Unit 6

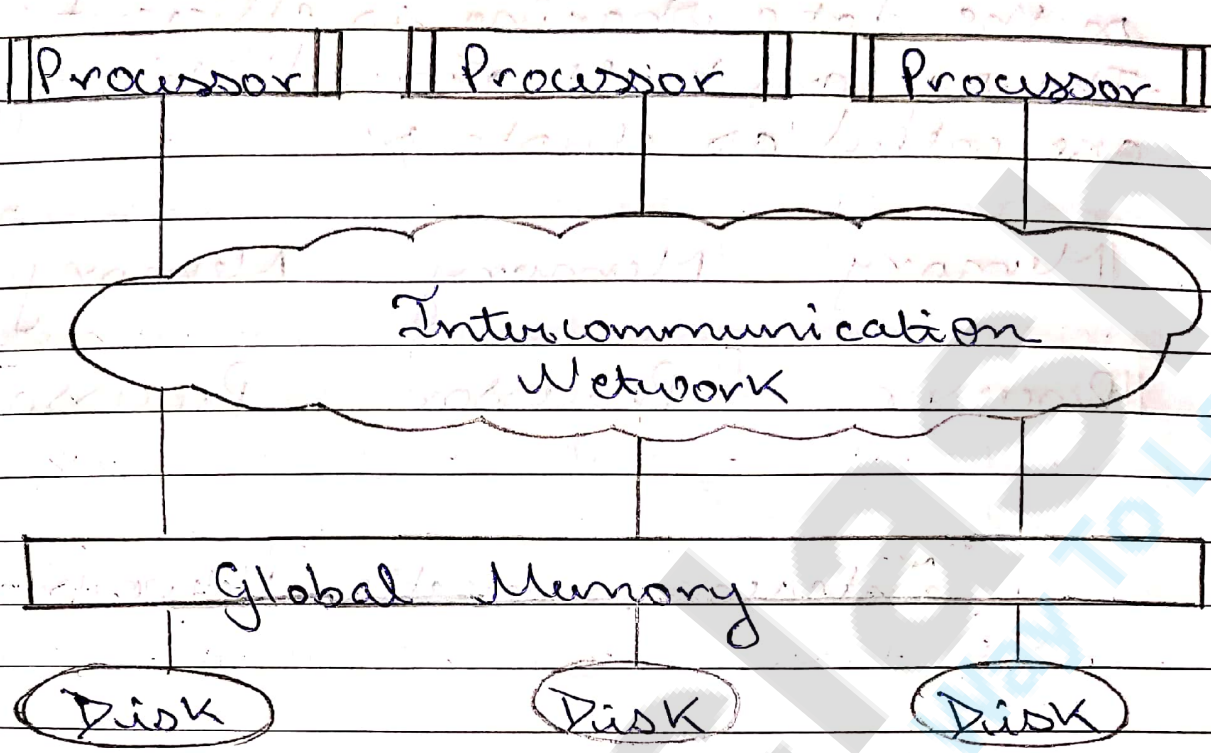
Q] Architecture for Parallel Databases

Shared memory system

Shared memory system uses multiple processors which is attached to a global shared memory via intercommunication channel or communication bus. Shared memory system have large amount of cache memory at each processors, so referencing of the shared memory is avoided.

If a processor performs a write operation to memory location, the data should be updated or removed from that

location.



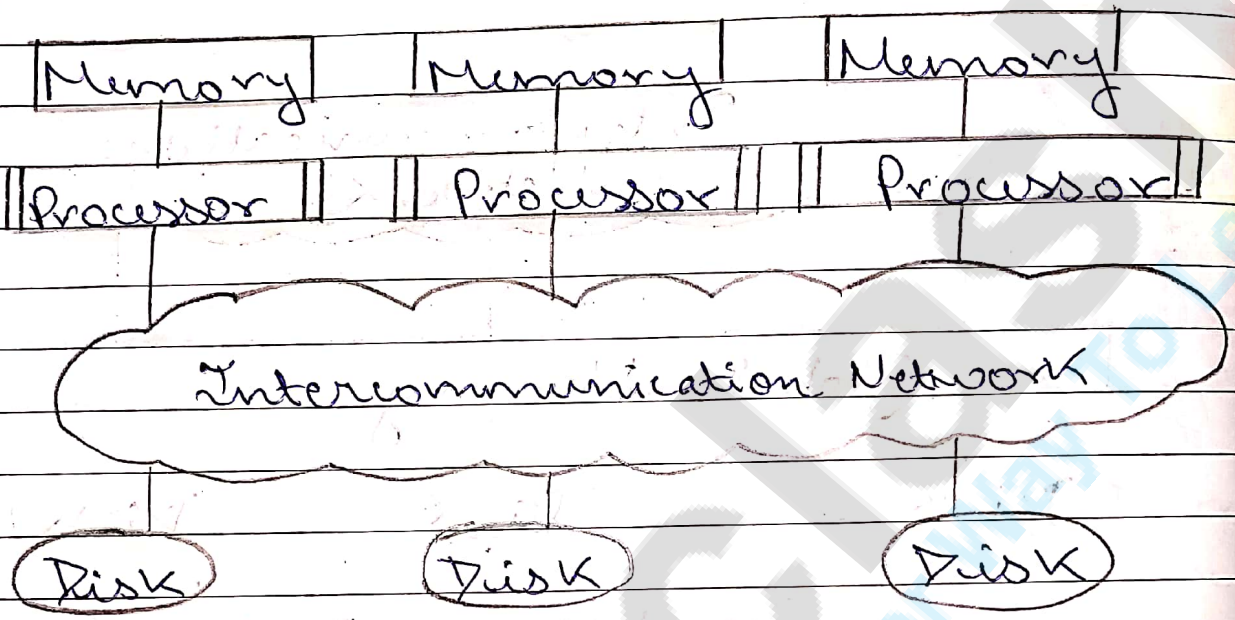
Shared Memory System in Parallel Databases

Advantages of shared memory system
- Data is easily accessible to any processor.
- One processor can send message to other efficiently.

