SUBJECT: ADVANCED DATABASE SYSTEM PAPER CODE: CSE-324

UNIT: 1 PARALLEL AND DISTRIBUTED DATABASES.

## Introduction:

A parallel database is one which involves multiple processor, working in parallel on the database to provide the services. A parallel database system seeks to improve performance through parallelization of various operations such as loading data, building indexes and evaluating queries. parallel systems improve processing and I/O speeds by using multiple CPUs & disks in parallel.

### How it works:

Parallel processing divides a large task into many smaller tasks and executes the smaller tasks concurrently on several CPU's and complete its more quickly.

The driving force behind parallel database system is the demand of application that have to query extremely large data bases (of the order of terabytes - that is, $10^{12}$ bytes) or that have to process an extremely large no. of transactions per second (of the order of thousands of transactions per second)

In parallel processing, many operations are performed simultaneously, as opposed to serial processing, in which the computational steps are performed sequentially.

### Performance Measure

There are two main measures of performance of a database system:

1) Throughput
2) Response time

**Throughput** The number of tasks that can be completed in a given time interval. A system that processes a large number of small transactions can improve throughput by processing many transaction in parallel.

**Response time** The amount of time it taks to complete a single task from the time it is submitted. A system that processes large transactions can improve response time as well as throughput by performing subtasks of each transaction in parallel.

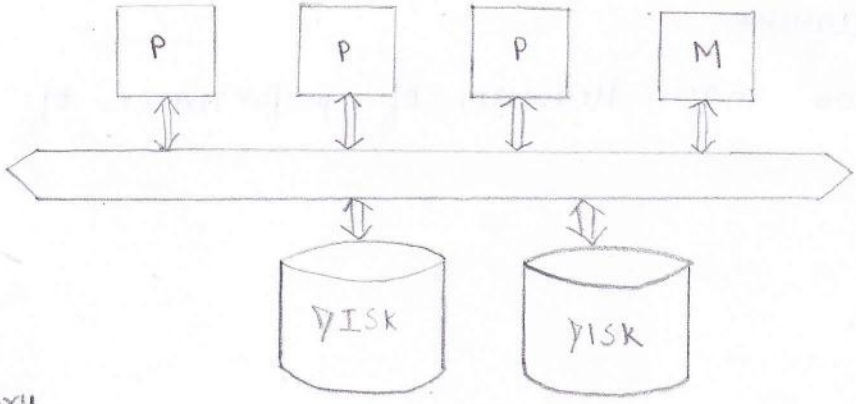
## PARALLEL DATABASE ARCHITECTURE →

The four main architecture for parallel database are as follows:

1) SHARED- MEMORY ARCHITECTURE
2) SHARED- DISK ARCHITECTURE
3) SHARED- NOTHING ARCHITECTURE
4) HIERARCHICAL ARCHITECTURE

The description of each is as follows:

1) SHARED- MEMORY ARCHITECTURE : In this, the CPUs and disks have access to a common memory through a bus or an inter-connection network. This architecture is attractive for achieving moderate parallelism since limited no of CPUs can be exploited.




P → Processor

M → Main Memory

Example: IBMS on symmetric Multiprocessor, sequent, Sun, Encore.

ADVANTAGES:

1. Efficient communication :- communication overheads are low since data in shared Memory can be accessed by any processor without being moved with software.

2 CPU-to-CPU communication is very efficient since a CPU can send data to other CPU with the speed of memory write (which takes less than a Microsecond)

3. Shared Memory Architectures usually have large Memory caches at each processor, so that referencing of the shared memory is avoided whenever possible.

4. Suitable to achieve moderate parallelism.

DISADVANTAGES

1. The architecture is not scalable beyond 32 or 64 CPUs since the processors will spend most of their time waiting for their turn on the bus to access memory.

2. Existing CPUs get slowed down, due to bus contention or network Bandwidth.

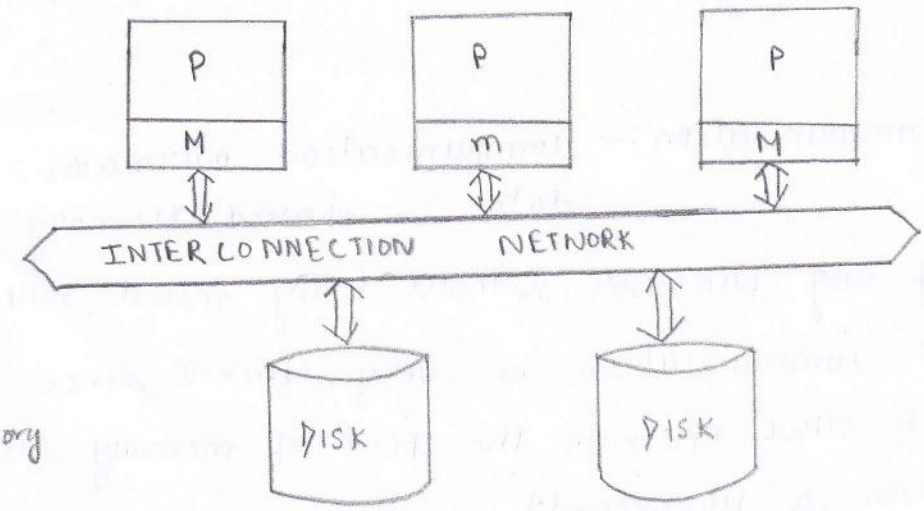
3. Maintaining cache coherency becomes an increasing overhead with increasing no. of processors

II SHARED DISK ARCHITECTURE

In this Model, all processors can access all disks directly via an interconnection network. The processors also have their private memories.

Example: DEC's VAX cluster, IMS/VS Data sharing.

The shared disk architecture is as follows:




P → Processor
M → Main Memory

ADVANTAGES

- As each processor has its own memory, thus the memory bus is not a bottleneck.
- It offers a degree of fault tolerance: If a processor (or its memory) fails, the other processors can take over its tasks, since the database is resident on disks that are accessible from all processors.
- It can scale to somewhat large number of CPUs.

DISADVANTAGE

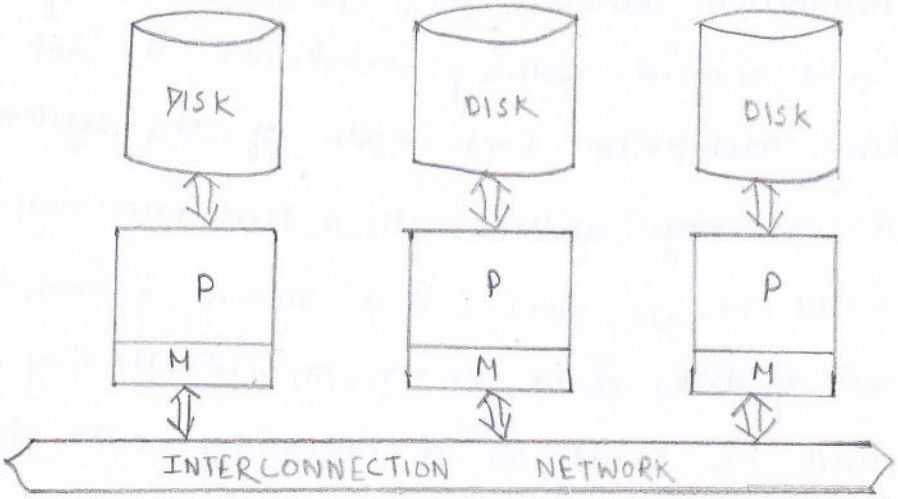
- The architecture is not scalable, since the interconnection to disk subsystem is now the bottleneck.
- CPU-to-CPU communication is slower, since it has to go through a communication network.
- The basic problem with shared disk model is interference i.e as more CPUs are added, existing CPUs are slowed down because of the increased contention for memory accesses & n/w bandwidth.

III SHARED NOTHING ARCHITECTURE

In this system, each node of the machine consists of processors, memory and one or more disks. The processor at one node may communicate with another processor at another node by a high-speed Interconnection network. The node functions as the server

for the data on the disk or disks that the node owns. as each processor has its own copy of OS, its own copy of DBMS & own data.

Example: Teradata, Gamma, Bubba




ADVANTAGE

- The Interconnection networks for shared-nothing model are usually designed to be scalable so that their transmission capacity increases as more nodes are added. Thus these architectures are more scalable and easily support a large no. of processor.
- It overcomes the disadvantage requiring all I/O to go through a single Interconnection n/w.
- It provide linear speed-up and linear scale-up i.e time taken for operations decreases in proportion to the increase in the no of CPUs & disks; scale up means the performance is sustained if the no of CPUs & disks are increased in proportion to the amount of data.

