the current transactions. This control achieved by a mechanism called concurrency control mechanism.

5.1 LOCKING TECHNIQUES FOR CONCURRENCY CONTROL

5.1.1 Lock

A lock is a variable associated with a data item that describes the status of the item with respect to possible operations that can be applied to the transaction.

- A locking protocol is a set of rules that state when a transaction may lock or unlock each of the data items in the database.
- Every transaction must follow the following rules :

(i) A transaction T must issue the operation Lock-item (A) before any read-item (A) or write-item (A) operations are performed in T.

(ii) A transaction T must issue the operation unlock-item (A) after all read-item (A) and write-item (A) operations are completed in T.

(iii) A transaction T will not issue a lock-item (A) operation if it already holds the lock on item (A).

These rules inforced by lock manager between the lock-item (A) & unlock-item (A).

There are many modes in which a data item may be locked.

(i) Shared Mode (S) : If a transaction Ti has obtained a shared mode lock on data item (A), then Ti can read, but cannot write A. Shared mode denoted by S.

(ii) Exclusive-Mode (X) : If a transaction Ti has obtained an exclusive-mode lock on item A, Then Ti can both read & write A.

Exclusive-mode denoted by X.

	S	X
S	True	False
X	False	False

Lock-compatibility matrix comp.

- A transaction requests a shared lock on data item A by executing the lock-S(A) instruction. Similarly, a transaction requests an exclusive lock through the lock-X(A) instruction.

For Example :

T_1 : lock-X (B);
read (B);
B : = B − 50;

```
        write (B)
        unlock (B);
        lock-X (A);
        read (A);
        A : = A + 50;
        write (A);
        unlock (A);
T2 :    lock-S (A);
        read (A);
        unlock (A);
        lock-S (B)
        read (B);
        unlock (B);
```

5.1.2 The Two-Phase Locking Protocol

A transaction is said to follow the two-phase locking protocol if all locking operations precede the first unlock operation in the transaction.

This protocol has divided into two phase.

- **Growing Phase :** A transaction may obtain new lock, but may not release any lock.
- **Shrinking Phase :** A transaction can realase lock but may not obtain any new lock.

Condition of 2 phase Locking Protocol :

- A transaction is in the growing phase. The transaction obtained locks as needed. Once the transaction release a lock, it enters the shrinking phase and it can issue no more lock request. There are various version of two phase locking protocol.

(i) Dynamic 2-phase locking : Here a transaction locks a data item immediately before any operation is applied on the data item. After finishing all the operations on all d ta item, it release all the locks .

Example :

```
T :     lock-X(A);
        read (A);
        A : = A - 50;
        write (A);
        lock-X(B);
        read (B);
        B : = B + 50;
        write (B);
        unlock (A);
        unlock (B);
```

(2) Static (Conservative) two-phase locking :

In this scheme, all the data items are locked before any operation on them and are released any after the last operation performed on any data item.

Example :

```
T :     lock-X(B);
        lock-X(A);
        read (B);
        B : = B - 50;
```

```
write (B);
read (A);
A : = A + 50;
write (A);
unlock (B);
unlock (A);
```

(3) Strict two-phase locking : In computer science, strict 2-PH locking is a locking method used in concurrent system.

The two rules of strict 2PL are :

(i) If transaction T wants to read/write an object, it must request a shared/exclusive lock on the object.

(ii) All exclusive lock held by transaction T are released when T commits or aborts (& not before).

T_1	T_2
Lock – S (A)	
read (A)	
unlock (A)	
	Lock – S (A)
	read (A)
	Lock – X (B)
	read (B)
	write (B)
	unlock (A)
	unlock (B)

Note : Strict 2PL prevents transactions reading uncommitted data, overwriting uncommitted data and unrepeatable reads. Thus it prevent cascading roll backs since exclusive lock must be held until a transaction commits.

(4) The Rigorous 2 P Locking : Which requires that all locks be held until the transaction commits. We can easily verify that, which rigorous two-phase locking, transaction can be serialized in the order in which they commit.

T_1	T_2
Lock – X (A)	
read (A)	
A : = A + 50	
write (A)	
unlock (A)	
	Lock – X (A)
	read (A)
	temp : = A * 0.3
	A = A + temp
	write (A)
	unlock (A)

5.2 CONCURRENCY CONTROL BASED ON TIMESTAMP PROTOCOL

Timestamp is a unique identifier created by the DBMS to identify a transaction.

Timestamp values are assigned in the order in which the transactions are submitted to the system, so a timestamp can be though as the start time.

This techniques do not use locks, so deadlocks can not occur.

The algorithm associates with each database item X two time stamp (TS) values.

(1) **Read-TS(X)** : The read timestamp of item X. This is the timestamps of transactions that have successfully read item X.

i.e., read-TS(X) = TS (T)

where T is the youngest transaction that has read X successfully.

(2) **Write-TS(X)** : The write timestamp of item X. This is the largest of all the timestamp of transactions that have successfully written item X.

i.e., Write-TS (X) = TS (T)

where T is the yougest transaction that has written X successfully.

- Whenever some transaction T ties to issue a read-item (X) or a write-item (X) operation, the basic to algorithm compares the timestamp of T with read-TS (X) & write-TS(X) to ensure that the timestamp order of transaction execution is not violated.
- Whenever conflicting operations violate the timestamp ordering in the following two cases.

(i) Transaction Ti issues read (X) operation :

(a) If write - TS(X) > TS(T),

- Then T needs to read a value of X that was already over written.
  Hence, the read operation is rejected & T is rolled back.

(b) If write-TS(X) ≤ TS(T),

- Then the read operation is executed & read-TS(X) is set to the larger of TS(T) and the current read-TS(X).

The read Rule :

```
if (TS (Ti) ≥ W-TS(X))
        {
                R-TS(X) = MAX (R-TS(X), TS(i));
                return OK;
        }
        else
        {
                return REJECT;
        }
```

(ii) Transaction T issues write (X) operation :

(a) If read-TS(X) > TS(T)

- Then the value of X that T is producing was needed previously, and the system assumed that, that value would never be produced. Hence, the system 'REJECTS' the write operation & 'Roll back' T.

(b) If write-TS(X) > TS(T),

- Then T is attempting to write an absolute value of X.
  Hence the system 'REJECTS' this write operation and rollback T.

(C) Otherwise, the system executes the write operation and Set write-TS(X) to TS(T).
Write Rule :

```
if (TS(Ti) ≥ W-TS (X) &&
          TS(Ti) ≥ R-TS(X))
  {
          W-TS(X) = MAX (W-TS(X), TS(i));
          return OK;
  }
  else
  {
          return REJECT;
          }
```

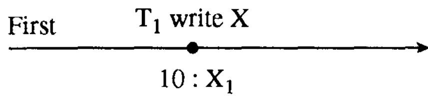
Ex. 1. T_1 : read (B);
read (A);
display (A + B);

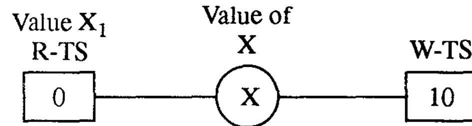
T_2 : read (B);
B : B – 50;
write (B);
read (A);
A : = A + 50;
display (A + B);

In this schedules under timestamp protocol, we shall assume that a transaction is assigned a timestamp immediately before its instruction.

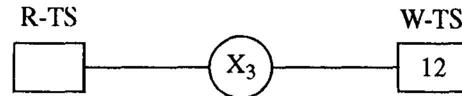
T_1	T_2
read (B)	
	read (B)
	B : = B – 50
	write (B)
read (A)	
	read (A)
display (A + B)	
	A : = A + 50
	write (A)
	display (A + B)

Thus in this schedule $TS(T_1) < TS(T_2)$ and the schedule is possible under time stamp protocol.
Ex. 2. T_1 : W(X) T_2 : R(X) T_3 : W(X)
- T_1 Starts, given TS 10.... Then T_2 Starts, given TS11.
- a while later, T_3 starts, given TS12.





Then, T_3 over writes X with X_3



Now T_2 request to read X?

Advantages of timestamp protocol :

- Deadlock free because no transaction even waits.
- Avoids control lock manager.
- Attractive in distributed database system.

Disadvantages of timestamp protocol :

- T abort/restart can degree performance under high contention.
- Storage overhead per data item for timestamps.
- Update overhead to maintain timestamp.
- Potentially update during each read/write to a data item.

Note : The timestamp-ordering protocol ensure conflict serializability, because operations are processed in timestamp order.

5.3 VALIDATION (OPTIMISTIC)-BASED PROTOCOL

A validation scheme is an concurrency-control method in cases where a majority of transactions are read only transaction and, thus the rate of conflicts among these transactions is low.

- The validation or certification technique also known as optimistic concurrency control techniques.
- In validation concurrency control technique, no checking is done while the transaction is execution. During the transaction execution, all updates are applied to local copies of the data items that are kept for the transaction.
- If serializability is not violated, the transaction is committed and the database is updated from the local copies; otherwise, the transaction is aborted and the restarted later.

There are three phases for validation concurrency control protocol.

(1) Read phase : A transaction can read values of committed. However, updates are applied only to local copies of data items kept in the transaction workspace.