Disadvantages of shared memory system
- Waiting time of processors is increased due to more number of processors.
- Bandwidth problem.

Shared Disk System
Shared disk system uses multiple processors which are accessible to multiple disks via intercommunication channel and every processor has local memory.

- Each processor has its own memory so the data sharing is efficient.
- The system built around this system are called as clusters.



Shared disk system is Parallel Databases

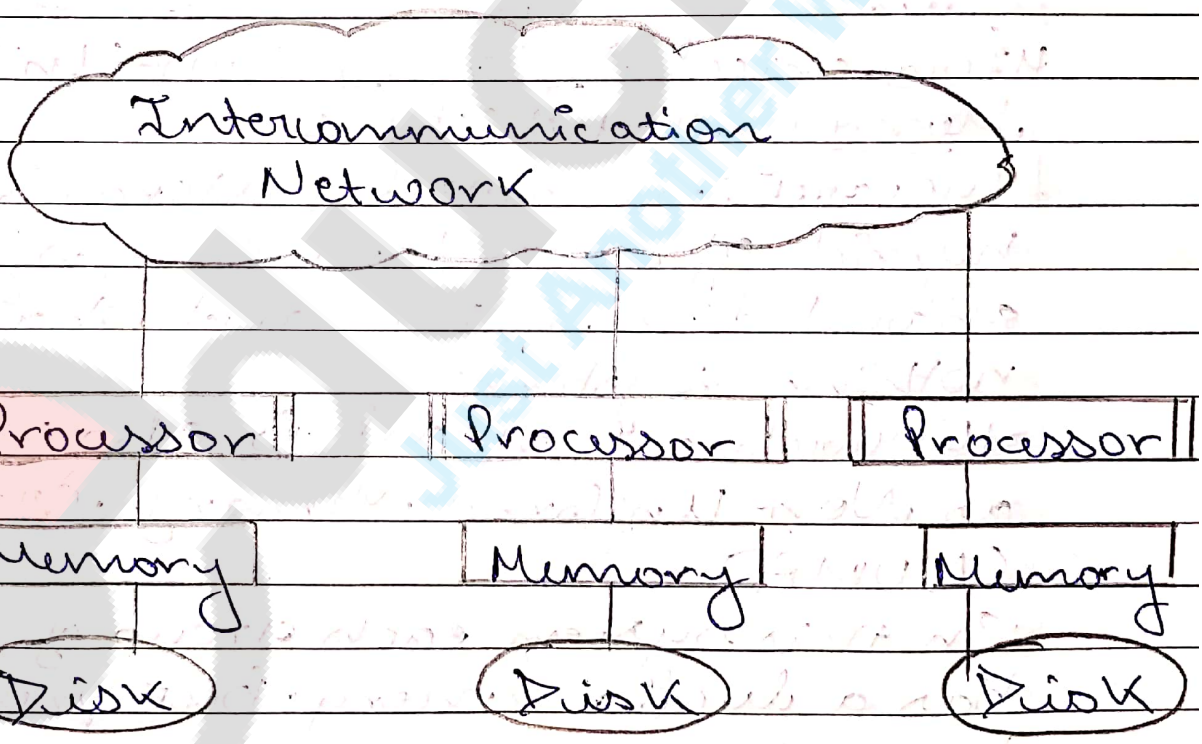
Advantages of shared Disk System
- Fault tolerance is achieved using shared disk system.
If a processor or its memory fails, the other processor can complete the task. This is called as fault tolerance.

Disadvantages of Shared Disk System
Shared disk System has limited scalability as large amount of data travels through the interconnection channel.
If more processors are added the

existing processors are slowed down.

Shared nothing disk system

- Each processor in the shared nothing system has its own local memory and local disk.
- Processors can communicate with each other through intercommunication channel.
- Any processor can act as a server to serve the data which is stored on local disk.



Shared nothing disk system in Parallel Databases

Advantages of Shared nothing disk system:-

- Number of processors and disk can be connected as per the

requirement in share nothing disk system.
Shared nothing disk system can support for many processor, which makes the system more scalable.

Disadvantages of Shared nothing disk system
Data partitioning is required in shared nothing disk system
Cost of communication for accessing local disk is much higher.

Hierarchical System or Non-Uniform Memory Architecture
- Hierarchical modul system is a hybrid of shared memory system, shared disk system and shared nothing system.
- Hierarchical modul is also known as Non-Uniform Memory Architecture (NUMA).
- In this system each group of processor has a local memory. But processors from other groups can access memory which is associated with the other group in coherent.
NUMA uses local and remote memory (Memory from other group). Hence it will take longer time to communicate with each other.

PAGE NO.
DATE / /

Advantages of NUMA

- Improves the scalability of the system.
- Memory bottleneck (shortage of memory) problem is minimized in this architecture.

Disadvantages of NUMA

The cost of the architecture is higher compared to other architecture.

Q) Parallel Query Evaluation

The two techniques used in query evaluation are as follows:-

1) Inter query parallelism

This technique allows to run multiple queries on different processors simultaneously.

