MASTER OF COMPUTER APPLICATIONS

SEMESTER 1

RELATIONAL DATABASE MANAGEMENT SYSTEM

Unit 6

Adaptive Query Processing and Query Evaluation

Table of Contents

1. INTRODUCTION

In the previous unit, you studied query execution and its various related aspects such as physical-query-plan operators, nested-loop joins, and various algorithms. You also studied how to use heuristics in query optimisation. In this unit, we will introduce you to adaptive query processing and query evaluation.

With the diversification of data management field into more complicated settings, where queries are becoming increasingly complex, the traditional optimise-then-execute paradigm is proving insufficient. This has led to new techniques, usually placed under the common standard of 'Adaptive Query Processing', which focus on applying runtime feedback to modify query processing in order to provide better response time or more efficient CPU usage.

Adaptive query processing refers to a set of procedures used to correct the inherent flaws of traditional query processing. It is used for creating an optimal query plans in situations when the traditional plans fail.

In this unit, you will study many of the common techniques, and approaches associated with **Adaptive Query Processing**. Our goal in this unit is to provide not only an overview of each technique, but also a basic framework for understanding the field of adaptive query processing in general. We focus primarily on processing mechanism, eddy architecture, query processing algorithms, complexities, synchronisation barriers, robust query processing through progressive optimisation. We conclude with a discussion of query evaluation techniques for large databases and query evaluation plans.

1.1 Objectives

After studying this unit, you should be able to:

- *Explain the Eddy architecture and how it allows for extreme flexibility*
- *Discuss the properties of query processing algorithms*
- *Identify and demonstrate the need and uses of adaptive query processing*
- *Explain different types of complexity*
- *Discuss synchronisation barriers in query processing*
- *Identify Robust query processing through progressive optimisation*

- *Demonstrate query evaluation techniques for large databases*
- *Discuss query evaluation plans*

2. QUERY PROCESSING MECHANISM: EDDY

Information Resources can display erratic characteristics in shared-nothing databases and big federated databases. During the process of query processing, assumptions that are made at the time when a query is submitted will rarely hold. Therefore, execution techniques and traditional static query optimisation are not very effective in such environments.

In this unit, we will introduce about a query processing process known as eddy. It constantly reorders operators in a query plan. Pipelined joins can be easily reordered, and the synchronisation barriers that require inputs from various sources can be flawlessly coordinated by characterising the moments of symmetry (discussed in section 6.4).

We can merge the execution as well as an optimisation phase of query processing by combining eddies with suitable join algorithms. This allows each tuple to have a flexible ordering of the query operators. This flexibility is controlled by combining a simple learning algorithm along with algorithms similar to fluid dynamics as in the river. River defined here is basically a dataflow query engine. It is analogous in certain ways to Volcano, and Gamma or commercial parallel database engines.

A more conceptual implementation of an adaptive query processing operator refers to Eddies. The basic idea is that the purpose of the addition of an eddy is to control various operators in a query. The fact that data involved from these operations (running as independent threads) moves through the eddy, is apparent from the Figure 6.1.