(2) Validation phase : Checking is performed to ensure that serializability will not be violated if the transaction updates are applied to the database.

(3) Write phase : If the validation phase is successful, the transaction updates are applied to the database otherwise, the updates are discarded and the transaction is restarted.

To perform the validation test, we need to know when the various phases of transaction T took place.

There are three different timestamps with transaction T.

(1) Start (T) : The time when transaction T started its execution.

(2) Validation (T) : The time when transaction T finished its read phase and started its validation phase.

(3) Finish (T) : The time when transaction T finished its write phase.

The validation test for transaction Tj requires that, for all transaction Ti which TS(Ti) < TS(Tj), one of the following two conditions must hold.

(i) Finish (Ti) < start (Tj)

Since Ti completes its execution before Tj started.

(ii) Start (Tj) < Finish (Ti) < Validation (Tj)

This condition ensure that the writes of Ti & Tj do not overlap; because writes of Ti do not affect the read of Tj.

Example :

T_1	T_2
read (B)	
	read (B)
	B : = B – 50
	read (A)
	A : = A + 50
read (A)	
<validate>	
display (A + B)	
	<validate>
	write (B)
	write (A)

Advantages of Optimistic Methods

The optimistic concurrency control has the following advantages :

- This technique is very efficient when conflicts are rare. The occasional conflicts result in the transaction roll back.
- The rollback involves only the local copy of data, the database is not involved and thus there will not be any cascading rollbacks.

Problems of Optimistic Methods

The optimistic concurrency control suffers from the following problems :

- Conflicts are expensive to deal with, since the conflicting transaction must be rolled back.
- Longer transactions are more likely to have conflicts and may be repeatedly rolled back because of conflicts with short transactions.

Applications of Optimistic Methods

- Only suitable for environments where there are few conflicts and no long transactions.
- Acceptable for mostly Read or Query database systems that require very few update transactions.

5.4 MULTIPLE GRANULARITY LOCKING

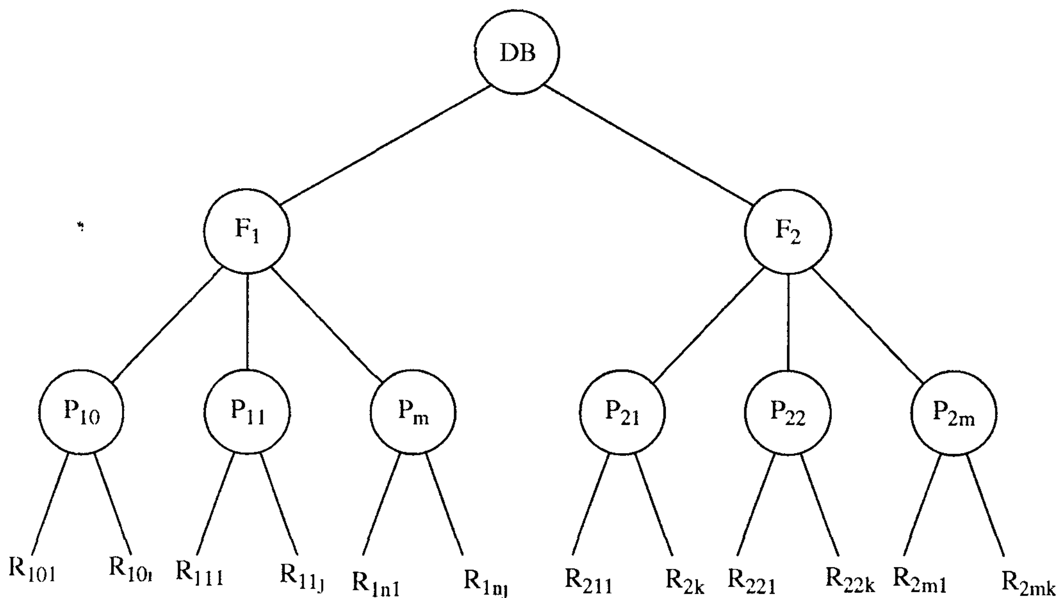
- In computer science, multiple granularity locking, sometimes called the 'John Rayner' locking method, is a locking method used in DBMS & RDBMS.

- In multiple granularity locking, locks are set of objects that contains object. MGL exploits the hierarchical nature of the contains relationship.
- A database item could be choosen one of the following :
  (i) A database record.
  (ii) A field value of a database/record.
  (iii) A whole file.
  (iv) The whole database.
  (v) A disk block.

The size of data items is often called the data itme of granularity.

Fine granularity means small item sizes where as coarse granularity mean large item size.

Example : A database may have files, which contain pages, which futher contains records. This can be thought of as a tree of objects, where each node contains its children. A lock locks a node and its descendents.



Granularity Hierarchy

- Multiple granularity locking is usually used with Non-Strict two phase locking to guarantee serializability. Where a lock can be requested at an level.

There are three types of intention locks.

(1) Intention-shared (IS) indicates that a shared lock (S) will be requested on some descendent node (S).

(2) Intention-exclusive (IX) indicates that on exclusive lock (X) will be requested on some descendant node (S).

(3) Shared-Intention-exclusive (SIX) indicates that the current node is locked in shared mode but an exclusive lock (S) will be requested on some descendent node (S).

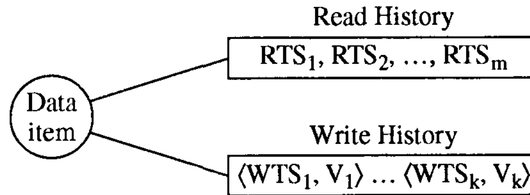
	IS	IX	S	SIX	X
IS	Yes	Yes	Yes	Yes	No
IX	Yes	Yes	No	No	No
S	Yes	No	Yes	No	No
SIX	Yes	No	No	No	No
X	No	No	No	No	No

To lock a node in (S or X), MGL has the transaction locks of all of its ancestors with IS (or IX), so if a transaction lock a node in S (or X), no other transaction can access its ancestors in X (or S & X).

5.5 MULTI-VERSION SCHEMES

In multiversion concurrency control schemes.

- Each write (X) operation creates a new "version" of the item X.
- A TS (Xn) does not overwrite old values of a data item X.
- A read operation is never rejected.



The W-timestamp & R-timestamp of Nth Version : In this method, several versions X_1, X_2, X_3 .... X_n of each data item X are maintained for each version, the value of version X_n.

(i) Read-TS(X_n) OR [R-TS (X_n)] : The read timestamp of X_n is the largest of all the timestamp of transactions that have successfully read version X_n.

i.e.,

read by TS(X_n) with timestamp TS
read Vj where
$j = \text{Max}\{i / TSi < TS\}$
Add TS to read history;

(ii) Write-TS (X_n) OR [W-TS (X_n)] :

The write timestamp of X_n is the timestamp of the transaction that wrote the value of version X_n.

When ever a transaction T is allowed to execute a write-item (X) operation a new version X_{n+1} of item X is created with read-TS (X_{n+1}) & write-TS (X_{n+1}) set to TS(T).

That is write val by TS (X_n) with
TS
if (there exists k such that
$TS < R\text{-}TS_k < W\text{-}TSj$
where $j = \min\{i \backslash TS < W\text{-}TSi\}$
{
REJECT;
}
else,

{
Add < TS, Val> to write history;
}

- When a transaction T is allowed to read the value of version X_n, the value of read-TS(X_n) is set of the larger of the current read-TS(X_n) and TS(T).

To ensure serializability, the following two rules are used.

(1) If transaction T issues write (X) operation, and version n of x has the highest write-TS(X_n) of all version of X that is also less than or equal to TS(T),
i.e., write-TS(X_n) $\leq$ TS(T)
& read-TS (X_n) > TS(T), Then
abort & rollback transaction T;
otherwise, create a new version Xj of X with
read-TS(Xj) = write-TS(Xj) = TS(T).

(2) If transaction T issues a read (X) operation, find the version n of X that has the highest, write-TS(X_n) of all version of X that is <= TS (T)
i.e., write-TS(X_n) $\leq$ TS(T). Then return the value of X_n to T, & set the value of read-TS(X_n)> to the larger of TS(T) & current read-TS(X_n).

5.6 MULTI-VERSION TWO-PHASE LOCKING

The multiversion two-phase locking protocol attempts to combine the advantages of multiversion concurrency control with the advantages of two-phase locking.

This protocol differentiates between read-only transactions & updates transaction.

- Update transactions perform rigorous two-phase locking, that is, they hold all locks up to the end of the transaction.

Thus they can be serialized according to their commit order. Each version of a data item has a single timestamp. The timestamp in this case is not a real clock-based timestamp.

- When a read-only transaction Ti issues a read (Q), the value returned is the contents of the version whose timestamp is the largest timestamp less than TS(Ti).
- When an update transaction reads an item, it gets a shared lock on the item, and reads the latest version of that item.
- When an update transaction wants to write an item, it first gets an exclusive lock on the item, and then creates a new version of the data item. When the update transaction Ti completes its actions, it carries out commit processing.

A multiversion two-phase locking scheduler works as follows :

(i) Each read requires a read lock on the item being read.
(ii) Each write requires a write lock on the item being written.
(iii) A write lock prevents....................
(a) Reading of the item but not its earlier version.
(b) Creating of a new version of the item.