Pipelined parallelism is achieved by using inter query parallelism, which improves the output of the system.

For example:- If there are 6 queries, each query will take 3 seconds for evaluation. Thus, the total time taken to complete evaluation process is 18 seconds. Inter query parallelism achieves this task only in 3 seconds. However, Inter query parallelism is difficult to achieve every time.

2) Intra Query Parallelism

In this technique query is divided

ins sub queries which can run simultaneously on different processors, this will minimize the query evaluation time.
Intra query parallelism improves the response time of the system.
For example - If we have 6 queries, which can take 3 seconds to complete the evaluation process, the total time to complete the evaluation process is 18 seconds. But we can achieve this task in only 3 seconds by using intra query evaluation as each query is divided in sub-queries.

Optimization of Parallel Query
- Parallel Query optimization is nothing but selecting the efficient query evaluation plan.
- Parallel Query optimization plays an important role in developing system to minimize the cost of query evaluation.

Two factors play a very important role in parallel query optimization
a) total time spent to find the best plan.
b) amount of time required to execute the plan.

Goals of Query optimization

PAGE No.
DATE / /

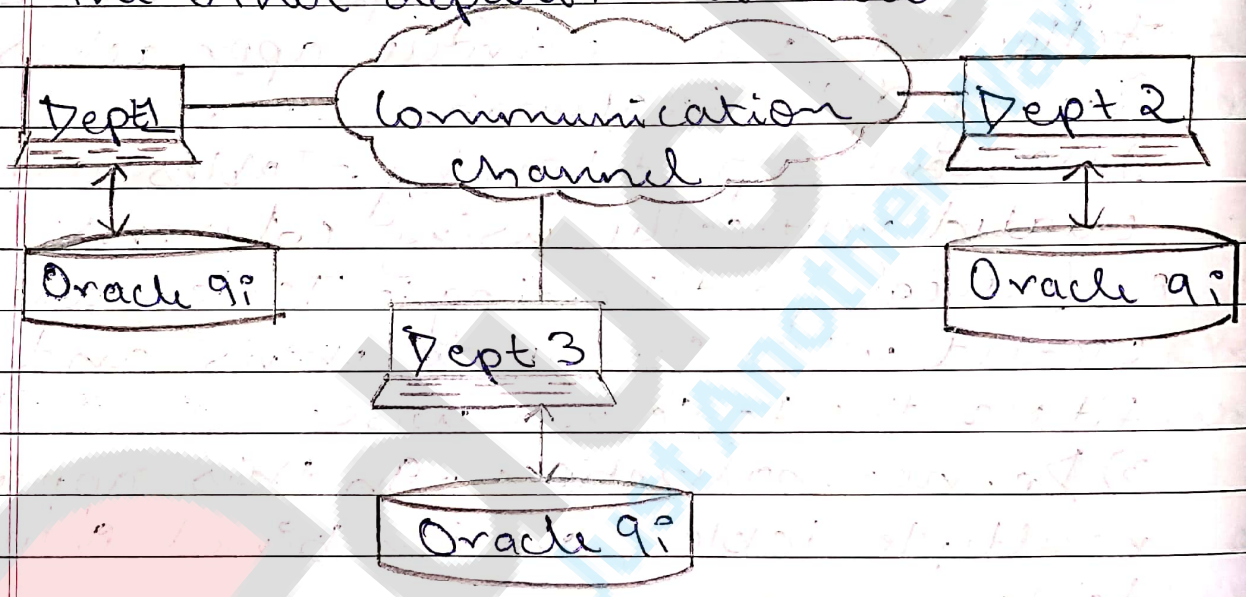
Query Optimization is done with an aim to:-
- Speed up the queries by finding the queries which can give the fastest result on execution.
- Increase the performance of the system.
- Select the best query evaluation plan
- Avoid the unwanted plan.

Approaches of Query Optimization:-
Following are the three approaches to Query Optimization:-
1) Horizontal partitioning:- Tables are created vertically using columns.
2) Vertical partitioning:- Tables are created with fewer columns and partition the table row wise.
3) De-normalization - In this approach multiple tables are combined into one table.

Q) Types of Distributed Databases:
The two types of distributed systems are as follows:
1) Homogenous distributed databases system:
- Homogenous distributed database system is a network of two or more databases (with same type of DBMS software) which can be stored

on one or more machines.
- So, in this system data can be accessed and modified simultaneously on several databases in the network. Homogeneous distributed system are easy to handle.

Example :- Consider that we have three departments using Oracle-9i for DBMS. If some changes are made in one department then, it would update the other department also.