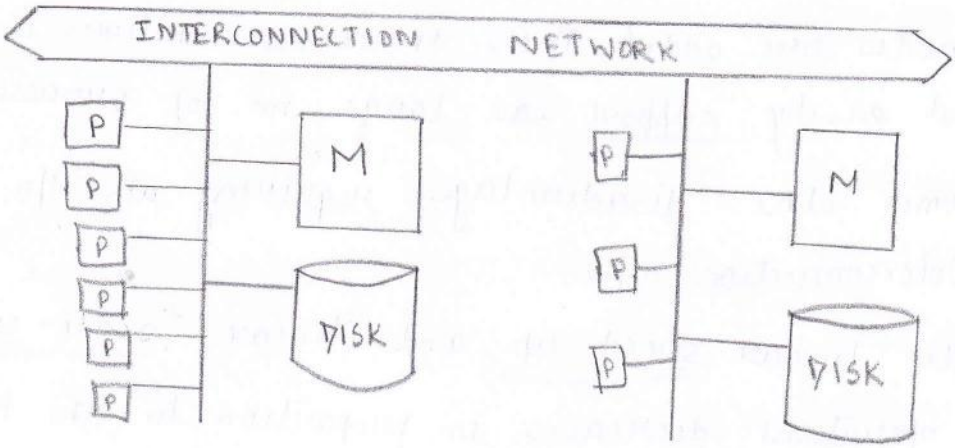
DISADVANTAGE

- CPU-to-CPU communication is very slow.

- The cost of communication and no-local disk access are higher than in shared memory or shared disk architecture since sending data involves s/w interaction at both sides.
- Shared Nothing architecture are difficult to load balance.

## IV HIERARCHICAL ARCHITECTURE

Hierarchical Architecture combines the characteristics of shared M/M, shared disk and shared Nothing architecture. The top level is Shared Nothing architecture. Each node of the system could actually be a shared memory system with a few processors. Alternatively, each node could be a shared disk system, & each of the systems sharing a set of disks could be a shared Memory system. Thus a system could be built as a hierarchy, with shared Memory architecture with a few processors at the base, and shared nothing at the top, with possibly a shared disk arch. in middle




Two Level Heirarchical Architecture

P → Processor

M → Main Memory

* This architecture is also known as <u>cluster based</u> Architecture.

Example :

ADVANTAGE

- It provides all the advantage of shared memory, shared disk and shared nothing architecture such as flexibility & better performance of shared M/M archi., higher data availability of shared disk architecture, and high extensibility of shared nothing architecture.

DISADVANTAGE

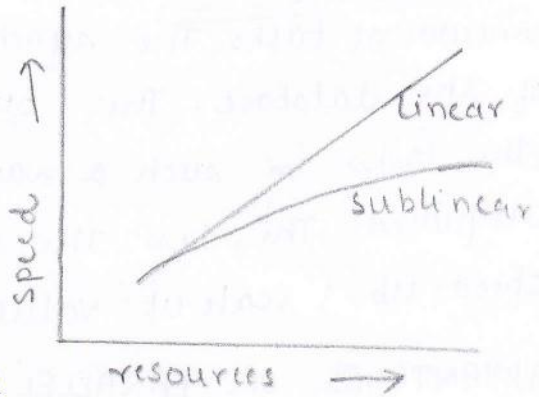
- Complexity.

KEY ELEMENTS OF PARALLEL DATABASE PROCESSING

(i) Speed-Up :→ It is the property in which the time taken for operations decrease in proportion to the increase in the number of CPUs and disks in parallel. The speed-up can be measured by using the following formula

$$\text{speed up} = \frac{\text{Actual Processing time}}{\text{Processing time in parallel}}$$

The speed-up curve shows how, for a fixed database size, more transactions can be executed per second by adding CPUs.

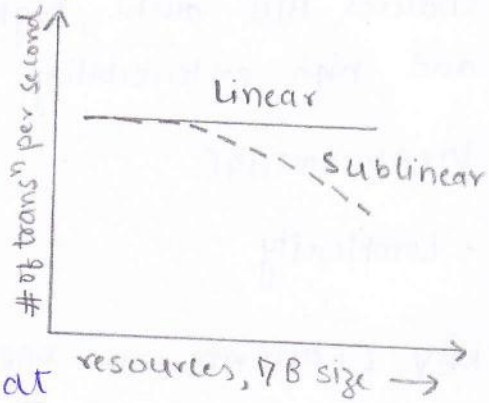



(ii) Scale-Up → It is defined as the property in which the performance of the parallel database is sustained if the no of CPUs & disks are increased in proportion to the amount of data. Scale up means the ability of the parallel database to handle larger tasks by increasing the degree of parallelism. The scale-up is measured by following formula

$$\text{scale-up} = \frac{\text{Transaction volume processed in parallel}}{\text{Transaction volume processed originally}}$$

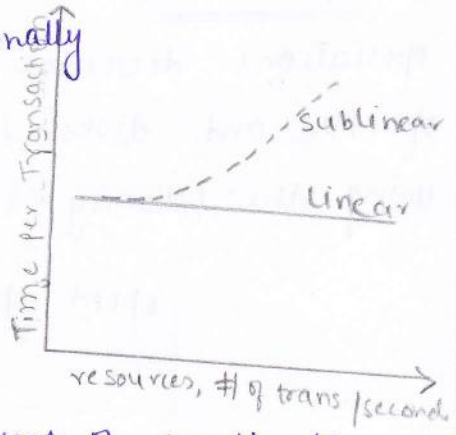
There are two kinds of scale up depending on how the size of the task is measured:

i) Batch scaleup In Batch scaleup, the size of database increases, and the tasks are large jobs whose run time depends on the size of the database




ii) Transaction Scale up In this, the rate at which transactions are submitted to the database increases and the size of the database increases proportionally to the transaction rate.

Scaleup is usually the more important metric for measuring efficiency of parallel database system.




iii) Synchronization:→ is defined as the process of coordinating the concurrent tasks. The synchronisation is required for correctness of the database. The successful parallel processing divide up the tasks in such a way that very little synchronisation is required. The Less the synchronisation required, the better the speed-up & scale up values.

ADVANTAGES OF PARALLEL DATABASE SYSTEM

(i) Increased Throughput i.e more number of tasks can be completed in a given time interval using parallel databases.

(ii) Improved Response time i.e the amount of time needed to complete a single task is reduced with the use of parallel DBses.

(iii) Using several resources (CPUs & disks) in parallel can significantly improve performance.

(iv) Increased availability: If a site containing a rel^n goes down,

the relation continues to be available if a copy is maintained at another site.

(v) Possible to serve large no. of users.

(vi) Greater flexibility.

(vii) Useful for applications to query extremely large databases and to process an extremely large number of transaction rate.

DISADVANTAGES OF PARALLEL DATABASES.

(i) Start-up cost :→ There is start-up cost associated with initiating a single process. In a parallel operation consisting of thousands of processes,

$$\text{Start-up time} \propto \frac{1}{\text{Speed-up}}$$

(ii) Interference → Existing CPUs are slowed down, as more CPUs are added because of the increased contention for memory accesses & network Bandwidth. Both speed-up & scaleup are affected by this phenomenon.

(iii) Skew → By breaking down a single task into a number of parallel steps, we reduce the size of the average step. The service time for the single slowest step will determine the service time for the task as a whole. It is often difficult to divide a task into exactly equal-sized parts, and the way that the sizes are distributed is therefore skewed.

ARCHITECTURE FOR PARALLEL QUERY EVALUATION

means how to design an architecture that allow parallel execution of multiple queries. The goal can be achieved in the shared nothing parallel database architecture.

Query Parallelism techniques are:

→ I/O parallelism

→ Interquery Parallelism

→ Intraquery Parallelism

→ Intra operation Parallelism

→ Inter operation Parallelism.

I) I/O Parallelism refers to reducing the time required to retrieve relations from disk by partitioning the relations on multiple disks. In this, the data is partitioned and stored on different disks and then each partition is processed in parallel and results are combined later on to produce the final result. The most common form of data partitioning in parallel database environment is horizontal partitioning (i.e tuples are distributed among disks).

Partitioning Techniques

a) Round-Robin partitioning.

