optimising parallel queries by considering all alternatives is much more expensive. So, there are two heuristic approaches which reduces number of the execution plans that we have to consider.

(a) Only consider evaluation plans that parallelize every operation across all processors, & that do not use any pipelining. This approach is used in the Teradata system.

(b) First choose the most efficient sequential evaluation plan, & then to parallelize the operations in the evaluation plan.

* DISTRIBUTED DATABASE

* (1) Homogeneous Database :- All sites have identical database management system software, aware of one another & agree to co-operate in processing users requests.

(2) Heterogeneous Database :- opposite to Homogeneous.

* Distributed Data Storage :-

Two approaches to store a relation in database :

① Data Replication :- In this, if we want to store a relation r on the database then the several copies of r are created & stored on some sites, may be on every site in the system, then it is called as full replication.

*Advantages & Disadvantages :

- Availability :- system can continue irrespective of any nodes failure.
- Increased parallelism :-
- Increased overhead on update.

② Data Fragmentation :- a relation is revided in n number of fragments & that stored on various sites.

Two types of fragmentation :-

(a) horizontal fragmentation : A table is divided into subtables.

(b) Vertical fragmentation : In this the attributes are fragmented.

These two [illegible]

* Transperany: To hide where & how the data are physically located nor how data can be accessed at the specific local site, this called transperency.
- Fragmentation transperency: Users are not required to know how relations are fragmented.
- Replication transperency: Users not to know which data objects are replicated & where they are placed.
- location transperency: Not required to know physical location of data.

• There is a problem of having same name of distinct data items in two sites. To solve this we can have name server, but it is inefficient as if the server crashes all other operation on sites will be stopped. The other solution is to add prefix of each site to the names of data items, but this will not ensure the location transperency. So the ultimate solution is to have aliases in the sites for the data items created by the data-base systems. User access that database using data-items using the aliases. Mapping of aliases to the real names is done on the site.

Distributed Transactions:-

- System Structures:- every site has its own local transaction manager who ensures ACID properties of the transactions on that system.

Consider a abstract model of a transaction system, in which each site contains two subsystems:

(a) Transaction Manager: manages the execution of transaction which aces data stored on that local site. Trasaction can be global or local.

(b) Transaction coordinator: coordinates the execution of various transactions initiated at that site.

Responsibilities of TM:-

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

Responsibilities of TC:

- Starting the execution of transaction.
- Breaking the transaction into a number of subtransactions & distributing these subtransactions to the appropriate sites.
- Coordinating termination of the transaction, which may be commit or abort to all sites.

* System failure modes:-
  - Failure of a site
  - Loss of messages
  - Failure of communication link
  - Network partition.

* Commit Protocols:-
  (1) 2 Phase Commit :- ~~It is the simplest~~
  - mostly used in DDB
  - Mainly two phase:
    1) Voting phase: In which, sub-transactions are requested to ~~state~~ tell their readiness.
    2) Decision phase: In this, a decision as to whether 'all sub-transaction should commit or abort.
  - It like: either abort or commit.
  - any TM involved can abort the trans.
  To ensure that this msg will survive sender's side crash the log record describing the decision is always forced to stable storage.

(2) Three Phase Commit Protocol:-
- 3PC is non-blocking for site failure, except in case all sites get failure.
- 3 Phases. In 3PC:
  1) Same as 2PC.
  2) In this all sites sends msg to Ci, if at least k nodes send the commit msg then Ci precommit the

transaction make log of that force that to stable storage & then sends msg to all participants of commit. Then all participants acknowledge the msg & if at least k nodes acknowledge then third phase is executed else transaction is aborted.

(3) In third phase, if in second phase at least k nodes send commit msg then third phase is executed. In this the transaction is commited & record of log is forced to stable storage & sends commit msg to all participants.

* Distributed query processing:-

- Query transformation: Means a trivial query like "select * from 'account' relation" can be will be complecated to process on distributed system as the tuples may be fragmented or replicated or both. If 'account' relation is not fragmented then we have to choose replica with lowest cost. But if it is fragmented then it is difficult as we have to compute several joins or unions to reconstruct the relation.

- Simple Join Processing
- Semijon Stratergy

- Semijoin Strategy:

$r_1 \bowtie r_2$ (R_1 & R_2 are schemas)

$= \Pi_{R_1 \cap R_2}(r_1) = t_1$ (say)

$t_1 \bowtie r_2 = t_2$

$t_2 \bowtie r_1 = r_1 \bowtie r_2$

as there will overhead, if we shift r_2 to site S_1 to calculate joins if there are many tuples of r_2 that do not join with any tuple of r_1. So we first take intersection of schema attributes & then first join it with r_2 & that result join with r_1 at site S_1.

- Join Strategies that Exploit Parallelism.

* Recovery & Concurrency Protocol: If recovery is done after any crash, then if there are in-doubt transactions, means there is <ready T> log for transaction is there but neither commit or abort log is there, then they will block all other new transactions until they resolve. So solution to this is lock. at the time of writing logs instead of <ready T> we will write <ready T, L> & because of that at the time of recovery new transactions can occur concurrently which does not require lock of in-doubt transactions.

lock request to

* Majority protokol: It sends ↑ more than half nodes where Q is replicated for lock of Q. 2(n/2 +1) msg are involved for locking & (n/2 +1) msgs are involved in unlocking.

* Biased protocol: ~~#~~ Same as majority but shared locks are more favorable.
  - shared lock: only request lock to one site where one copy of Q is stored.
  - exclusive lock: requests lock to all the sites where replicas of Q are stored.

* Quorum Consensus Protocol: generalization of the majority protocol.
  Q_r = read quorum Q_w = write quorum
  S = weight all sites where x resides.
  $Q_r + Q_w > S$ & $2 * Q_w > S$