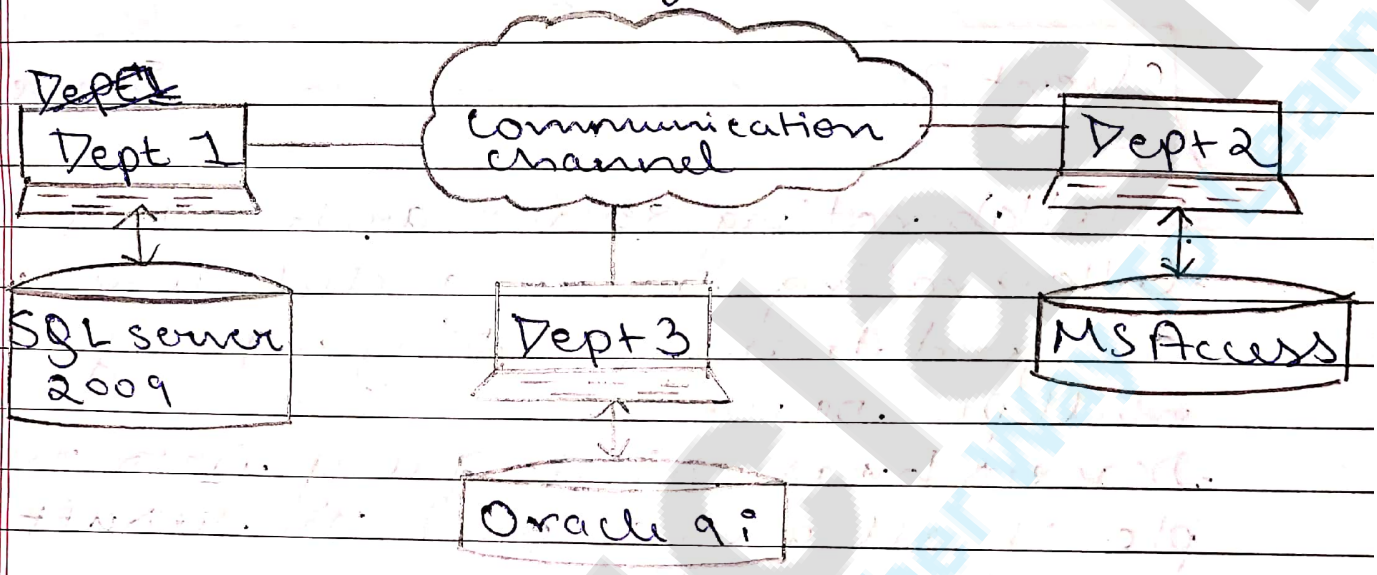
Homogeneous distributed system

2) Heterogeneous distributed database system.

- Heterogeneous distributed database system is a network of two or more databases with different type of DBMS software, which can be stored on one or more machines.
- In this system data can be accessible to several databases in the

network with the help of generic connectivity (ODBC and JDBC).

Example: In the following diagram, different DBMS software are as accessible to each other using ODBC and JDBC.



Heterogeneous distributed system.

Q) Distributed DBMS Architecture

The basic types of distributed DBMS are as follows:-

1) Client-server architecture of distributed system.

A client server architecture has a number of clients and a few servers connected in a network.

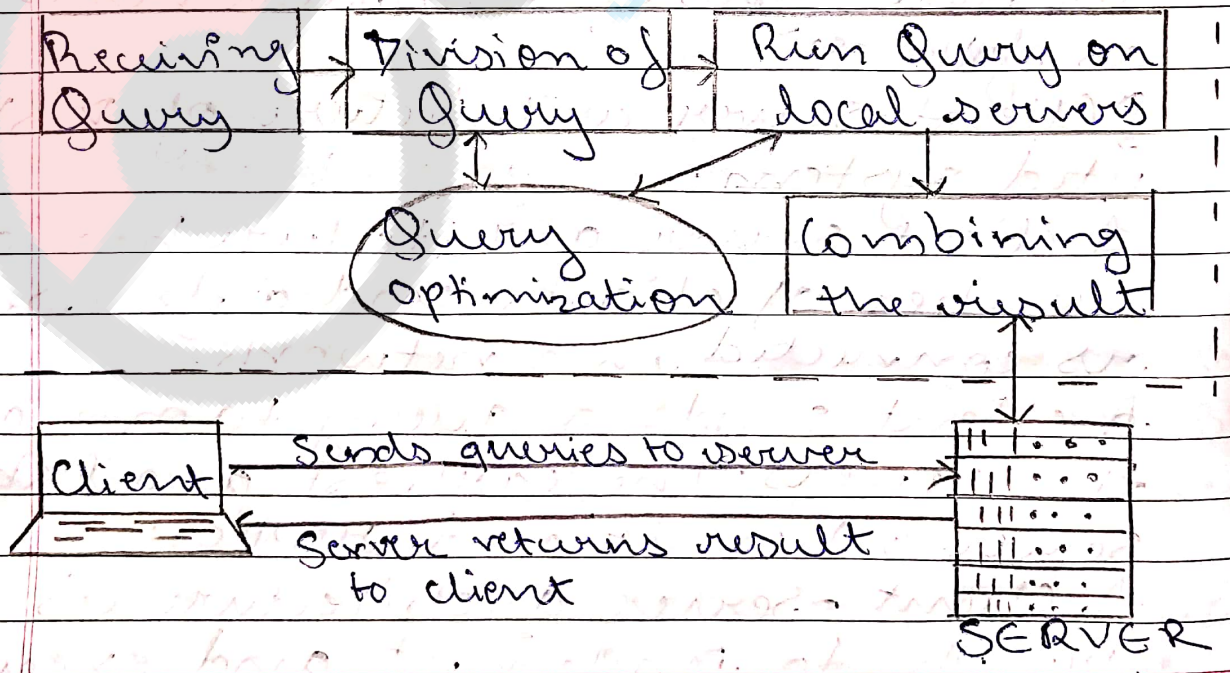
A client sends a query to one of the servers. The earliest available server solves it and replies.

A client-server architecture is simple to implement and execute

due to centralized server system.

2) Collaborating server architecture

- Collaborating Server architecture is designed to run a single query on multiple servers.
- Servers break single query into multiple small queries and the result is sent to the client.
- Collaborating server architecture has a collection of database servers. Each server is capable for executing the current transaction across the databases.