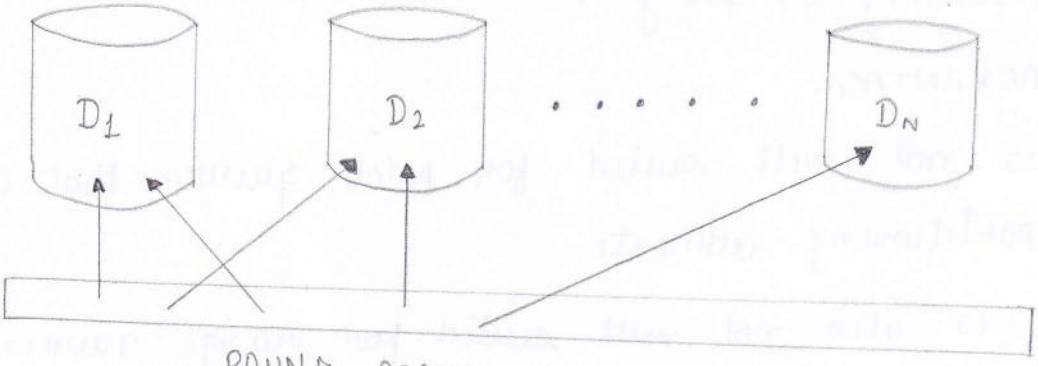
b) Hash partitioning

c) Range partitioning

Round-Robin partitioning In this, the relation is scanned in any order and the ith row of the relation is sent to the disk number i mod N.

ADVANTAGE

- This scheme ensures an even distribution of tuples across disks i.e each disk has approximately same number of tuples as others.
- It is well suited for the applications that read the entire relation sequentially for each query




ROUND- ROBIN METHOD

DISADVANTAGE

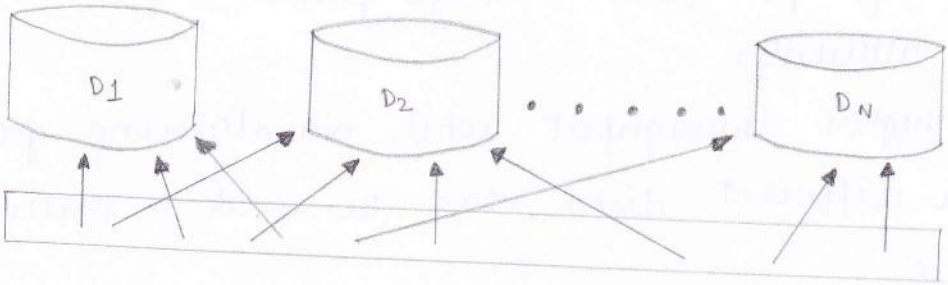
- Both point queries and range queries are very difficult to process, since all the n disks are required to process the query.

Point queries → locate tuples that have a specified value for a specific attribute (eg. emp-name = "ABC")

Range queries → locate all tuples for which the value of a given attribute lies within a specified range (eg. 10 < age < 30)

II) HASH PARTITIONING

This technique selects one or more attributes from given relation's attribute and designate them as partitioning attribute. A Hash function is chosen whose range is $\{0, 1, \ldots, n-1\}$. Each tuple of the original relation is Hashed on the partitioning attributes. If the hash function returns i, then the tuple is placed on disk $D_i$.




ADVANTAGE

- Best suited for point queries that are based on partitioning attribute.
- It is useful for sequential scans of the entire relation.
- If the hash function is a good randomizing function, then the tuples are evenly distributed among multiple disks.
- Directing a query to a single disk saves the start-up cost

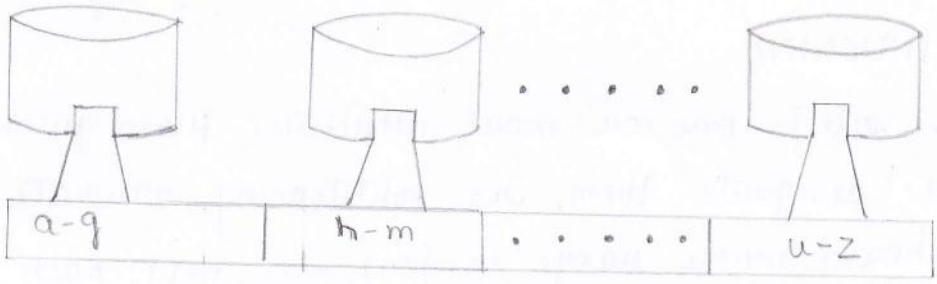
of initiating a query on multiple disks

▷ DISADVANTAGE

- It is not well suited for point queries that are based on non partitioning attribute.
- It is also not well suited for range queries, since all the disks need to be scanned to give answer.

III) RANGE PARTITIONING

This scheme distributes contiguous attribute-value ranges to each disk. In this technique, tuples are sorted and n ranges are chosen for the sort key values so that each range contains roughly the same no. of tuples; tuples in the range i, are assigned to processor i.




ADVANTAGE

- Best suited for point and range queries that are based on partitioning attributes.
- It gives higher throughput while maintaining good response time since different disks can be used to answer diff. queries

▷ DISADVANTAGE

- Skew

*PARALLEL QUERY PROCESSING TECHNIQUES →

We are having two query processing Techniques.

1) Inter query Parallelism
2) Intra query "

## II) INTERQUERY PARALLELISM

In Interquery parallelism, multiple queries or transactions execute in parallel with one another. This form of parallelism increases transaction throughput. The response time of individual transaction is almost same as if they were run in isolation.

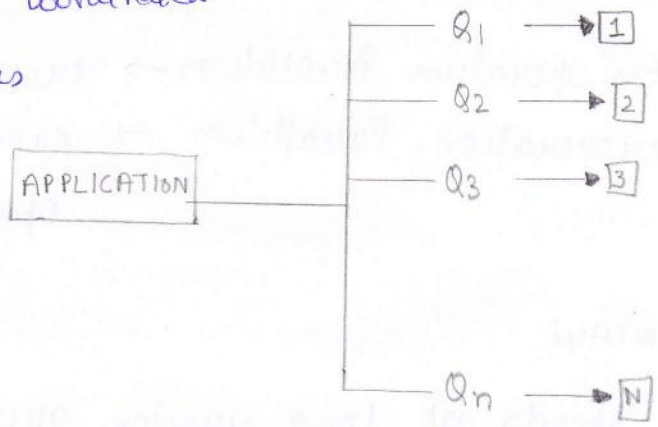
→ The primary use of this form of parallelism is to scale up a transaction-processing system to support a large number of transactions per second.

→ It is easiest form of parallelism to support in a database system- particularly in <u>shared-memory parallel system.</u>

→ Transactions that would have operated in a time-shared concurrent manner on a sequential H/C operate in parallel in the shared memory parallel architecture.

→ It is more complicated to support in a ~~time~~ shared disk or shared-nothing architecture

- Locking and logging must be coordinated by passing messages b/w processors
- Data in local buffer may have been updated at another processor.
- Cache coherency has to be maintained - reads & writes of data in buffer must find latest version of data.



Interquery parallelism

→ Example of a cache coherency protocol for shared disk systems;

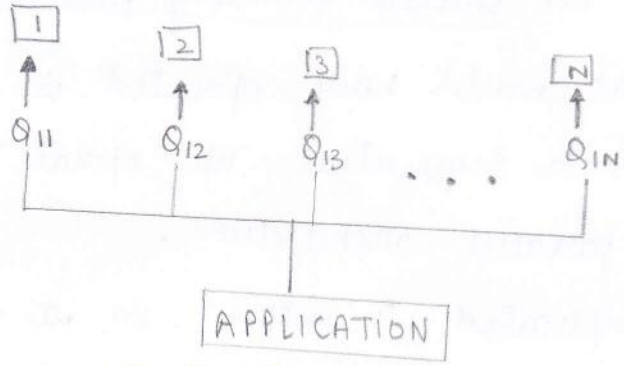
- Before reading | writing to a page, the page must be locked in shared | Exclusive mode
- On locking a page, the page must be written to disk if it was mo-

→ More complex protocols with fewer disk reads/writes exist

→ Cache coherency protocols for shared-nothing systems are similar. Each database page is assigned a home processor. Requests to fetch page or write it to disk are sent to the home processor.

## III INTRAQUERY PARALLELISM →

refers to the execution of a single query in parallel on multiple processors and disks. Intraquery parallelism helps in speeding-up long running queries.




The execution of a single query can be parallelized in two different ways:

→ Intra operation Parallelism → execution of each individual operator

→ Interoperation Parallelism → execution in parallel different operations in query expression.

ADVANTAGE

- It speeds up long running queries
- It is useful in decision support application

## PARALLELIZING INDIVIDUAL OPERATIONS →

means how various operations can be implemented in parallel in a Shared-Nothing Architecture. We assume that each relation is Partitioned horizontally across several disks.