Slightly more flexible variants of the two 2PL schedulers. ....

- Read not only the most recent version of items.
- But that can lead to cascading aborts.

The scheduler uses three types of locks

- read lock that collides with certify lock (but not write lock)
- write lock that collides with write & certify lock (but not read lock)
- certify lock that collides with read, write, & certify lock.

5.7 RECOVERY WITH CONCURRENT TRANSACTIONS

We discuss here how we can modify and extend the log-based recovery scheme to deal with multiple concurrent transaction.

(i) Interaction with concurrency control : The recovery scheme depends mostly on the concurrency-control scheme that is used, to roll back a failed transaction, we must undo the updates performed by the transaction.

For example : Suppose a transaction T_1 has to be rolled back, and a date item Q that was updated by T_1 has to be restored to its old value.

Using the log-based schemes for recovery, we restore the value by using the UNDO information in a log record.

(ii) Transaction Rollback : We rollback a failed transaction Ti by using log.

The system scans the log back word for every log record of the form $< T_1, T_2, V_1, V_2 >$ found in the log, the system restores the data item X_2, to its old value V_1. Scanning of the log terminates when the log record $< T_1$, start $>$ is found.

(iii) Check points : We used checkpoints to reduce the number of log records that the system must scan when it recovers from a crash.

It was necessary to consider only the following transactions during recovery.

- Those transactions that started after the most recent checkpoint.
- The one transaction, if any that was active at the time of the most recent checkpoint.

In a concurrent transaction processing system, we require that the checkpoint log record be of the form < checkpoint L>, where L is a list of transaction active at the time of the checkpoint.

(iv) Restart Recovery : When the system recovers from a crash, it constructs two lists :

(a) The UNDO-Lists consists of transaction to be : undone.

(b) The REDO-lists consists of transaction to be redone.

Once the REDO-lists and UNDO-lists have been constructed, the recovery proceeds as follows:

(a) The system rescans the log from the most recent record backward & performs an UNDO for each log record.
The scan stops when the < T start > records have been found for every transaction T in the UNDO-list.

(b) The system locates the most recent < checkpoints L > record on the log.

(c) The system scans the log forward from the most recent < checkpoint > record, & perform REDO & each log record that belongs to a transaction T that is on the REDO-lists.

5.8 DISTRIBUTED DATABASE

Distributed System : Distributed system is a collection of autonomus systems that communicate via communication network.

Distributed Database Concept : A distributed database is a collection of multiple logically interrelated database distributed over a computer network.

A distributed database management system is a software system that manages a distributed database while making the distribution transparent to the user.

In a Distributed database system, both data and transaction processing are divided between one or more computers (CPUs) connected by network, each computer playing a special role in the system. The computers in the distributed systems communicate with one another through various communication media, such as high-speed networks of telephone lines. They do not share main memory or disk. A distributed database system allows applications to access data from local and remote database. Distributed database systems use client/Server architecture to process information requests.

The computers in distributed system may vary in size and function, ranging from workstations upto mainframe systems. The computers in a distributed database system are referred to by a number of different names, such as sites or nodes.

Properties of Distributed Databases : Distributed database system should make the impact of data distribution transparent. Distributed database systems should have the following properties.

- Distributed data independence.
- Distributed transaction atomicity.

Distributed Data Independence : Distributed data independence property enables users to ask queries without specifying where the reference relations or copies or fragments of the relations are located. This principlė is a natural extension of physical and logical data independence. Further, queries that span multiple sites should be optimised systematically in a cost-based manner, taking into account communication costs and difference in local computation costs.

Distributed Transaction Atomicity : Distributed transaction atomicity property enables users to write transactions that access and update data at several sites just as they would write transactions over purely local data. In particular, the effects of a transaction across sites should continue to be atomic. That is all changes persist if the transaction commits, and none persist if it aborts.

5.8.1 Classification of Distributed Database

We can classifie distributed database as :

- Homogeneous
- Hetrogeneous

Homogeneous Distribtued Database

In a homogeneous distributed database, all sites have identical management system software that agree to cooperate in processing users requests.

In such a system, local sites surrender a portion of their autonomy in terms of their right to change schemes or DBMS software.

Homogeneous DDBS is the simplest form of a distributed database where there are several sites, each running their own applications on the same DBMS software. All sites have identical DBMS software, all users use identical software, are aware of one another end agree to co-operate in processing user's request. The application can all see the same schema and run the same transactions. That is, there is location transparency in homogeneous DDBS.

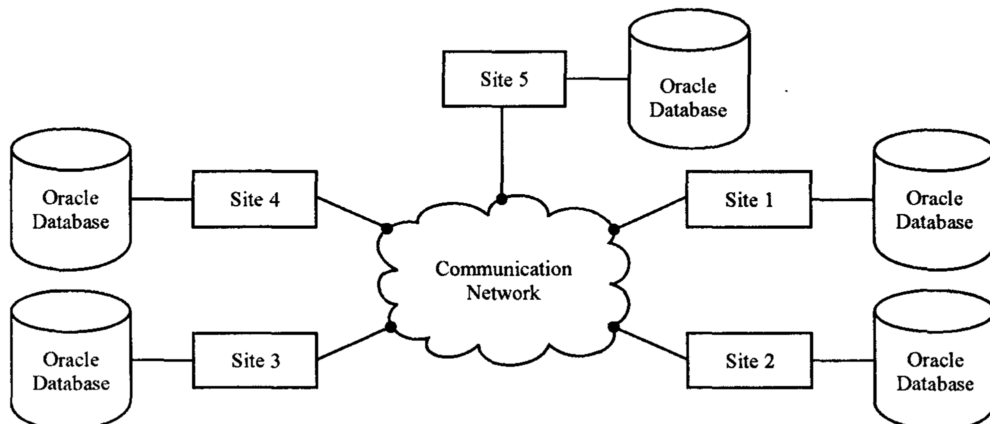


Fig. Homogeneous distributed database architecture.

Heterogneous Distributed Database

In a heterogeneous distributed database, different sites may use different schemes and different database managment system software.

This sites may not be aware of one another and they may provide only limited facilities for cooperation in transaction processing.

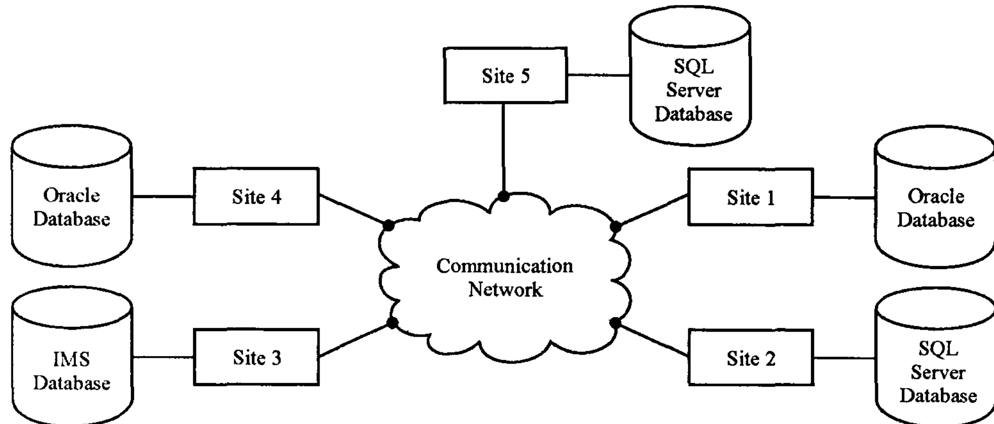


Fig. Heterogeneous DDBS acrhitecture.

Heterogeneous distributed database system is also referred to as a multi-database system or a federated database system. Heterogeneous database systems have well-accepted standards for gateway protocols to expose DBMS functionality to external applications. The gateway protocols help in masking the differences of accessing database servers, and bridge the differences between the different servers in a distributed system.

5.8.2 Functions of Distributed Database

The distributed database management system (DDBMS) must be able to provide the following additional functions as compared to a centralized DBMS.

(1) Ability of keeping track of data, data distribution, fragmentation, and replication by expanding DDBMS catalog.
(2) Ability of replicated data management to access and maintain the consistency of a replicated data item.
(3) Ability to manage distributed query processing to access remote sites and transmission of queries and data among various sites via a communication network.
(4) Ability of distributed transaction management by devising execution strategies for queries and transactions that access data from several sites.
(5) Should have fragmentation independence, that is users should be presented with a view of the data in which the fragments are logically recombined by means of suitable JOINs and UNIONs.
(6) Should be hardware independent.
(7) Provide local autonomy.
(8) Distributed catalog management.
(9) Should be location independent.
(10) Should be operating system independent.
(11) Should be network independent.

(12) Should be DBMS independent.
(13) Efficient distributed database recovery management in case of site crashes and communication failures.
(14) Proper management of security of data by provide authorized access privileges to users while executing distributed transaction.