Collaborating server architecture

3) Middleware architecture

Middleware architecture are designed in such a way that single query is executed on multiple servers.

- This system needs only one server which is capable of managing queries and transactions from multiple servers
- Middleware architecture uses local servers to handle local queries and transactions.
- The softwares are used for execution of queries and transactions across one or more independent databases servers, this type of software ~~was~~ is called as middleware.

Q) Storing Data in a Distributed DBMS

There are 2 ways in which data can be stored on different sites. These are:

1) Replication

In this approach, the entire relation is stored redundantly at 2 or more sites. If the entire database is available at all sites, it is a fully redundant database. Hence, in replication, systems maintain copies of data.

This is advantageous as it increases the availability of data at different sites. Also, now query requests can be processed in parallel.

However, it has certain disadvantages

as well. Data needs to be constantly updated. Any change made at one site needs to be recorded at every site that relation is stored or else it may lead to inconsistency. This is a lot of overhead. Also, concurrency control becomes way more complex as concurrent access now needs to be checked over a number of sites.

2) Fragmentation

In this approach, the relations are fragmented (i.e, they're divided into smaller parts) and each of the fragments is stored in different sites where they're required. It must be made sure that the fragments are such that they can be used to reconstruct the original relation (i.e, there isn't any loss of data). Fragmentation is advantageous as it doesn't create copies of data, consistency is not a problem.

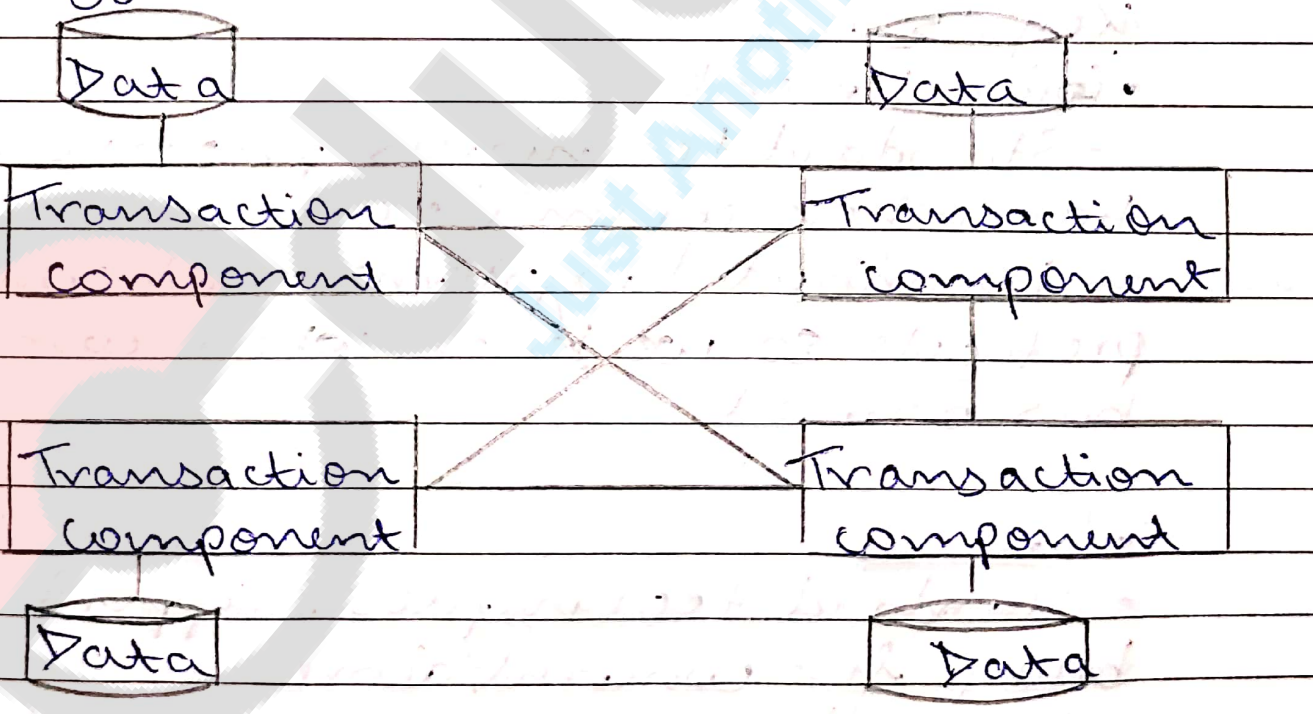
Fragmentation of relations can be done in two ways:

1) Horizontal fragmentation - Splitting by rows - The relation is fragmented into groups of tuples so that each tuple is assigned to at least one fragment.

2) Vertical fragmentation - Splitting by columns - The schema of the relati

on is divided into smaller schemas. Each fragment must contain a common candidate key so as to ensure lossless join.
In certain cases, an approach that is hybrid of fragmentation and replication is used.

(g) Distributed Transactions
- Must still support ACID properties
- Important for all the same reasons
- Some properties are harder to implement
- Basic single-system techniques are not sufficient



Why are Distributed Transactions Hard?
- Atomic
  - Different parts of a transaction may be at different sites
  - How do we ensure all or none

committed?