## Interoperation parallelism →

The Interoperation parallelism is achieved by executing in ↑↑ the different operations in a query expression. There are two forms of Interoperation parallelism:

→ Pipelined parallelism

→ Independent parallism.

**Pipelined parallelism** → In pipelined parallelism, the output tuples of one operation, A, act as input to second operation, B, even before the first operation has produced the entire set of tuples in its output. Thus, it is possible to run operations A & B simultaneously on different CPUs, so that tuples produced by A is taken by B as input in parallel.

example consider a join of four relations

$$r_1 \bowtie r_2 \bowtie r_3 \bowtie r_4$$

Suppose processor $P_1$ is assigned the computation of $\boxed{temp_1 \leftarrow r_1 \bowtie r_2}$

Ally Processor $P_2$ is assigned $\boxed{temp_2 \leftarrow r_3 \bowtie temp_1}$

As $P_1$ computes tuples in $r_1 \bowtie r_2$, it makes these tuples available to processor $P_2$. $P_2$ can use these tuples to begin computation of $temp_1 \bowtie r_3$, even before $r_1 \bowtie r_2$ is fully computed by $P_1$. Likewise, as $P_2$ computes tuples in $(r_1 \bowtie r_2) \bowtie r_3$, it makes these tuples available to $P_3$, which computes the join of these tuples with $r_4$.

ADVANTAGE

- It is useful with small number of CPUs.
- Writing intermediate results to the disk can be avoided.

DISADVANTAGE

- It doesn't scale-up well
- When the degree of parallelism is high, it is less important than partitioning.
- Only marginal speed-up is obtained, when one operator's cost is much higher than others.
- It is not possible to pipeline relational operators that do not produce output until all i/p have been accessed eg. set difference operation.

Independent parallelism

In Independent parallelism, the operations in a query expression that do not depend on one another can be executed in parallel.

example consider the join of four relations.

$$r_1 \bowtie r_2 \bowtie r_3 \bowtie r_4$$

We can compute $temp_1 \leftarrow r_1 \bowtie r_2$

$temp_2 \leftarrow r_3 \bowtie r_4$

Then: $temp_1 \bowtie temp_2$.

ADVANTAGE

- It is useful with a lower degree of parallelism

DISADVANTAGE

- It doesn't provide a high degree of parallelism
- The operations in a query expression can be parallelized only if they are independent.

## INTRA OPERATION PARALLELISM →

means parallelizing the execution of each individual operation of a query such as sort, select, project & join.

1. Sorting

suppose that we wish to sort a relation that resides on n disks $D_0, D_1, D_2 \cdots D_{n-1}$. If the relation has been range-partitioned on the attributes on which it is to be sorted, we can sort each partition separately and can concatenate the results to get the full sorted relation. The degree of parallelism is likely to be limited by the Merging phase.

Example if we want to sort a collection of employees tuples by salary salary value ranges from 10 to 210, & we have 20 processors, we could send tuples with salary range 10 to 20 to first processor, 21 to 30 to 2nd processor & so on. Each processor then sorts the tuple assigned to it, using some sequential sorting algo. & writes out a run on local disk until all sorted tuples have been written to local disk.

→ The basic challenge in ↑↑ sorting is to do the range partitioning so that each processor recieves roughly the same number of tuples; otherwise, a processor that recieves a disproportionately large no. of tuples to sort becomes a bottleneck & limits the scalability of the parallel sort.

Solution

Application: of parallel sorting is sorting the data entries in tree structured indexes. Sorting entries can speed up the process of bulk-

## Parallel Join

- Partitioned Join
- Fragment-and-Replicate Join
- Partitioned parallel Hash Join

**Partitioned Join** → technique works correctly only if the Join

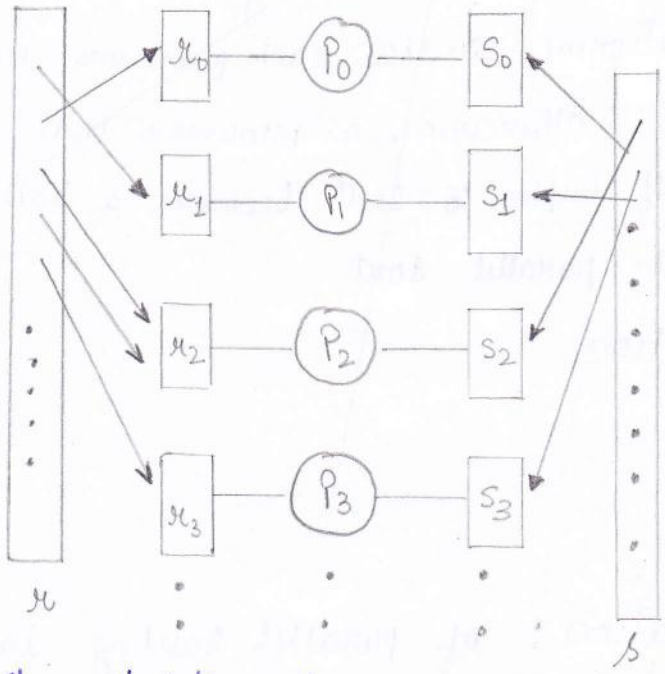
- is an equi-Join (eg. $r \bowtie_{r.A=s.B} s$)
- if we partition $r$ and $s$ by the same partitioning function on their Join attribute.

For certain kinds of joins, such as equi-joins and natural joins, it is possible to partition the two input relations across the processors and to compute the Join locally at each processor.

Suppose we are using $n$ processors and $r$ & $s$ are relations to be joined. Partitioned Join works as follows:

The system partitions the relations $r$ and $s$ each into $n$ partitions denoted $r_0, r_1, \ldots r_{n-1}$ and $s_0, s_1, \ldots s_{n-1}$. The system sends partition $r_i$ and $s_i$ to processor $P_i$, where their join is computed locally.

Any join technique can be used locally at each processor $P_i$ to compute join of $r_i$ & $s_i$ eg. hash join, nested loop join




ADVANTAGE

- Simplicity

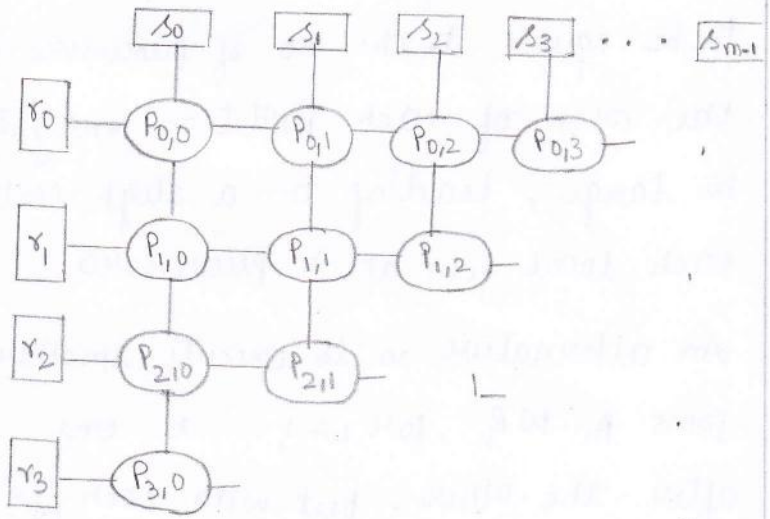
Disadvantage

- Skew is a problem if range Partitioning Method is used.
- Same Partitioning Method is used

3) Not used for inequality Joins

## Fragment-and-Replicate Join

- is used for inequality Join condition, such as $r \bowtie_{r.a < s.b}$, it may be possible that all Tuples in $r$ join with some tuple in $s$. (and vice versa). Thus, there is no easy way of partitioning $r$ and $s$ so that tuples in $r_i$ joins with only tuples in $s_i$.
- Such joins can be parallelize by using Fragment-and-Replicate Join Technique
- The system partition relation $r$ into $n$ partitions $(r_0, r_1 \ldots r_{n-1})$ and partitions $s$ into $m$ partitions $(s_0, s_1 \ldots s_{m-1})$ The values of $m$; $n$ do not need to be equal.; No. of processor $= m * n$. Let the processors be $P_{0,0}, P_{0,1}, \ldots, P_{0,m-1}, P_{1,0} \ldots P_{n-1,m-1}$. Processor $P_{i,j}$ computes the join of $r_i$ with $s_j$. Each processor must get those tuples in the partitions on which it works. To accomplish this, the system replicates $r_i$ to processors $P_{i,0}, P_{i,1} \ldots P_{i,m-1}$ and replicates $s_i$ to processors $P_{0,i}, P_{1,i} \ldots P_{n-1,i}$. Any join technique can be used locally at each processor $P_{i,j}$.