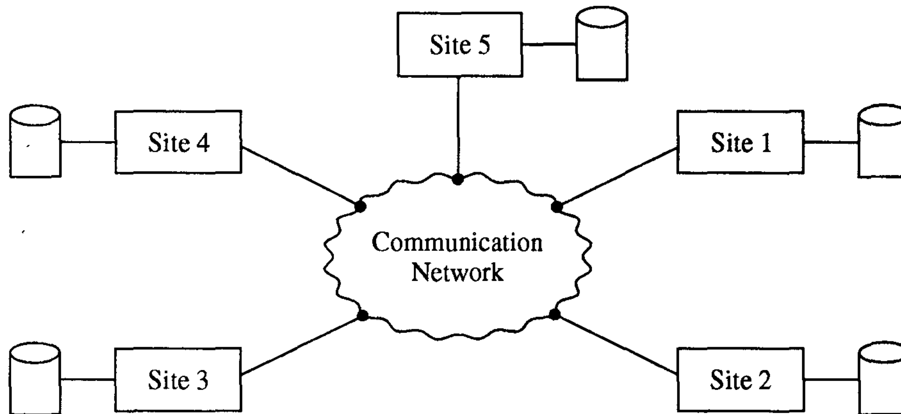


Fig. Distributed database architecture

5.8.3 Advantages of Distributed Database

These are following advantages of distributed database :

(1) Management of distributed data with different levels of Transparency : Distribution transparent means hiding the details of where each file (table) is physically stored in the system.

These are following types of tranparencies which are possible in DDB.

(a) Fragmentation Transparency : Users are not required to know how a relation has been fragmented.

It is two types :

(i) Horizontal fragmentation distributes a relation into set of rows.
(ii) Vertical fragmentation distributes a relation into a set of columns.

(b) Replication Transparency : Replication means the copies of data.

That is, in the replication transparency, copies of data may be stored at multiple sites for better availability, performance & reliability.

(c) Network Transparency : In network transparency, freedom for user from the operational details of the network.

- Naming transparency
- Location transparency

(i) Naming transparency implies that once a name is specified, the named objects can be accessed unambiguously without additional specification.

(ii) In location transparency, users are not required to know the physical location of the data.

(2) Increased Reliability & Availability : These two are most important advantages of distributed database.

Reilability means, " The probability that a system is running (not down) at certain time point".

Availability means "The probability that the system is continuously available during a time interval".

(3) **Security :** Distributed transaction executed with the proper management of the security of the data.

(4) **Distributed query processing :** The ability to access remote sites & transmit queries and data among the various sites via a communication network.

(5) **Distributed Database Recovery :** The ability of DDB to recover from individual sites crashes.

(6) **Replicated Data Management :** The ability to decide which copy of a replicated data item to access to maintain the consistency of the replicated data items.

(7) **Keeping track of data :** The ability of DDB to keep track of the data fragmentation, distribution, & replication by expanding DDBMS catalog.

(8) Improved scalability.

(9) Easier expansion.

(10) Improved the performance.

(11) Parallel evaluation.

5.8.4 Disadvantages of Distributed Database

(1) Technical problem of connecting dissimilar machine.

(2) **Software cost and Complexity :** More complex software is required for a distributed database environment.

(3) **Difficulty in Data Integrity Control :** A byproduct of the increased complexity and need for coordination is the additional exposure to improper updating and other problems of data integrity.

(4) **Processing Overhead :** The various sites must exchange messages and perform additional calculation to ensure proper coordination among the sites.

(5) Communication Network failures.

(6) Loss of messages.

(7) Recovery of failure is more complex.

(8) Increased complexity in the system design and implementation.

(9) Security concern of replicated data in multiple location and the network.

(10) Increased transparency leads to a compromise between ease of use and the overhead cost of providing transparency.

(11) Greater potential for bugs.

5.8.5 Architecture of Distributed Databases

Distributed databases use a client/server architecture to process information requests.

Client/Server Architecture : Client/Server architectures are those in which a DBMS related work load is split into two logical components namely client and server, each of which typically executes on different systems. Client is the user of the resource whereas the server is a provider of the resource. The applications and tools are put on one or more client platforms and are connected to database management system that resides on the server. The applications and tools act as 'client' of the DBMS, making requests for its services. The client/sever architecture can be used to implement a DBMS in which the client is the transaction processor and the server is the data processor.

The client applications issue SQL statements for data access, just as they do in centralised computing client applications to connect to the server, send SQL statements and receive results or error return code after server has processed the SQL statements.

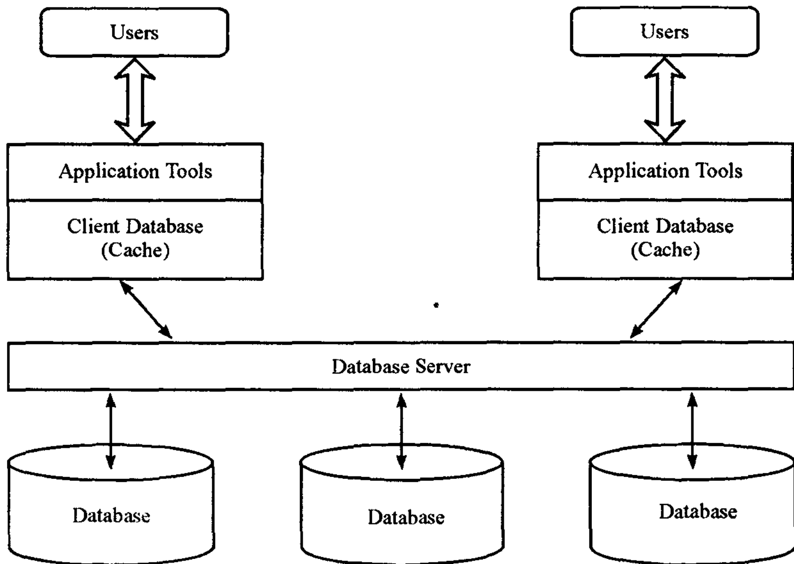


Fig. Client/Server database architecture.

Client/Server architecture consists of the following main components :

- Clients in the form of intelligent workstations as the user's contact point.
- DBMS server as common resources performing specialised tasks for devices requesting their services.
- Communication networks connecting the clients and the servers.
- Software applications connecting clients, servers and networks to create a single logical architecture.

A client can connect directly or indirectly to a database server. A direct connection occurs when a client connects to a server and accesses information from a database contained on the server. In

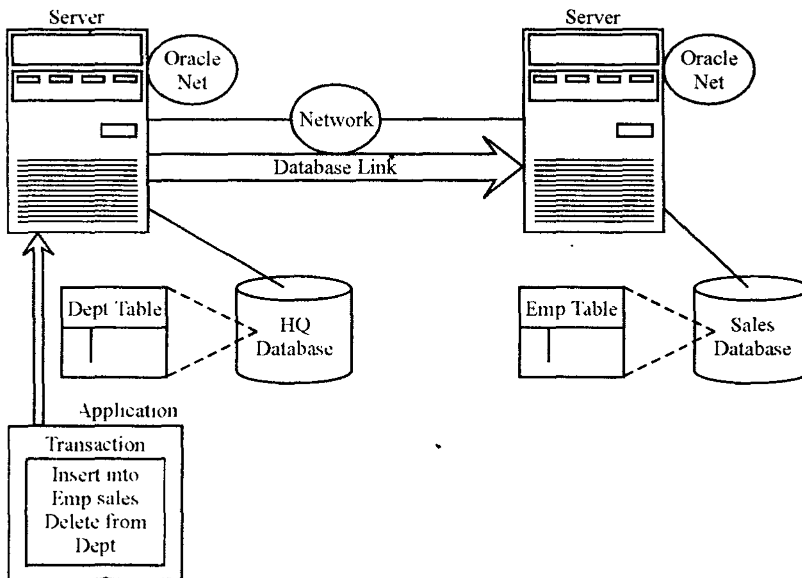


Fig. An oracle database distributed database system

contrast, an indirect connection occurs when a client connects to a server and then access information contained in a database on a different server.

Example : If you connect to the HQ database and access the dept table on this database as in figure, you can issue the following :

SELECT * FROM dept;

This query is direct because you are not accessing on obejct on a remote database.

- If you connect to the HQ database but access the emp table on the remote sales database as in figure, you can issue the following :

SELECT * FROM emp@sales;

This query is indirect because the object you are accessing is not on the database to which you are directly connected.

Advantages of Client/Server Database Architecture

(1) In most cases, a client/server architecture enables the roles and responsibilities of a computing system to be distributed among several independent computers that are known to each other only through a network.
This creates an additional advantages to this architecture : greater ease of maintenance.
(2) It functions with multiple different users with different capabilities.
(3) This architectures is relatively simple to implement, due to its clean separation of functionality because the server is centralised.
(4) Improved performance with more processing power scattered throughout the organization.
(5) All the data is stored on the servers, which generally have far greater security controls than most clients. Server can better control access and resources, to guarantee that only those clients with the appropriate permissions may access and change data.
(6) Since data stored in centralised, updates to that data are far easier to administer than what would be possible under a peer to peer.
(7) Reduced the total cost of ownership.
(8) Increases productivity.
(9) As clients do not play a major role in this model, they require less administrator.
(10) Improved the security.

Disadvantages

(1) Traffic congestion on the network has been an issue since the inception of the client/server paradigm. As the number of simultaneous client requests to a given server, the server can become overloaded.
(2) The client/server paradigm lacks the robustness of a good peer to peer network. Under client/server, should a critical server fail, clients requests cannot be fulfilled.