- Consistent
  - Failure may affect only part of transaction

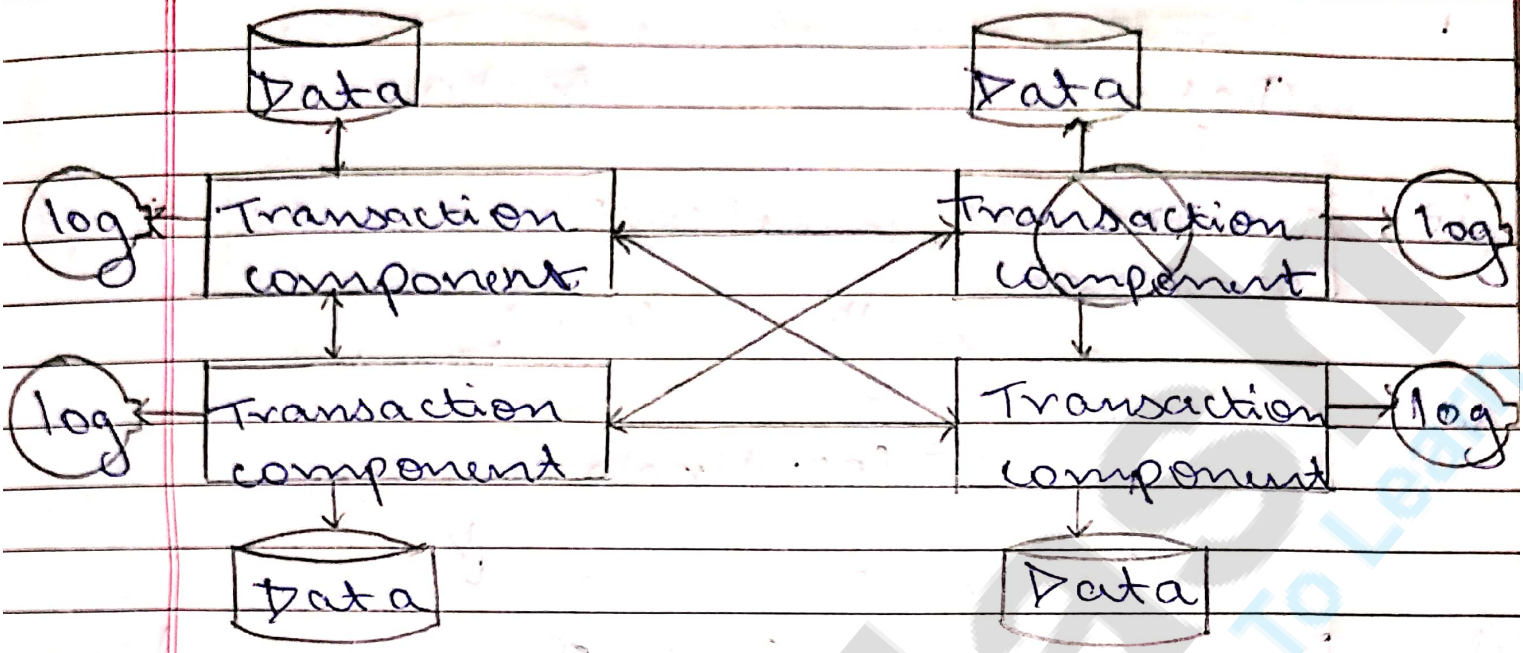
- Isolated
  - Commitment must occur "simultaneously" at all sites

- Durable
  - Not much different when other problems solved
  - Makes "delayed commit" difficult

Key Issues
- Commitment
  - Standard techniques preserve properties when commit occurs
  - Distributed systems need commit protocols so we know when commit has occured

- Failures
  - Standard techniques support durability for commit/abort
  - What happens if a site fails during commitment?