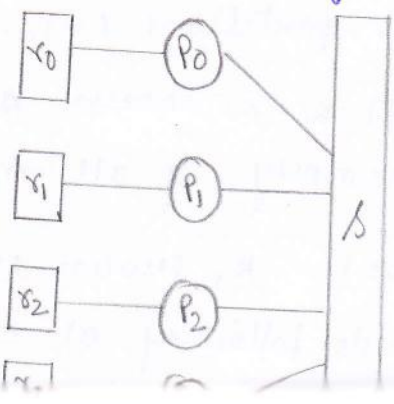

→ Asymmetric Fragment-&-Replicate Join

→ is a special case of fragment-&-Replicate Join when $m = 1$.

→ In this technique → a) The system partitions one of relations (say $r$). Any partitioning technique can be used on $r$, including Round-Robin method

b) The system replicates other relation (say $s$) across all processors

c) processor $P_i$ then locally computes join



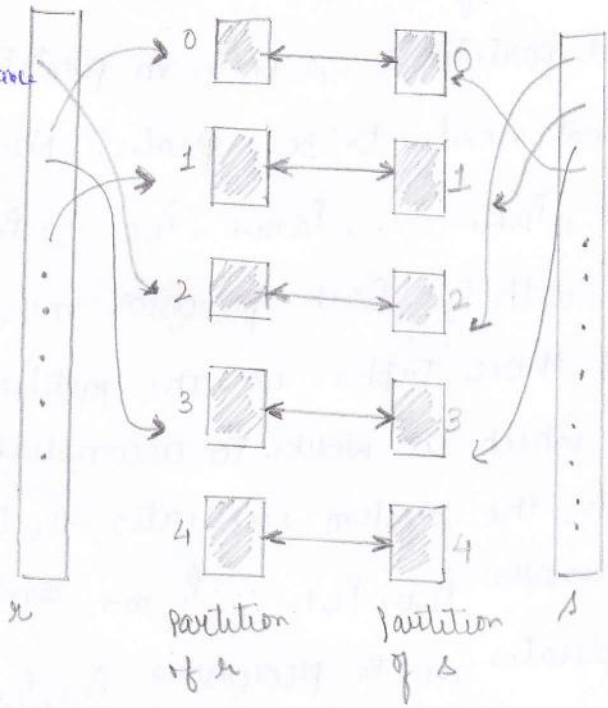
Disadvantage

when r & s are of roughly the same size, then fragment & Replica Method have higher cost, as at least one of the relation has to be replicated.

→ Asymmetric Fragment-&-Replicate Method is used when one of the relation is small since it may be cheaper to replicate that relation across all processors.

III) Partitioned parallel Hash-Join

- Hash based approach improves performance
- If A & B are very large relations, and the no of partitions k is chosen to be equal to the no of processors n, the size of each partition may still be large, leading to a high cost for each local join at n processors.




Hash partitioning of relations

An alternative is to execute smaller joins $A_i \bowtie B_i$, for $i = 1, \ldots k$ one after the other, but with each join executed in ↑↑ using all the processors. This approach allows us to utilize the total available main memory at all n processors in each join $A_i \bowtie B_i$. The description is as follows:

1. At each site, apply a hash fun $h_1$ to partition A & B tuples at this site into partitions $i = 1, \ldots k$. Let A be the smaller relation. The no of partition k is chosen such that each partition of A fits into combined memory of all n processors.

2. For $i = 1, \ldots k$, process the join of $i^{th}$ partitions of A & B. To compute $A_i \bowtie B_i$, do following at each site:

a) Apply a second Hash fun $h_2$ to all $A_i$ tuples to determine where

they should be joined & send to Tuple t to site h2(t)

b) As Ai tuples arrive to be joined, add them to an in-memory hash table

c) After all Ai tuples have been distributed apply h2 to Bi tuples to determine where they should be joined & send tuple t to site h2(t)

d) As Bi tuples arrive to be joined, probe in-memory table of Ai tuples and o/p result tuples. The use of 2nd Hash funⁿ h2 ensures that tuples are uniformly distributed across all n processors.

## * PARALLEL QUERY OPTIMIZATION →

→ Query Optimizer is a measure for success of relational Technology

→ Query optimizer takes a query & finds the cheapest execution plan among the many possible execution plans that give same Ans. By Execution plan, we mean a data flow network of relational, merge, join operators.

→ Optimizing a single query for parallel execution has recieved more attention:

→ ∥ query optimizer are more complicated than query optimizer for sequential query evaluation. The reasons are:

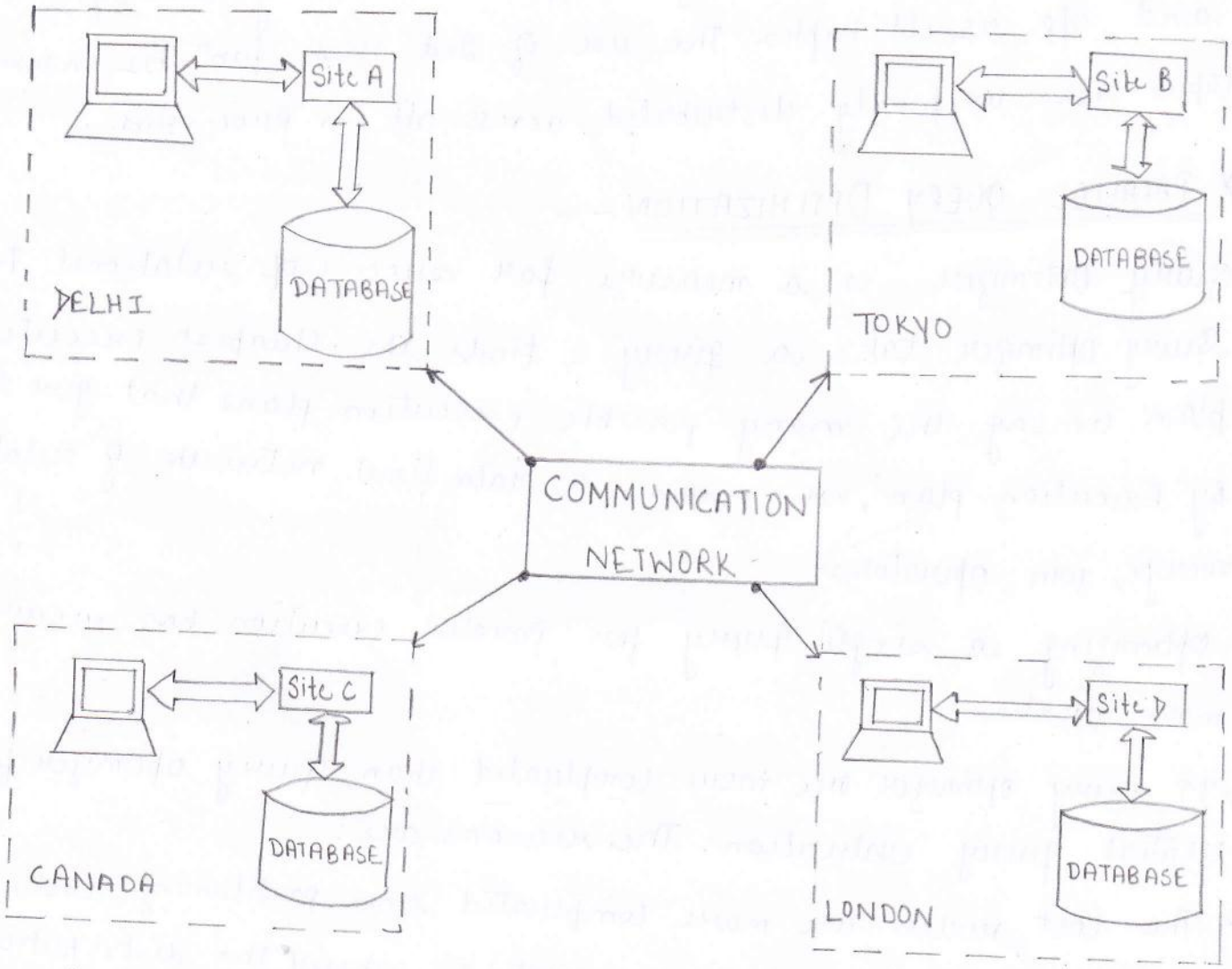
⇒ The cost models are more complicated, since partitioning costs have to be accounted for and issues such as skew (the distribution of work among the processors, with some processor getting a large no of tuples than others) and resource contention (resulting in delays) must be taken into account.