5.8.6 Distributed Database System Design

The design of a distributed database system is a complex task. Therefore, a careful assessment of the strategies and objectives is required. Some of the strategies and objectives that are common to the most distributed database system design are as follows :

- **Data Fragmentation :** Which are applied to relational database system to partition the relations among network sites.
- **Data Allocation :** In which each fragment is stored at the site with optional distribution.

- **Data Replication :** Which increases the availability and improves the performance of the system.
- **Location Transparency :** Which enables a user to access data without knowing, or being concerned with, the site at which the data resides. The location of the data is hidden from the user.
- **Replication Transparency :** Meaning that when more than one copy of data exists, one copy is chosen while retrieving data and all other copies are updated when changes are being made.
- **Configuration Independence :** Which enables the organization to add or replace hardware without changing the existing software component of the DBMS. It ensures the expandability of existing system when its current hardware is saturated.
- **Non-homogeneity DBMS :** Which helps in integrating databases maintained by different DBMSs at different sites on different computers.

Data fragmentation data replication and data allocation are the most commonly used techniques that are used during the process of DDBS design to break up the database into logical units and storing certain data in more than one site.

5.8.7 Transaction Processing in Distributed Systems

Transaction System : Transaction procesing systems are systems with large database and a large number of concurrent users are executing database transaction.

Transaction Processing : In a distributed DBMS, a given transaction is submitted at some one site, but it can access data at other sites. When a transaction is submitted at some sites, the transaction manager at that site breaks it up into a collection of one or more subtransaction that execute at different sites.

There are two types of transaction.

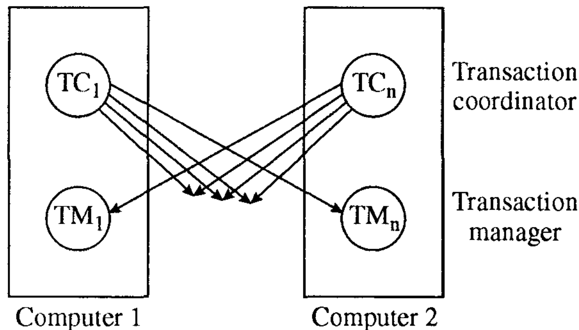
- **Local Transaction :** The local transactions are those that access and update data in only one local database.
- **Global Transactions :** The global transactions are those that access and update data in several local database.

5.8.7.1 System Structure : Each site has its own local transaction manager, whose function is to ensure the ACID properties of those transactions that execute at that site.

In transaction system each site contains two subsystems.

(i) Transaction Manager : The transaction manager manages the execution of those transaction that access data stored in a local site.

(ii) Transaction Coordinator : The transaction coordinator coordinates the execution of the various transactions initiated at that site.



Responsibility of Transaction Manager

These are following responsibility of each transaction manager :

- Maintaining a log for recovery purposes.
- Participating in an appropriate concurrency control scheme to coordinate the concurrent execution of the transactions executing at that site.

Responsibility of Coordinator

These are following responsibility of coordinator :

- Starting the execution of the transaction.
- Breaking the transaction into a number of subtransactions and distributing these subtransactions to the appropriate sites for execution.
- Coordinating the termination of the transaction, *i.e.*, either transaction may be committed at all sites or aborted at all sites.

5.8.7.2 System Failure Modes : A distributed system may suffer from the same types of failure that a centralized system does.

These are the following types of failure :

- Failure of a site.
- Loss of messages.
- Failure of a communication link.
- Network Partition.

5.8.8 Data Fragmentation

The system partitions the relation into several fragments and stores each fragment at a different site.

Suppose relation r is fragmented, r is divided in a number of fragments r_1, r_2, r_3 ...., r_n. These fragments contains sufficient information to allow reconstruction of the original relation r.

Types of Fragmentation : These are following fragmentation :

(1) Horizontal Fragmentation : The Horizontal fragmentation splits the relation by assigning each tuple (row) of r to one or more fragments. We can construct the fragmentation r_i by using selection.

$$r_i = \sigma_{P_i}(r)$$

where Pi is a predicate. We can reconstruct the relation r by taking the union of all fragments; that is,

$$r = r_1 U\ r_2 U \ldots\ldots\ldots U r_n$$

Example : **Relation Student**

Roll No.	Name	Address	City	State
1.	Vijay	H–327	Gr. Noida	U.P.
2.	Santosh	F–300	New Delhi	Delhi
3.	Gopal	L–62	Noida	U.P.
4.	Sanjay	M–129	New Delhi	Delhi

Fragment Student1

Roll No.	Name	Address	City	State
1.	Vijay	H–327	Gr. Noida	U.P.
2.	Santosh	F–300	New Delhi	Delhi

Fragment Student2

Roll No.	Name	Address	City	State
3.	Gopal	L–62	Noida	U.P.
4.	Sanjay	M–129	New Delhi	Delhi

(2) Vertical Fragmentation : Vertical fragmentation splits the relation by decomposition the schema R of the relation r.

Each fragmentation r_i of r is defined by

$$r_i = \pi_{R_i}(r)$$

where R is an attribute. The fragmentation should be done in such a way that we can reconstruct relation r from the fragments by taking the natural join.

$$r = r_1 \bowtie r_2 \bowtie \cdots\cdots\cdots \bowtie r_n$$

Example : **Relation Student**

Roll No.	Name	Address	City	State
1.	Vijay	H–327	Gr. Noida	U.P.
2.	Santosh	F–300	New Delhi	Delhi
3.	Gopal	L–62	Noida	U.P.
4.	Sanjay	M–129	New Delhi	Delhi

Fragment Student 1

Roll No.	Name	Address
1.	Vijay	H–327
2.	Santosh	F–300
3.	Gopal	L–62
4.	Sanjay	M–129

Fragment Student 2

Roll No.	City	State
1.	Gr. Noida	U.P.
2.	New Delhi	Delhi
3.	Noida	U.P.
4.	New Delhi	Delhi

Mixed Fragmentation : We can intermix the two types of fragmentation (Horizontal & Vertical Fragmentations) yielding mixed fragmentation.

The mathematically representation of mixed fragmentation is :

$$r_i = \pi_{R_i}(\sigma_{P_i}(r))$$

These are following cases arises.

Case 1 : if Pi ≠ True & Ri = ATTRS (R),
Then, we get a vertical fragment.

Case 2 : If Pi = True & Ri ≠ ATTRS (R),
Then, we get a horizontal fragment.

Case 3 : If Pi ≠ True & Ri ≠ ATTRS(R),
Then, we get mixed fragment.

5.8.9 Data Replication & Allocation

Data Replication means the copies of data. Data replication is a technique that permits storage of certain data in more than one site. The system maintains several identical replicas (copies) of the relation and store each copy at a different site. Typically, data replication is introduced to increase the

availability of the system when a copy is not available due to site failure(s), it should be possible to access another copy.

Fully Replicated Distributed Database : The replication of the whole database at every site in the distributed system, is called fully replicated distributed database.

Advantages : These are following advantages :

(i) This can improve the availability because the system can continue to operate as long as at least one site is up.

(ii) It also improves performance of retrieval for global queries, because the result of such a query can be obtained locally from any one site.

Disadvantages : The disadvantages of fully replication is that :

(i) It can slow down update operations drastically, because a single logical update must be performed on every copy of the database to keep the copies consistent.

(ii) Fully replication makes the concurrency control and recovery techniques more expensive than they would be if there were no replication.

- **Replication Schema :** A description of the replication of fragments is called a replication schema.
- **Partial Replication :** In partial Replication some fragments of the database may be replicated where as others may not.

5.8.10 Data Allocation

Data allocation describes the process of deciding about locating (or placing) data to several sites. Following are the data placement strategies that are used to distributed database systems.

- Centralised
- Partitioned or fragmented
- Replicated

In case of centralised strategies, entire single database and the DBMS is stored at one site. However, users are geographically distributed across the network. Locality of reference is lowest as all sites, except the central site, have to use the network for all data accesses. Thus, the communication costs are high. Since the entire database resides at one site, there is loss of the entire database system in case of failure of the central site. Hence, the reliability and availability are low.

In partitioned or fragmented strategy, database is divided into several disjoint parts (fragments) and stored at several sites. If the data items are located at the site where they are used most frequently, locality of reference is high. As there is no replication, storage costs are low. The failure of system at a particular site will result in the loss of data of that site. Hence, the reliability and availability are higher than centralised strategy.

However, overall reliability and availability are still low. The communication cost is low and overall performance is good as compared to centralised strategy.

In replication strategy, copies of one or more database fragments are stored at several sites. Thus, the locality of reference, reliability and availability and performance are maximized. But, the communication and storage costs are very high.

In data allocation or data distribution each copy of a fragment must be assigned to a particular site in the distributed system.

- **Non Redundant Allocation :** If each fragment is stored at exactly one site, then all fragments must be disjoints except for the repetition of primary keys among vertical or mixed fragments, this is called no redundant allocation.

The choice of sites, the degree of replication depend on the performance and availability goals of the system and on the types and frequencies of transactions submitted at each site.

For Example : If high availability is required and transactions can be submitted at any site and if most transactions are retrieval only, a fully replicated database is a good choice.