Committing a Distributed Transaction



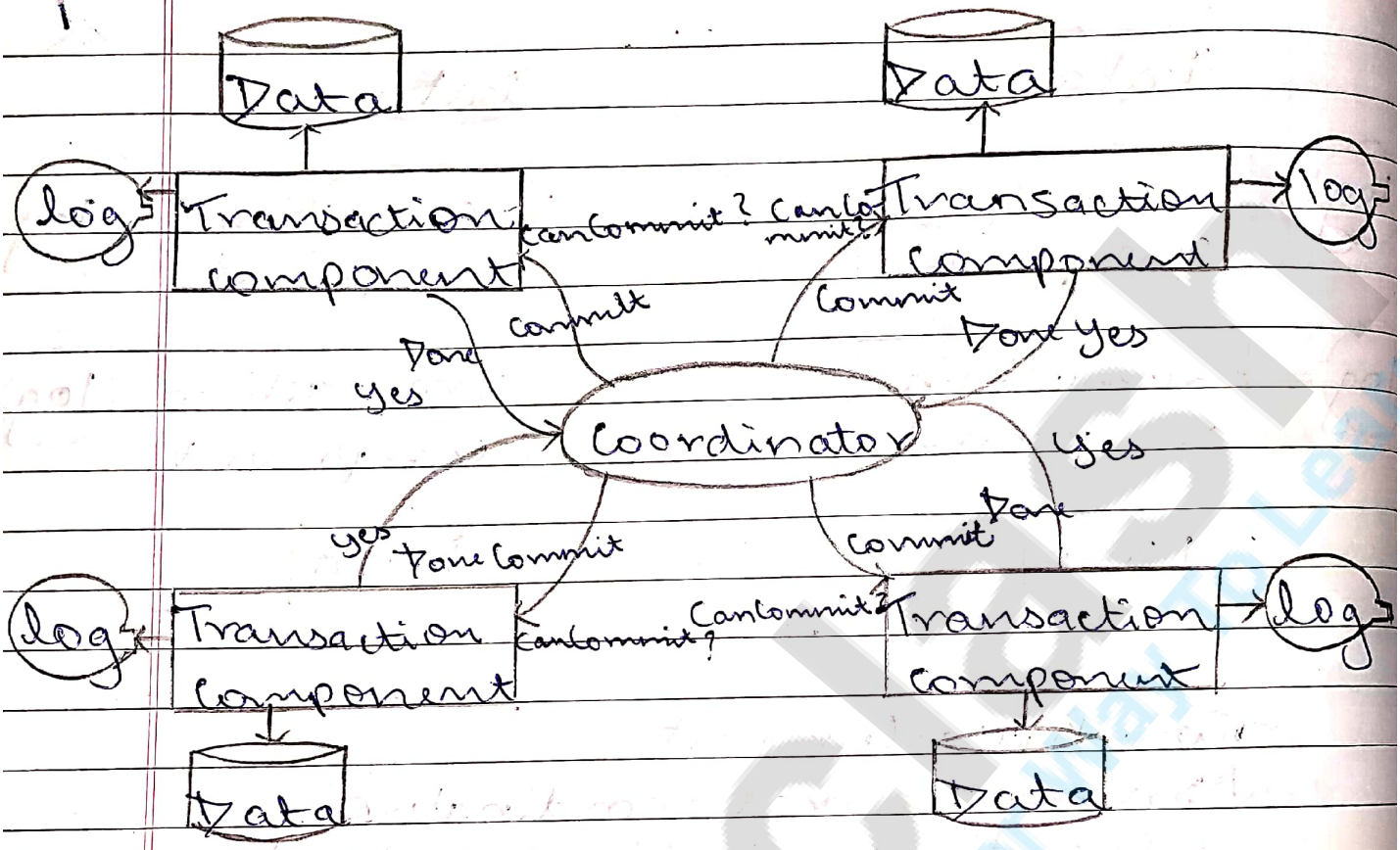
Two-Phase Commit

- Assumes central coordinator
  - Coordinator initiates protocol
  - Participants: entities with actions to be committed/aborted

- Phase 1:
  - Coordinator asks if participants can commit
  - Participants repond yes/no

- Phase 2:
  - If all votes yes, coordinator sends commit.
    Otherwise send Abort
  - Participants sends Have Committed/ Have Aborted.

Two-Phase Commit diagram representation



Two-Phase Commit: Fault Tolerance

- Participant fails in Phase 1:
  - Coordinator doesn't get unanimous yes
  - Abort
- Participant fails in Phase 2:
  - On reawakening, can ask coordinator if it should commit or abort
  - Requires that both commit and abort states be durable before sending "yes" vote

Two-Phase Commit: Problems

- Blocks on failure
  - Timeout before abort if participant fails
  - All participants must wait for

recovery if coordinator fails.
- While blocked, transaction must remain isolated
  - Hold locks on data items touched
  - Prevents other transactions from completing

Q] Distributed Concurrency Control
Distributed Two-phase Locking Algorithm
The basic principle of distributed two-phase locking is same as the basic two-phase locking protocol. However, in a distributed system there are sites designated as lock managers. A lock manager controls lock acquisition requests from transaction monitors. In order to enforce co-ordination between the lock managers in various sites, at least one site is given the authority to see all transactions and detect lock conflicts.
Depending upon the number of sites who can detect lock conflicts, distributed two-phase locking approaches can be of three types -
- Centralized two-phase locking - In this approach, one site is designated as the central lock manager. All the sites in the environment know the location of the central lock manager and obtain lock from it during transactions.

• Primary copy two-phase locking - In this approach, a number of sites are designated as lock control centers. Each of these sites has the responsibility of managing a defined set of locks. All the sites know which lock control center is responsible for managing lock of which data table/fragment item.

• Distributed two-phase locking - In this approach, there are a number of lock managers, where each lock manager controls locks of data items stored as at its local site. The location of the lock manager is based upon data distribution and replication.

Distributed Timestamp Concurrency Control :-

In a centralized system, timestamp of any transaction is determined by the physical clock reading. But, in a distributed system, any sites local physical/logical clock readings cannot be used as global timestamps, since they are not globally unique. So, a timestamp comprises of a combination of site ID and that site's clock reading.

For implementing timestamp ordering algorithms, each site has a scheduler that maintains a separate queue

for each transaction manager. During transaction, a transaction manager sends a lock request to the site's scheduler. The scheduler puts the request to the corresponding queue in increasing timestamp order. Requests are processed from the front of the queues in the order of their timestamps i.e the oldest first.

Conflict Graphs.

Another method is to create conflict graphs. For this transaction classes are defined. A transaction class contains two set of data items called read set and write set. A transaction belongs to a particular class if the transaction's read set is a subset of the class read set and the transaction's write set is a subset of the class' write set. In the read phase, each transaction issues its read requests for the data items in its read set. In the write phase, each transaction issues its write requests.

A conflict graph is created for the classes to which active transactions belong. This contains a set of vertical, horizontal and diagonal edges. A vertical edge connects two nodes within a class and denotes conflicts within the class. A horizontal edge

connects two nodes across two classes and denotes a write-write conflict among different classes. A diagonal edge connects two nodes across two classes and denotes a write-read or a read-write conflict among two classes.
The conflict graphs are analyzed to ascertain whether two transactions within the same class or across two different classes can be run in parallel.