⇒ The cost of executing operations in ∥ differs from executing sequentially

⇒ The No of resources - such as processor, disk, buffer space -, that should be allocated to each ~~processor~~ operation are known only at run-time.

# DISTRIBUTED DATABASE

A Distributed database system is a database physically stored on several computer systems across several sites connected together via communication Network. Each site is typically managed by a DBMS that is capable of running independently of other sites.




- Each site is a database system in its own right; has its own local users; its own local DBMS; its own local data comm$^n$ manager.
- In a distributed database system, both data & transaction processing are divided between one or more computers (CPUs) connected by a n/w, each computer playing a special role in the system.
- A distributed database system allows applications to access data from local & remote database.
- Distributed database systems use client / server architecture to process information requests. The computers in distributed database system are

→ The classical view of distributed database system is that the system should make the impact of data distribution transparent.

→ Distributed database system is based on the principle of Local Autonomy: This is the capability to administer a local database and to operate independently when connection to other nodes have failed.

Difference :→

| Parallel DB | Distributed DB |
|---|---|
| → Multiple CPUs are tightly coupled. | → Multiple CPUs are loosely coupled. |
| → Single database is stored on Multiple processor | → Each ; every site has their own database |
| → Data is stored at single site | → data is geographically distributed. |
| → " | → Diff sites communicate through telephone lines, optical fiber etc. |
| → In TTDB, local ; global transactions are differentiated. | → In distributed DBs, local ; global transactions are differentiated. |

Properties of Distributed Data Bases

a) Distributed Data Independence → property enables users to ask queries without specifying where the reference relations or copies or fragments of the relation, are located. This principle is natural extension of physical ; logical Data Independence.

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

Types: →

1) Homogeneous DDBMS → where there are several sites, each running their own application on same DBMS software. All sites have identical DBMS software, all users use identical software, are aware of one another &

2) Heterogeneous DDMS → where different sites run under the control of different DBMSs, essentially autonomously and are connected to enable access to data from multiple sites.

Functions: →

→ Provide local Autonomy
→ Should be location Independent
→ No reliance on a central site
→ Ability of replicated data Mgmt to access & maintain the consistency of a replicated data item.
→ Ability of distributed transaction Mgmt by devising exec^n strategies for queries & transactions that access data from several sites
→ Should be Hardware independent
→ Should be operating System Independent
→ Should be network independent
→ should be DBMS independent
→ Efficient distributed database recovery Mgmt in case of site crashes & comm^n failures.

Advantages →

→ Increased Efficiency of processing by keeping the data close to the poi where it is most frequently used.
→ Efficient Mgmt of distributed data with diff levels of transparency.
→ Improved reliability due to greater accessibility
Improved performance
→ Parallel Evaluation by subdividing a query into sub-queries involving data from several sites.
→ Increased local Autonomy where each site is able to retain degree o control over data that are stored locally.

→ Increased Availability in which if one site fails, the remaining site may be able to continue operating.

→ Increased Accessibility by allowing to access data between several sites via commⁿ Network.

→ Easier Expansion with the growth of organization in terms of adding more data increasing database sits or adding more CPUs.

Disadvantage →

→ Recovery of failure is more complex

→ Increased S/W development cost.

→ Increased processing overhead.

→ Difficulty in database Integrity control

→ Security concern of replicated data in multiple location & the n/w

→ Increased Complexity in the system design & implementation

* ARCHITECTURE OF DDBMS → (Ref. Database System by S.K. Singh)

→ Client - Server Architecture

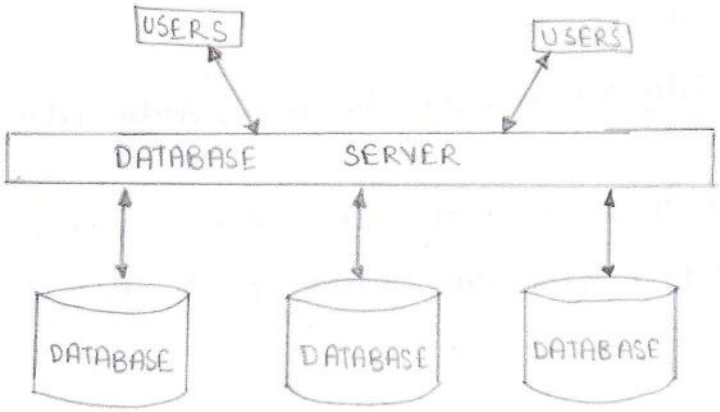
are those in which a DBMS - related workload is split into two logical components namely client & server, each of which typically executes on different systems. Client / server architecture has one or more client processes and one or more server processes. The applications and tools are put on one or more client platforms and are connected to DBMS that resides on the server. The applications and tools act as 'client' of the DBMS, making requests for its services. The DBMS, in turn, services these requests & returns the results to the client(s). Clients are responsible for user-interface issues and server manage data & execute transactions. A client process could run on a personal computer and send queries to a server running on a mainframe computer.

Main Components:

→ clients in form of intelligent workstation as the user's contact point

→ DBMS server as common resources performing specialised tasks for devices, requesting their services

→ commⁿ N/W connecting client & server




ADVANTAGE

→ This architecture is relatively simple to implement, due to its clean separation of functionality and bcoz the server is centralized.

→ Use of GUI on microcomputers by the user at client, improves functionality & simplicity.

→ Less Expensive

→ Computing platform Independence

→ Improved performance

DISADVANTAGE

→ An Increase in the number of users and processing sites often create security problems.

→ The client process is quite complex and its capabilities begin to overlap with the Server. This results in difficulty in distinguishing b/w clients & Servers.

→ The client/server architecture doesn't allow a single query to span multiple servers bcoz the client process would have to be capable of breaking such a query into appropriate sub-queries to be executed at different sites & then putting together the answer to sub-queries.

II) COLLABORATING SERVER SYSTEM

In this architecture, there are several database servers, each capable of running transaction against local data, which cooperatively execute transactions spanning multiple servers. When a server receives a query that requires access to data at other servers, it generates appropriate sub-queries to be executed by other servers

and puts the results together to compute Answers to the original query.

III) MIDDLEWARE SYSTEMS

is also called Data Access Middleware, is designed to allow a single query to span multiple servers, w/o requiring all database servers to be capable of managing such multisite execution strategies. It is especially attractive when trying to integrate several Legacy systems, whose basic capabilities can't be extended.

The Idea is that we need just one database server capable of managing queries & transactions spanning multiple servers; the remaining servers need to handle only local queries & transactions. We can think of this special server as a layer of s/w that coordinates the execution of queries & transaction across one or more indep. database server; such s/w is often called Middleware. The Middleware layer is capable of executing joins & other relational operations on data obtained from other servers but, doesn't itself maintain any data.

(Ref. Database System by S.K. Singh)

* STORING DATA IN A DDBMS →

In a DDBMS, relations are stored across several sites. Accessing a relation stored at a remote site incurs message-passing costs and, to reduce this overhead, a single relation may be partitioned or fragmented across several sites, with fragments stored at the sites where they are most often accessed or replicated at each site where the Rel$^n$ is in High demand.

Data fragmentation

Fragmentation consists of breaking a relation into smaller relations or fragments and storing the fragments at different sites. In data fragmentation, a rel$^n$ can be partitioned into several fragments & there may be several replicas of each fragment. These fragments contain sufficient information to allow reconstruction of the original rel$^n$. All fragments of a given rel$^n$ are independent.

Example →

| EMP-ID | EMP-NAME | DEPT-ID | EMP-SALARY |
|---|---|---|---|
| E-106519 | Kumar Abhishek | 4 | 55,000 |
| E-112233 | Thomas Mathew | 5 | 45,000 |
| E-123456 | Alka Parasar | 2 | 60,000 |
| E-100600 | Jose Martin | 5 | 30,000 |
| E-198900 | Meena Singh | 4 | 80,000 |

VERTICAL FRAGMENTATION

HORIZONTAL FRAGMENTATION

Now this relⁿ can be fragmented into three fragments as follows:

| FRAGMENT | EMPLOYEE AS | | |
|---|---|---|---|
| MUMBAI-EMP | AT SITE | "Mumbai" | where DEPT-ID=2 |
| JAMSHEDPUR-EMP | AT SITE | 'Jamshedpur' | where DEPT-ID=4 |
| LONDON-EMP | AT SITE | 'LONDON' | where DEPT-ID=5; |

Now we get 3 fragments of above relⁿ. and fragments are stored at their respective site. The DDBMS's internal fragment names are MUMBAI-EMP. etc. Reconstruction of the original relⁿ is done via suitable JOIN & UNION operⁿ.

→ A system that supports data fragmentation should also support fragmentation Independence | Transparency i.e users should not be logically concerned about fragmentation.