5.8.11 Overview of Concurrency Control

For concurrency control purposes, numerous problems arise in a distributed DBMS environment that are not encountered in a centralized DBMS environment.

These include the following :

- **Dealing with multiple copies of the data items :** The concurrency control method is responsible for maintaining consistency among these copies.
- **Failure of individual sites :** The DDBMS should continue to operate with its running sites if possible, when one or more individual sites fail.
- **Failure of Communication Link :** The system must be able to deal with failure of one or more of the communication links that connect the sites.
- **Distributed deadlock :** Deadlock may occur among several sites, so techniques for dealing with deadlocks must be extended to take this into account.
- **Distributed Commit :** Problems can arise with committing a transaction that is accessing databases stored on multiple sites if some sites fail during the commit process.
- **Distributed Concurrency Control Techniques :** Distributed concurrency control techniques must deal with these and other problems.

These are following techniques :

(1) **Locking Protocols :** The locking protocols can be used in a distributed environment. The lock manager deals with replicated data. We present possible schemes that are applicable to an environment where data can be replicated in several sites.

(a) **Single Lock-Manager Approach :** In the single lock-manager approach, the system maintains a single lock manager that resides in a single chosen site consider Si.

All lock and unlock requests are made at site Si.

- When a transaction needs to lock a data item, it sends a lock request to Si.
- The lock-manager determines whether the lock can be granted immediately. If the lock can be granted, the lock manager sends a message to that effect to the site at which the lock request was initiated. Otherwise, the requests is delayed until it can be granted.

Advantages : The scheme has following advantages :

(i) **Simple Implementation :** This scheme requires two messages for handling lock requests, and one message for handling unlock requests.

(ii) **Simple Deadlock Handling :** Since all lock and unlock requests are made at one site, the deadlock-handling algorithms can be applied directly to this environment.

Disadvantages : The disadvantages of the schemes are :

(i) **Bottleneck :** The site Si becomes a bottle neck, since all requests must be processed there.

(ii) **Vulnerability :** If the site Si fails, the concurrency controller is lost. Either processing must stop, or a recovery scheme must be used so that a backup site can take over lock management from Si.

(b) **Distributed lock Manager :** In which the lock-manager function is distributed over several sites.

Each site maintains a local lock manager whose function is to administer the lock and unlock requests for those data items that are stored in that sites.

When a transaction wishes to a lock data item X, which is not replicated and resides at site Si, a message is sent to the lock manager at site Si requesting a lock.

If data item X is locked in an incompatible mode, then the request is delayed until it can be granted.

Once it has determined that the lock request can be granted, the lock manager sends a message back to the initiator indicating that it has granted the lock requests.

Advantages :

- The simple implementation.
- Reduces the degree to which the coordinator is a bottleneck.

5.8.12 Distributed Recovery

The recovery process in distributed database is quite involved. We give only a very brief idea of some of the issue here.

In some cases it is quite difficult even to determine whether a site is down without exchanging numerous message with other sites.

For Example : Suppose that site X sends a message to site Y and expects a response from Y but does not receive it.

There are several possible explanations :

- The message was not delivered to Y because of communication failure.
- Site Y is down and could not respond.
- Site Y is running and sent a response, but the response was not delivered.

Another problem with distributed recovery is distributed commit. When a transaction is updating data at several sites, it cannot commit until it is sure that the effect of the transaction on every site cannot be lost.

This means that every site must first have recorded the local effects of the transactions permanently in the local site log on disk. The two phase commit protocol is often used to ensure the correctness of distributed commit.

5.8.13 Two-Phase Commit Protocol

This protocol assumes that one of the cooperating processes acts as a coordinators, and other processes are as cohorts.

- At the begining of the transaction, the coordinator sends a start transaction message to every cohort.

Phase I : At the Coordinator :

(1) The Coordinator sends a COMMIT REQUEST message to every cohort requesting the cohorts to commit.

(2) The Coordinator waits for replies from all the cohorts.

At Cohorts : On receiving the COMMIT-REQUEST message a cohort takes the following actions.

- If the transaction executing at the cohort is successful, it writes 'UNDO' and 'REDO' log on the stable storage and send on 'AGREED' message to coordinator.
- Otherwise, it sends an ABORT message to the coordinator.

Phase II : At the Coordinator :

(1) If all the cohorts reply AGREED and the coordinator also agrees, then the coordinator writes a 'COMMIT' record into the log. Then it sends a COMMIT message to all the cohorts. Otherwise, the coordinator sends an ABORT message to all the cohorts.

(2) The coordinator then waits for acknowledgments from each cohort.
(3) If an acknowledgment is not received from any cohort within a time out period, the coordinator resends the COMMIT/ABORT message to that cohort.
(4) If all the acknowledgement are received, the coordinator writes a COMPLETE record to the log.

At Cohorts :

(1) On receiving a COMMIT message, a Cohort release all the resources and lock held by it for executing the transaction, and sends an acknowledgment.
(2) On receiving an ABORT message, a cohort undoes the transaction using the UNDO log record, release all the resources and locks held by it for performing the transaction and send an acknowledgment.

5.8.14 Handling of Failures

The two-phase commit protocol responds in different ways to various types of failures.

(1) Failure of Participating Site : If the coordinator Ci detects that a site has failed, it takes these actions :

- If the site fails before responding with a ready T message to Ci, the coordinator assumes that it responded with an ABORT T message.
- If the site fails after the coordinator has received the READY T message from the site, the coordinator executes the rest of the commit protocol in the normal way, ignoring the failure of the site.

When a participating site Si recovers from a failure, it most examine its log to determine the fate of those transactions that were in the midst of execution when the failure occured.

We consider each of the possible cases.

- The log contains a < COMMIT T > record. In this case, the site executes REDO (T).
- The log contains an < ABORT T > record. In this case, the site executes UNDO (T).
- The log contains a < READY T > record. In this case, the site must consult Ci to determine the fate of T.
- The log contains no control records (ABORT, COMMIT, READY) concerning T.

(2) Failure of the Coordinator : If the coordinator fails in the midst of execution of the commit protocol for transaction T, then the participating sites must decide the fate of T.

In certain cases, the participating sites cannot decide whether to COMMIT or ABORT T, & therefore these sites must wait for the recovery of the failed coordinator.

- If an active site contains a < COMMIT T > record in its log, then T must be committed.
- If an active site contains an < ABORT T > record in its log, then T must be aborted.
- If some active site does not contain a < READY T > record in its log, then the failed coordinator Ci cannot have decided to commit T, because a site that does not have a < READY T > record in its log cannot have sent a READY T message to Ci. However, the coordinator may have decided to ABORT T, but not to COMMIT T. Rather than wait for Ci to recover, it is preferable to abort T.
- If none the preceding cases holds, then all active sites must have a < READY T > record in their log, but no additional control records such as < ABORT T > or < COMMIT T >.

Solved Problems

Q. 1. *Write down the method of write-ahead-logging mechanism for data recovery.* **(UPTU 2006)**

Ans. Write-Ahead Logging (WAL) : Write-Ahead Logging (WAL) is a standard approach to transaction logging.

WAL's central concept is that changes to data files (where tables and indexes reside) must be written only after those changes have been logged, that is when log records have been flushed to permanent storage.

If we follow this procedure, we do not need to flush data pages to disk on every transaction commit, because we know that in the event of a crash we will be to recover the database using the log.

- Any changes that have not been applied to the data pages will first be redone from the log records, this is roll forward recovery, also known as REDO.
- And then the changes made by uncommitted transactions will be removed from the data pages, this is roll backward recovery UNDO.

For example : Consider the following Write-Ahead Logging (WAL) protocol for a recovery Algorithm that requires both UNDO & REDO :

(1) The before image of an item cannot be over written by its after image in the database on disk until all UNDO-type log records for the updating transaction-upto this point in time have been force-written to disk.

(2) The commit operation of a transaction cannot be completed until all the REDO-type & UNDO-type log records for that transaction have been force-written to disk.

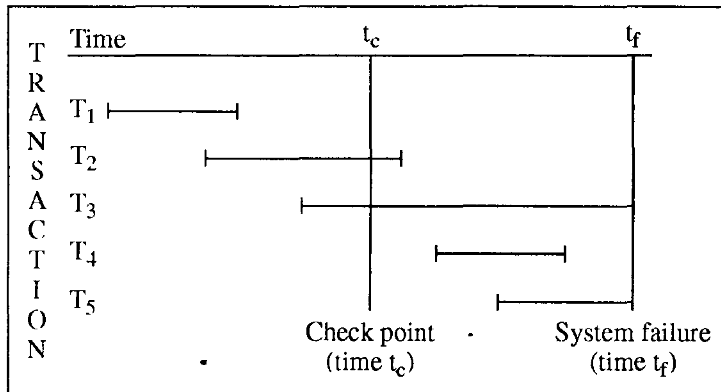
Advantages of WAL :

- Write-Ahead-Logging is techniques for providing atomicity and durability in database system.
- In a system using WAL, all modifications are written to a log before they are applied to the database. Usually both REDO and UNDO information is stored in the log.
- WAL is a significantly reduced number of disk writes, since only the log file needs to be flushed to disk at the time of transaction commit.
- The next advantage is consistency of the data pages.