Distributed Optimistic Concurrency Control Algorithm

Distributed optimistic concurrency control algorithm extends optimistic concurrency control algorithm. For this extension, two rules are applied.
Rule 1 - According to this rule, a transaction must be validated locally at all sites when it executes. If a transaction is found to be invalid at any site, it is aborted. Local validation guarantees that the transaction maintains serializability at the sites where it has been executed. After a transaction passes local validation test, it is globally validated.
Rule 2 - According to this rule, after a transaction passes local

validation test, it should be globally validated. Global validation ensures that if two conflicting transactions run together at more than one site, they should commit in the same relative order at all the sites they run together. This may require a transaction to wait for the other conflicting transaction, after validation before commit. This requirement makes the algorithm less optimistic since a transaction may not be able to commit as soon as it is validated at a site.

9) Distributed Recovery

In order to recuperate from database failure, database management systems 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. However, sometimes, transaction redo may also be adopted to recover to a consistent state of the transaction.
- In case of hard failures resulting in extensive damage to database,

recovery strategies encompass restoring a past copy of the database from archival backup. A more current state of the database is obtained through redoing operations of committed transactions from transaction log.

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 system restart. Recovery manager initiates recovery from the transaction logs.

In case of immediate update mode, the recovery manager takes the following actions-

- Transactions which are in active list and failed list are undone and written on the abort list.
- Transactions which are in before-commit list are redone.
- No action is taken for transactions in commit or abort lists.

In case of deferred update mode, the recovery manager takes the following actions-

- Transactions which are in the active list and failed list are written onto the abort list. No

undo operations are required since the changes have not been written to the disk yet.
- Transactions which are in before-commit list are redone.
- No action is taken for transactions in commit or abort list.

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 deferred update modes.
The recovery manager takes the following actions-
- The transactions in the commit list and before-commit list are redone and written onto the commit list in the transaction log.
- The transactions in the active list and failed list are undone and written onto the abort list in the transaction log.

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 a system crash, the recovery manager does not have to redo the transactions that have committed before checkpoint. Periodic Periodical checkpointing shortens the recovery process.

The two types of checkpointing techniques are -

- Consistent checkpointing
- Fuzzy checkpointing

Consistent Checkpointing

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

- The active transactions are suspended temporarily
- All changes in main-memory buffers are written onto the disk.
- A "checkpoint" record is written in the transaction log.
- The transaction log is written

to the disk.
- The suspended transactions are resumed.

If in step 4, the transaction log is archived as well, then this checkpointing aids in recovery from disk failures and power failures, otherwise it aids recovery from only power failures.

Fuzzy checkpointing

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

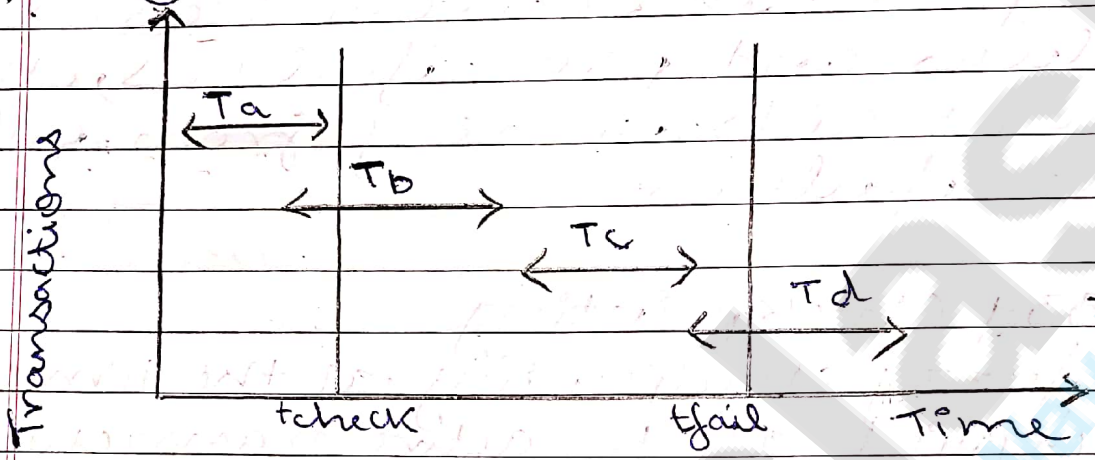
Example of checkpointing

Let us consider that in system the time of checkpointing is tcheck and the time of system crash is tfail. Let there be four transactions T_a, T_b, T_c and T_b such that -

- T_a commits before checkpoint.
- T_b starts before checkpoint and commits before system crash.
- T_c starts after checkpoint and commits before system crash.

• Td starts after checkpoint and was active at the time of system crash.

The situation is depicted in the following diagram—



The actions that are taken by the recovery manager are—

• Nothing is done with Ta.

• Transaction redo is performed for Tb and Tc.

• Transaction undo is performed for Td.

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 the transactions that have used values written

by the faulty transactions.
Transaction recovery in these cases is a two-step process -
- UNDO all faulty transactions and transactions that may be affected by the faulty transactions.
- REDO all transactions that are not faulty but have been undone due to the faulty transactions.

Steps for the UNDO operations are -
- If the faulty transaction has done INSERT, the recovery manager deletes the item(s) inserted.
- If the faulty transaction has done DELETE, the recovery manager inserts the deleted data item(s) from the log.
- If the faulty transaction has done UPDATE, the recovery manager eliminates the value by writing the before-update value from the log.

Steps for the REDO operations are -
- If the transaction has done INSERT, the recovery manager generates an insert from the log.
- If the transaction has done DELETE, the recovery manager generates a delete from the log.
- If the transaction has done UPDATE, the recovery manager generates an update from the log.