Types of fragmentation

→ Horizontal fragmentation

→ Vertical fragmentation

→ Mixed fragmentation

Horizontal fragmentation → of a relⁿ is a subset of rows with all attributes in that relⁿ.

- It divides the relⁿ "Horizontally" by assigning each tuple or group of tuples of a relⁿ to one or more fragments, where each tuple or a subset has certain logical meaning. These fragments can then be assigned

to different sites in the distributed System.

- Horizontal fragmentation may be defined as $\boxed{\sigma_p(R)}$

where $\sigma$ = relational algebra operators for selection

$p$ = predicate based on one or more attributes of relⁿ

$R$ = a relation (Table)

| Relation : LONDON_EMP | | | |
|---|---|---|---|
| EMP-ID | EMP_NAME | DEPT-ID | EMP_SALARY |
| E-112233 | Thomas Mathew | 5 | 45000 |
| E-100600 | Jose Martin | 5 | 30000 |

Therefore MUMBAI_EMP : $\sigma_{DEPT_ID=2}$ (EMPLOYEE)

LONDON_EMP : $\sigma_{DEPT_ID=5}$ (EMPLOYEE)

In Horizontal fragmentation, UNION operation is done to reconstruct the original relation.

II) <u>VERTICAL FRAGMENTATION</u> → splits the relation by decomposing 'vertically' by columns (attributes). A vertical fragment of a relⁿ keeps only certain attributes of the relⁿ at a particular site, bcoz each site may not need all the attributes of a relⁿ. Thus, vertical fragmentation groups together the attributes in a relⁿ that are used jointly by the imp transaction. Since there is no common attribute between the two fragments, we can't put the original employee tuples back together. Therefore, it is necessary to include the primary key (or candidate key) attributes in every vertical fragment so that the full relⁿ can be reconstructed from fragments. In vertical fragmentation, system provided 'tuple-ID' or (TID) is used as primary key (or candidate key) attribute along with the stored relation as address for linking tuples.

Vertical fragmentation corresponds to the relational operations of projection and is defined as $\Pi_{a_1, a_2 \ldots a_n}(R)$

where $\Pi$ = Relation Algebra operator for projection

$a_1 \ldots a_n$ = attributes of relⁿ

R = a relⁿ (Table)

Example:

FRAGMENT EMPLOYEE AS

MUMBAI_EMP (TID, EMP-ID, EMP-NAME) AT SITE 'MUMBAI'

JAMSHEDPUR_EMP (TID, DEPT_ID) AT SITE 'JAMSHEDPUR'

LONDON_EMP (TID, EMP-SALARY) AT SITE 'LONDON';

Therefore, vertical fragmentation

MUMBAI_EMP : $\Pi_{TID, EMP-ID, EMP_NAME}$ (EMPLOYEE)

LONDON_EMP : $\Pi_{TID, EMP-SALARY}$ (EMPLOYEE)

LONDON_EMP

| TID | EMP_SALARY |
|---|---|
| T1 | 55000 |
| T2 | 45000 |
| T3 | 60000 |
| T4 | 30000 |
| T5 | 80000 |

fig(1)

MUMBAI-EMP

| TID | EMP-ID | EMP-NAME |
|---|---|---|
| T1 | E-106519 | Kumar Abhishek |
| T2 | E-112233 | Thomas Mathew |
| T3 | E-123456 | Alka Parasar |
| T4 | E-100600 | Jose Martin |
| T5 | E-198900 | Meena Singh |

## III) Mixed fragmentation →

→ Sometimes, Horizontal or vertical fragmentation of database scheme by itself is insufficient to adequately distribute the data for some Applications. Instead, Mixed or hybrid fragmentation is required.

→ Thus, Horizontal (or vertical) fragmentation followed by further vertical (or Horizontal) fragmentation of some of fragments, is called Mixed fragmentation.

→ Mixed fragmentation is defined using the selection (SELECT) and projection (PROJECT) operations of relational algebra. The original relation is obtained by a combination of JOIN and UNION operⁿ.

→ A mixed fragmentation is given as:

$$\sigma_p (\Pi_{a_1, a_2 \ldots a_n} (R))$$

OR

$$\Pi_{a_1, a_2 \ldots a_n} (\sigma_p (R))$$

Example:→ Vertical fragmentation of EMPLOYEE relⁿ followed by Horizontal fragmentation of "~~JAMSHEDPUR-EMP~~", "LONDON-EMP".

$S_1$ = MUMBAI_EMP : $\pi_{TID, EMP_ID, EMP_NAME}$ (EMPLOYEE)

$S_2$ = LONDON_EMP : $\pi_{TID, EMP_SALARY}$ (EMPLOYEE) [Tables shown in fig (i)]

Horizontal fragmentation on $S_2$.

$S_{21} = \sigma_{EMP_SALARY < 80000}(S_2)$ : $\sigma_{EMP_SALARY < 80000}$ (LONDON-EMP)

Mixed fragmentation ⟹

| LONDON_EMP | |
|---|---|
| TID | EMP_SALARY |
| T1 | 55000 |
| T2 | 45000 |
| T3 | 60000 |
| T4 | 30000 |

## 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 stores each replica at a different site.

→ Typically data replication is introduced to increase availability of system.

→ When a copy is not available due to site failure, it should be possible to access another copy.

→ Like fragmentation, data replication should also support Replication Independence that means, users should be able to behave as if the data were in fact not replicated at all. **

ADVANTAGE

- It enhances the performance of read operⁿ by increasing the processing speed at site.
- It increases the availability of data to read-only Tns. (transcⁿ)

DISADVANTAGE

- Increased overhead for update Transcation.
- More complexity in controlling concurrent updates by several transaction to replicated data.

** A Database is said to be:

Strictly partitioned - when no replicates of the fragments are allowed.

# * DISTRIBUTED CATALOG MANAGEMENT →

(some part of Replica)

2. fully Redundant → when complete database copies are distributed at a

3. partially Redundant → when only certain fragments are replicated

Choice 1 for data replication would lead to relatively expensive qu evaluation due to unavailability of the data locally or at some nea site. choice 2 is very expensive in terms of storage space and th overhead to maintain consistency. It is meaningless to replicate da at nodes where it is unlikely to be accessed. choice 3 is reasonabl allowing for reduced access time for frequently read local data or that from a nearby site. The choice allows for higher reliability and availability during site crashes. But, Bcoz of replication, upd ates are expensive. ** (end of Replica)

## Naming object

If a relation is fragmented and replicated, we must be able to uniquely identify each replica of each fragment. Generating Suc unique names require some care. If we use a global name-serv to assign globally unique-names, local autonomy is compromised; w want (users at) each site to be able to assign names to local objec w/o reference to names systemwide. The usual solⁿ to naming probl is to use names consisting of several fields. For eg.

Local name field, which is name assigned locally at the site where the relation is created.

Birth site field, which identifies the site where the relation was cr

## Catalog structure

System catalogs, or dictionaries, maintain metadata on database Relⁿ. In a distributed database system, info on locations, fragmentation's Replication is also added to catalogs. The catalogs themselves, just like the database, may be distributed in any of the above 3 ways. That is, catalog would be 1. Strictly Partitioned

2. fully Redundant.

3. Partially

Distributed Data Independence
means that users should be able to write queries w/o regard to how a relⁿ is fragmented or replicated; it is the responsibility of DBMS to compute the relation as needed.

## * DISTRIBUTED QUERY PROCESSING →

(Ref. Database System by S.K.Singh)

In DDBMS, a query may require data from the databases distributed in more than one site. Different relⁿ might reside at diff sites, multiple copies of a single relⁿ can be distributed among several sites or one relⁿ might be partitioned into sub-relⁿ and these sub-relⁿ distributed to multiple sites. In order to evaluate a query issued at a given site, it may be necessary to transfer data b/w various sites Therefore, it is important to optimise on the time required to access such a query, which will be largely comprised of time spent in transmitting data b/w sites and not the time spent on retrieval from the disk storage.

a) Costs (Transfer of data) of distributed query processing

The data transfer costs of distributed query processing involves cost of transferring intermediate files to other sites for processing and the cost of transferring the final results to the site where the result is required.

Data Transfer Cost = C * size

where C = cost per byte of transferring data

size = No. of bytes transmitted.

b) SEMIJOIN

The semijoin operation is used in distributed query processing to reduce the number of tuples in a relⁿ before transmitting it to another site. This reduction in the number of tuples, reduces the number and total size of the transmission that ultimately reduces the total cost of data transfer.