WAL saves the entire data page content in the log if that is required to ensure page consistency for after crash recovery.

Q. 2. *Show how the backward recovery technique is applied to a DBMS ?* **(UPTU 2002, 03)**

Ans. Backward-Recovery Technique : At restart time, the system goes through the following procedure in order to identify all transactions of Type T_2 - T_5



Five transaction categories

(1) Start with two lists of transactions, the UNDO list and the REDO list. Set the UNDO list equal to the list of all transactions given in the most recent checkpoint record; set the REDO list to empty.
(2) Search forward through the log, starting from the checkpoint record.
(3) If a BEGIN TRANSACTION log entry is found for transaction T, add T to the UNDO list.
(4) If a COMMIT log entry is found for transaction T, move T from UNDO list to the REDO list.
(5) When the end of the log is reached the UNDO & REDO list identify, respectively, transaction of types T_3 & T_5 & transaction of types T_2 & T_4.

The system now works backward through the log, undoing the transactions in the UNDO-list. Restoring the database to a consistent state by undoing work is called Backward Recovery.

Q. 3. *Consider the following Transactions*

T_1 : read (A);
read (B);
if $A = 0$ then $B := B + 1$;
write (B);
T_2 : read (A);
read (B);
if $B = 0$ then $A := A + 1$;
write (A);

Add lock & unlock instruction to transactions T_1 & T_2, so that they observe the two phase locking protocol. **(UPTU 2003, 04)**

Ans. Lock-X = Exclusive lock
Lock-S = Shared lock

T_1	T_2
Lock – S (A)	
Lock – X (B)	
read (A)	
read (B)	
if A = 0 then B : = B + 1	
write (B)	
Unlock – S (A)	
Unlock – X (B)	
	Lock – S (B)
	Lock – X (A)
	read (A)
	read (B)
	if B = 0 then A : = A + 1
	write (A)
	Unlock – X (A)
	Unlock – S (B)

Q.4. *State whether the following schedule is conflict serializable or not. Justify your answer.*

(UPTU 2003, 04)

T_1	T_2
Read (A)	
Write (A)	
	Read (A)
	Write (A)
Read (B)	
Write (B)	
	Read (B)
	Write (B)

Ans. In this schedule, the write (A) of T_1 conflicts with the Read (A) of T_2, while write (A) of T_2 does not conflict with the Read (B) of T_1. Because these two instructions access different data items.

We can swap nonconflicting instructions.

- Swap Read (B) of T_1 with Read (A) of T_2.
- Swap write (B) of T_1 with write (A) of T_2.
- Swap write (B) of T_1 with Read (A) of T_2.

After the swapping the schedule.

T_1	T_2
Read (A)	
Write (A)	
	Read (A)
Read (B)	
	Write (A)
Write (B)	
	Read (B)
	Write (B)

The concept of conflict equivalence leads to the concept of conflict serializability.

Thus we can say that it is a conflict equivalence.

Hence it is the conflict serializable.

Q. 5. *Which of the following schedules is conflict serializable ? For each serializable schedule determine the equivalent serial schedules.*

(1) $r_1\ (x);\ r_3\ (x);\ w_1(x),\ r_2(x);\ w_3(x);$

(2) $r_1\ (x);\ r_3\ (x);\ w_3(x),\ w_1(x);\ r_2(x);$

(3) $r_3\ (x);\ r_2\ (x);\ w_3(x),\ r_1(x);\ w_1(x);$ **(UPTU 2004, 05)**

Ans. (1)

T_1	T_2	T_3
$r_1(x)$		
		$r_3(x)$
$w_1(x)$		
	$r_2(x)$	
		$w_3(x)$

We can swap the following :

- Swap $r_1(x)$ of T_1 with $r_3(x)$ of T_3.
- Swap $r_2(x)$ of T_2 with $r_3(x)$ of T_3.

Now after the swapping we get the schedule.

T_1	T_2	T_3
		$r_3(x)$
$r_1(x)$		
$w_1(x)$		
	$r_2(x)$	
		$w_3(x)$

Hence the given schedule is conflict serializable.

(2)

T_1	T_2	T_3
$r_1(x)$		
		$r_3(x)$
		$w_3(x)$
$w_1(x)$		
	$r_2(x)$	

We can swap

- Swap $r_1(x)$ of T_1 with $r_3(x)$ of T_3.

After the swapping the schedule.

T_1	T_2	T_3
		$r_3(x)$
$r_1(x)$		
		$w_3(x)$
$w_1(x)$		
	$r_2(x)$	

Thus the given schedule is conflict serializable schedule because we can swap $r_1(x)$ of T_1 with $r_3(x)$ of T_3.

(3)

T_1	T_2	T_3
		$r_3(x)$
	$r_2(x)$	
		$w_3(x)$
$r_1(x)$		
$w_1(x)$		

We can swap the following :

- Swap $r_3(x)$ of T_3 with $r_2(x)$ of T_2.
- Swap $r_2(x)$ of T_2 with $r_1(x)$ of T_1.

After swapping the schedule.

T_1	T_2	T_3
	$r_2(x)$	
		$r_3(x)$
		$w_3(x)$
$r_1(x)$		
$w_1(x)$		

Now the new serial schedule.

T_1, T_2 & T_3 are.

r_2 (x); r_3 (x); $w_3(x)$, $r_1(x)$; $w_1(x)$;

Hence the schedule is conflict serializable.

Q. 6. *Consider the precedence graph in given Fig. Is the corresponding schedule conflict serializable? Explain your Answer.* **(UPTU 2006, 07)**

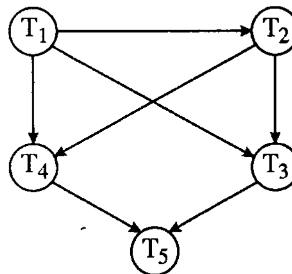


Fig. Precedence Graph

Ans. In the given precedence graph, the set of edges of graph holds one of the three conditions.

$T_1 \longrightarrow T_2$ (edge)

- T_1 executes write (Q) before T_2 executed read (Q)
- T_1 executes read (Q) before T_2 executed write (Q)
- T_1 executes write (Q) before T_2 executed write (Q)

An Edge $T_2 \longrightarrow T_3$

- T_2 executed write (Q) before T_3 executed read (Q)
- T_2 executes read (Q) before T_3 executed write (Q)
- T_2 executes write (Q) before T_3 executed write (Q)

An Edge $T_3 \longrightarrow T_5$

- T_3 executes write (Q) before T_5 executed read (Q).
- T_3 executes read (Q) before T_5 executed write (Q).
- T_3 executes write (Q) before T_5 executed write (Q).

An Edge $T_1 \longrightarrow T_4$

- T_1 executes write (Q) before T_4 executed read (Q).
- T_1 executes read (Q) before T_4 executed write (Q).
- T_1 executes write (Q) before T_4 executed write (Q).

An Edge $T_1 \longrightarrow T_3$

- T_1 executes write (Q) before T_3 executed read (Q).
- T_1 executes read (Q) before T_3 executed write (Q).
- T_1 executes write (Q) before T_3 executed write (Q).

An Edge $T_2 \longrightarrow T_4$: Since all instructions of T_2 are executed before the instruction of T_4 is executed.

Similarly : An Edge $T_4 \longrightarrow T_5$

Since all instructions of T_4 are executed before the instruction of T_5 is executed.

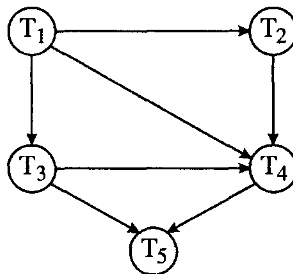
We know that

"If the precedence graph contain the cycle, then the schedule is not conflict serializable".

From the above description the precedence graph does not contain the cycle.

Hence corresponding schedule of the precedence graph is conflict serializable.

Q. 7. *Consider the precedence graph in Fig. Is the corresponding schedule conflict serializable ? Explain your answer.* **(UPTU 2004, 05)**



Ans. The set of edges of graph holds one of the three conditions.

$T_i \longrightarrow T_j$ edges

- T_i executes write (Q) before T_j executed read (Q).
- T_i executes read (Q) before T_j executed write (Q).
- T_i executes write (Q) before T_j executed write (Q).

An edge $T_1 \longrightarrow T_2$ edges : Since all instructions of T_1 are executed before the instructions of T_2 is executed.

An edge $T_1 \longrightarrow T_3$: Since all instructions of T_1 are executed before the instructions of T_3 is executed.

An edge $T_1 \longrightarrow T_4$: Since all instructions of T_1 are executed before the instructions of T_4 is executed.

An edge $T_2 \longrightarrow T_4$: Since all instructions of T_2 are executed before the instructions of T_4 is executed.

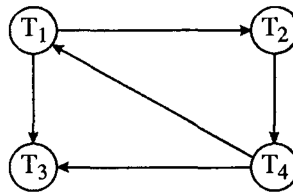
An edge $T_3 \longrightarrow T_5$: Since all instructions of T_2 are executed before the instructions of T_5 is executed.

From the above descriptions, the given precedence graph does not contain a cycle.

Since if a precedence graph contain a cycle, then the corresponding schedule is not conflict serializable.

Hence the corresponding schedule is conflict serializable.