Let us assume, that we have two relations $R_1$ & $R_2$ on site $S_1$ & $S_2$. We will send the joining column of one relation ($R_1$) to the site where the other ($R_2$) is located. This column is joined with $R_2$ at that site. The decision as to whether to reduce $R_1$ or $R_2$ can only be made after comparing the advantages of reducing $R_1$ with that of reducing $R_2$. Thus semijoin is an efficient solution to minimize the data transfer in distributed query processing.

## * UPDATING DISTRIBUTED DATA →

The classical view of a distributed DBMS is that it should behave just like a centralized DBMS from the point of view of user; issues arising from distribution of data should be transparent to user. With respect to updates, this view means that transactions should continue to be atomic actions, regardless of data fragmentation & Replication. We are having two approaches.

1) Synchronous Replication

All copies of a Modified relⁿ must be updated before the Modifying transaction commits. i.e Before an update transcⁿ commits, it synchronizes all copies of Modified data.

There are two basic technique for ensuring that transcⁿ see the same value regardless of which copy of an object they access.

a) Voting In this, a transcⁿ must write a majority of copies to modify an object and read at least enough copies to make sure that one of the copies is current. Each copy has a version Number and the copy with the highest version no. is current.

b) Read-any write-all In this, to read an object, a transaction can read any one copy, but to write an object, it must write all copies. Reads are fast, especially if we have a local copy, but writes are slower relative to first technique.

2) Asynchronous Replication

Copies of a Modified Relation are updated only periodically and a transcⁿ that reads different copies of the same relⁿ may see diff values. Thus, asynchronous Replication compromises distributed data Independence, but it can be implemented more efficiently than synchronous Replication. Asynchronous Replication comes in two flavors.

a) primary site In this, one copy of a relⁿ is designated the 1° or master copy. Replicas of the entire relation or fragments of the relⁿ can be created at other sites; these are 2° copies and Unlike the 1° copy, they can't be updated.

b) Peer-to-peer In this, more than one copy can be designated as updatable, that is, master copy. In addition to propagating changes

a conflict Resolution strategy must be used to deal with conflicting changes made at different sites.

## * DISTRIBUTED TRANSACTION

In distributed DBMS, a given transaction is submitted at some one site, but it can access data at other sites as well. when a transc$^n$ is submitted at some site, the transaction Manager at that site breaks it up into a collection of one or more subtransaction that execute at different sites, submits them to transc$^n$ managers at other sites & coordinates their activity.

## * DISTRIBUTED CONCURRENCY CONTROL →

Since data in DDBMS, is distributed, the transc$^n$ activities may take place at a no of sites and it can be difficult to maintain a time ordering among actions. The concurrency control becomes difficult when two or more transc$^n$ are executing concurrently and both require access to same data record in order to complete the processing. All concurrency control mechanisms must ensure that the consistency of data items is perserved, and that each atomic action is completed in a finite time. A good concurrency control mechanism for distributed DBMS should have foll. characteristics:

- Resilent to site & comm$^n$ failure.
- Permit TMsm to satisfy performance requirements.
- Satisfactory performance in a N/W environment having comm$^n$ delays.

a) Distributed locking.

- Simplest Method of concurrency control.
- In DDBMS, the lock manager function is distributed over several sites.
- The DDBMS maintains a lock manager at each site whose fun$^n$ is to administer the lock & unlock requests for those data items that are stored at that site.
- Lock can be applied in two modes → shared mode, Exclusive mode.
- If a transc$^n$ locks a record in shared mode then all trasc$^n$ can

If a transⁿ locks a record exclusive mode, all copies of the data items on record have to be modified & locked.

ADVANTAGE

- Simple Implementation
- Reduces the degree of bottleneck
- Reasonably low overhead

DISADVANTAGE

- Complex deadlock Handling bcoz lock & unlock requests are not made at single site.
- Possibility of inter-site deadlocks even when there is no deadlock within a single site.

b) Distributed Deadlock →

Concurrency Control with a locking based Algorithm may result in deadlock. As in the centralized DBMS, deadlock must be detected & resolved in DDBS by aborting some deadlock transcⁿ. In a DDBS, each site maintains a local-wait-for-graph (LWFG) & a cycle in local graph indicates a deadlock. In a distributed system, the detection of a deadlock requires the generation of not only local WFG for each site, but also Global WFG for the entire system. However GWFG has a disad. of the overhead required in generating such graphs.

In a distributed DS, the deadlock prevention Method by aborting the transcⁿ can be used such as time-stamping, wait-die method. The aborted transcⁿ are reinitiated with the original timestamp to allow them to eventually run to completion.

c) Timestamping →

is a method of identifying messages with their time of transⁿ. In DDBS, each copy of the data item contains two timestamp values, namely read timestamp & write timestamp.

## * RECOVERY CONTROL IN DDBs →

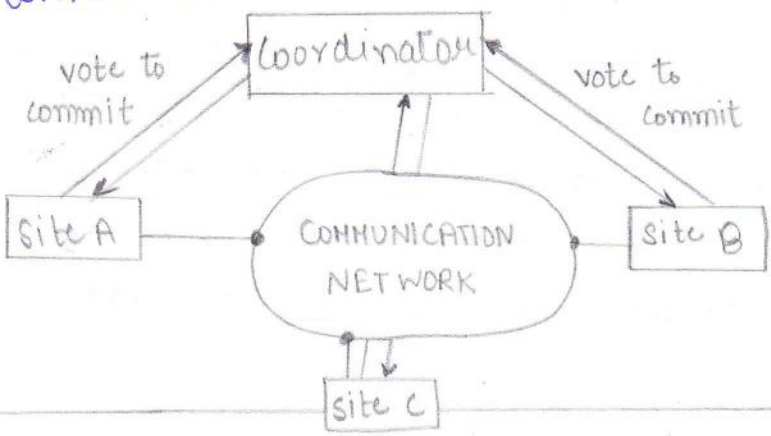
As with local recovery, distributed database recovery aims to maintain the atomicity & durability of distributed transc$^n$. A DB must guarantee that all statements in a transc$^n$, distributed or non-distributed, either commit or rollback as a unit.

In DDBMS, a given transc$^n$ is submitted at some one site, but it can access data at other sites as well. When a trasc$^n$ is submitted at some one site, the transc$^n$ manager at that site breaks it up into a collection of one or more sub-Transc$^n$ that execute at different sites. The transc$^n$ manager then submits these sub-trasc$^n$ to the trans$^n$ manager at the other sites and coordinates their activities. Recovery control in DDBsystem is typically based on 2PC protocol or 3PC.

### a) Two-phase Commit (2PC) →

- It is the simplest & most widely used technique for recovery & concurrency control in DDB environment.
- It guarantees that all database servers participating in a distributed transc$^n$ either all commit or all abort.
- As the name implies, 2PC protocol has two phases
  1) Voting phase
  2) Decision phase

1. Voting Phase → In this, the sub-trancs$^n$ are requested to vote on their readiness to commit or Abort.
2. Decision Phase → In this, a decision as to whether all sub-tranc$^n$ should commit or abort is made & carried out.



The Basic principle of 2PC is that any of the transc$^n$ manager involved (including the coordinator) can abort a trasc$^n$. when a msg is sent in 2PC, it signals a decision by the sender. In order to ensure that this decision survives a crash at the sender's site, the log record describing the decision is always forced to stable storage before the msg is sent.

A log record is maintained with entries such as type of record, the transc$^n$ identification & the identity of coordinator. when a system comes back after crash, recovery process is invoked. The recovery process reads the log & processes all transc$^n$ that were executing the commit protocol at the time of crash.

Limitations

- A failure of the coordinator of sub transc$^n$ can result in the transc$^n$ being blocked from completion until coordinator is restored.
- Requirement of coordinator results into more msg & more overhead.

Three phase Commit (3PC) →

- An extension of 2PC
- 3PC is nonblocking for site failure, except in case all sites get failure
- 3PC avoids blocking even if coordinator site fails during recovery.
- 3PC " " by introducing an extra 3rd phase where multiple sites are involved in the decision to comit.
- In 3PC, coordinator postpones the decision to commit until it ensures that at least predetermined no. of sites know that it intended to comit the transc$^n$. If the coordinator fails, remaining sites first select a new coordinator. This new coordinator checks the status of protocol from the remaining sites. If earlier coordinator had decided to commit at least one of other predetermined sites that it informed is made up and ensures that the commit decision is respected. The new coordinator restarts 3rd phase of the protocol if some site knew that the old coordinator intended to commit transc$^n$, otherwise the new coordinator aborts the transc$^n$. Adv:- 3PC doesn't Block sites Disad. 3PC add overhead & cost.