Q. 8. *Consider the precedence graph in Fig. Is the corresponding schedule conflict serializable? Explain your answer.*



Ans. The set of edges of graph holds one of the three conditions.

An edge $T_i \longrightarrow T_j$ edges

- T_i executes write (Q) before T_j executed read (Q).
- T_i executes read (Q) before T_j executed write (Q).
- T_i executes write (Q) before T_j executed write (Q).

An edge $T_1 \longrightarrow T_2$: Since all instructions of T_1 are executed before the instructions of T_2 is executed.

An edge $T_1 \longrightarrow T_3$: Since all instructions of T_1 are executed before the instructions of T_3 is executed.

An edge $T_2 \longrightarrow T_4$: Since all instructions of T_2 are executed before the instructions of T_4 is executed.

An edge $T_4 \longrightarrow T_1$: All the instructions of T_4 are executed before the instructions of T_1 is executed.

Since the precedence graph contains a cycle between the T_1, T_2 & T_4 hence the corresponding schedule is not conflict serializable.

Q.9. *Explain 'Blind-Writes' operation with help of example.*

Ans.

T_1	T_2	T_3
Read (A)		
	Write (A)	
Write (A)		
		Write (A)

In above schedule, transactions T_2 & T_3 perform write (A) operations without performed a read (A) operation. Writes of this sort are called blind writes.

Note : Blind writes appear in any view-serializable schedule but not in conflict serializable.

Q. 10. *Consider the two relations.*

$$R_1 \, (A, B, C, D)$$
$$R_2 \, (C, D, E, F)$$

The size of relation R_1 is 20,000 tuples having 10 records/block of secondary storage & the size of R_2 is 10,000 tuples having 25 records/block of secondary storage. The buffer allocated to R_1 is 10 & to R_2 is 20.

Find the number of block transfers that may be needed to compute the join of these relation. Assume any method of joining, but state the method assumed.

Ans. Given :

$$R_1 \, (A, B, C, D)$$
$$R_2 \, (C, D, E, F)$$

Size of R_1 = 20,000 tuples
bf R_1 = 10
Size of R_2 = 10,000 tuples
bf R_2 = 25
b (buffer size available) = b_1 = 10
b_2 = 20

$$\text{Total no. of block} = \left[\frac{R_1}{bf\,R_1}\right] + \left[\frac{1}{b_1} \times \frac{R_1}{bf\,R_1} \times \frac{1}{b_2} \times \frac{R_2}{bf\,R_2}\right]$$

$$= \left[\frac{20000}{10}\right] + \left[\frac{1}{10} \times \frac{20000}{10} \times \frac{1}{20} \times \frac{10000}{25}\right]$$

$$= 2000 + 4000$$
$$= 6000 \text{ blocks}$$

The size of natural join of R_1 & $R_2 \leq |R_1| * |R_2|$

$$= 20000 * 10000$$

Q. 11. *Estimation of cost and optimization of tuple transfer for join in distributed database.*

(UPTU 2006, 07)

Ans. In a distributed system, several additional factors further complicate query processing. The first is the cost of transferring data over the network. This data includes intermediate files that are transferred to other sites for further processing, as well as the final result files that may have to be transferred to the site where the query result is needed.

Although these are connected via a high performance local area network, they become quite significant in other types of networks.

Hence, DDBMS query optimization algorithms consider the goal of reducing the amount of data transfer as an optimization criterion in choosing a distributed query execution strategy.

We illustrate this with simple example query.

Suppose that the EMPLOYEE and DEPARTMENT relations are distributed in figure.

SITE 1

EMPLOYEE

FNAME	MNAME	LNAME	SSN	BDATE	ADDRESS	SEX	SALARY	DNO

10,000 records

Each record is 100 bytes long SSN field is 9 bytes long FNAME field is 15 bytes long, LNAME is 15 bytes long, DNO field is 4 bytes long

SITE 2

DEPARTMENT

DNAME	DNUMBER	MGRSNN	MGRSTARTDATE

100 records, each record is 35 bytes long DNUMBER field is 4 bytes long, DNAME is 10 bytes long, MGRSSN field is 9 bytes long.

We will assume in this example that neither relation is fragmented. According to figure, the size of the EMPLOYEE relation is $100 \times 10{,}000 = 10^6$ bytes, and the size of the DEPARTMENT relation is $35 \times 100 = 3500$ bytes.

Consider the query Q

"For each employee, retrieve the employee name and the department name of which the employee works"

$\pi_{\text{FNAME, MNAME, LNAME, DNAME}}$ (EMPLOYEE $\bowtie_{\text{DNO=DNUMBER}}$ DEPARTMENT)

The result of this query will include 10,000 records, assuming that every employee is related to a department. Suppose that each record in the query result is 40 bytes long.

The query is submitted at a distinct site 3, which is called the result site because the query result is needed there. Neither the EMPLOYEE nor the DEPARTMENT relations reside at site 3.

These are three simple strategies for executing this distributed query :

1. Transfer both the EMPLOYEE and the DEPARTMENT relation to the result site, and perform the join at site 3.

   In this case total of 1,000,000 + 3500 = 1,003,500 bytes must be transferred.

2. Transfer the EMPLOYEE relation to site 2 execute the join at site 2, and send the result to site 3.

   The size of the query result is $40 \times 10{,}000 = 400{,}000$ bytes
   so 400,000 + 1,000,000 = 1,400,000 bytes
   must be trnsferred.

3. Transfer the DEPARTMENT relation to site 1, execute the join at site 1, and send the result to site 3.

   In this case 400,000 + 3500 = 403,500 bytes must be tranferred.

   In minimizing the amount of data transfer is our optimzation criterion, we should choose strategy 3.

Now consider another query Q^1 :

"For each department, retrieve the department name and the name of the department manager"

The query Q^1 :

$\pi_{\text{FNAME, MNAME, LNAME, DNAME}}$ DEPARTMENT $\bowtie_{\text{MGRSSN = SSN}}$ EMPLOYEE)

Suppose that the result site is site 2, then we have two simple strategies :

1. Transfer the EMPLOYEE relation to site 2, execute the query, and present the result to the user at site 2. Here the same number of bytes 1,000,000 must be transferred for both Q and Q^1.
2. Transfer the DEPARTMENT relation to site 1, execute the query at site 1, and send the result back to site 2.
   In this case 400,000 + 3500 = 403,500 bytes must be transferred for Q and 4000 + 3500 = 7500 bytes for Q^1.

Review Questions

1. What do you mean by concurrent execution of transaction? What are locks? Explain two phase locking protocol in brief.
2. What is concurrency control? What are its objectives?
3. What are the different types of locks?
4. What is a timestamp? Discuss the timestamp ordering techniques for concurrency control.
5. List the salient features of two-phase locking protocol. Prove that two-phase locking ensures serializability.
6. Consider the following transactions :

   T_1 : Read (A)
   Read (B)
   If A = 0 then B : = B + 1
   Write (B)

   T_2 : Read (B)
   Read (A)
   If B = 0 then A : = A + 1
   Write (A)

   (a) Add lock and unlock instructions to transactions T_1 and T_2, so that they observe the two-phase locking protocol.
   (b) Can the execution of these transactions result in a deadlock?
7. What is multiple-granularity locking? What is the difference between implicit and explicit locking in multiple-granularity locking?
8. What is difference between exclusive lock and shared lock? Explain with example.
9. Explain the concurrency control scheme based on timestamping protocol.
10. Explain the validation concurrency control techniques.
11. What do you mean by multiversion scheme? Explain, what are W-time stamp and R-time stamp of Nth version of data object.
12. What is the optimistic method of concurrency control? Discuss the different phases through which a transaction moves during optimistic control.
13. List the advantages, problems and applications of optimistic method of concurrency control.
14. What is distributed database? Explain with a neat diagram.
15. What are the main advantages and disadvantages of distributed databases.

16. What do you mean by architecture of a distributed database system? What are different types of architecture? Discuss each of them with neat sketch.
17. What are the various types of distributed databases? Discuss in detail.
18. What are homogeneous DDBS? Explain in details.
19. What are heterogeneous DDBS? Explain in details.
20. What is a fragmenting a relation? What are the main types of data fragments? Why is fragmentation a useful concept in distributed database design?
21. What is horizontal data fragmentation? Explain with an example.
22. What is vertical data fragmentation? Explain with an example.
23. What is mixed data fragmentation? Explain with an example.
24. What is data replication? Why is data replication useful in a DDBMS? What typical units of data replicated?
25. What is data allocation? Discuss.
26. What do you mean by data replication? What are its advantages and disadvantages?
27. Explain the difference among following :
    (a) Homogeneous DDBMS and heterogeneous DDBMS.
    (b) Horizontal fragmentation and vertical fragmentation.
28. Explain the difference among fragmentation, transparency, replication transparency and location transparency.
29. Consider a failure occurs during two-phase commit for a transaction in distributed environment. Explain two-phase commit protocol.
30. Explain the term transaction system. How is the transaction processing done in distributed database?
31. Explain the algorithm : Read-before-write protocol.
32. Write a short notes on the following :
    (a) Distributed database (b) Data fragmentation
    (c) Data allocation (d) Data replication
    (e) Timestamping (f) Two-phase commit protocol.