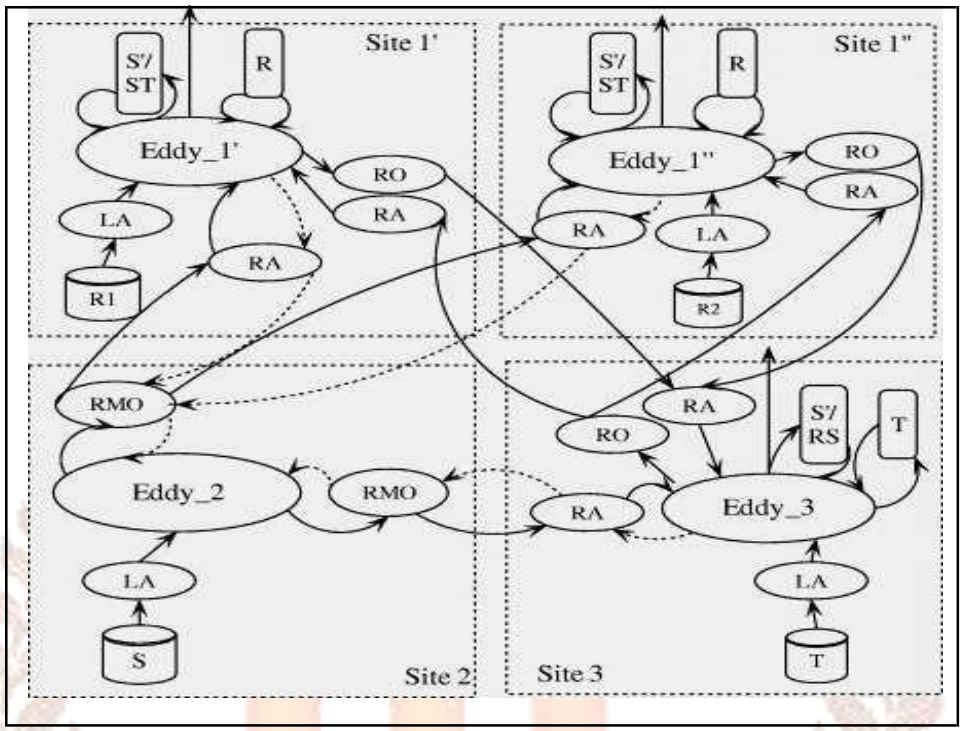

Fig 6.1: Query Processing Mechanism – Eddy

An eddy can adaptively choose the best possible order to route tuples as well as run each successive operator since it can function as a central unit between each operator.

The eddy maintains a priority queue of all tuples that require processing. Their priority level increases as tuples move from one operator to the next. This finally ensures that tuples at later stages of processing are processed first. Moreover, a tuple has to adjust its priority based on the production and consumption rates of the operators that it needs to be processed by. A low- cost operator represents a smaller percentage of the total processing time required and can be given more tuples in a shorter duration of time. The reason is that a low-cost operator can consume tuples faster as compared to a high-cost one.

By tracking the rates at which tuples are routed through them, the eddy learns the operators' relative performance.

SELF-ASSESSMENT QUESTIONS – 1

1. An *eddy* is a query processing mechanism which constantly reorders operators in a query plan as it runs. (True/ False)
2. The eddy functions by _____.

3. EDDY ARCHITECTURE AND HOW EDDY ALLOWS EXTREME FLEXIBILITY

We discussed the Query Processing Mechanism: Eddy in section 6.2. Now we will study Eddy Architecture and how it allows for the extreme flexibility. The discussion in the previous section allows us to consider easily reordering query plans at moments of symmetry (discussed in section 6.4).

River and eddies

In this section, we will illustrate the eddy mechanism during query processing for implementation of reordering in a natural manner. The techniques that we are going to describe can be used with any operator. However, more frequent re-optimisation algorithms are allowed by frequent moments of symmetry. Before the discussion on eddies, we will first introduce our basic query processing environment.

River: Eddies are implemented by us in the context of River. It is a shared- nothing parallel query processing framework. It adjusts dynamically to any fluctuations in workload and performance.

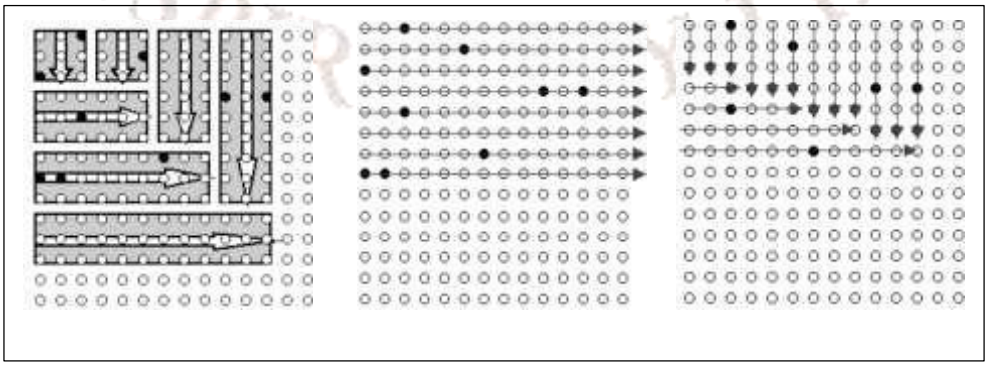

Fig 6.2: Tuples Generated by Block, Index and Hash Ripple Join

All tuples are formed by the join in block ripple. This is depicted in Figure 6.2. However, by the join predicate, some of them may be eliminated. The arrows for hash ripple join and index are symbols of the logical portion of the cross-product space checked so far. These joins only expend work on tuples satisfying the join predicate (black dots). In the hash ripple diagram, the arrival rate of one relation is three times faster than the other.

A River can be used to robustly produce near-record performance on I/O- intensive benchmarks such as hash joins and parallel sorting. It can be used in spite of the dissimilarities and dynamic variability in workloads across machines in a system as well as hardware components.

In Telegraph, our intention is to leverage the adaptability of River to allow dynamic shifting of load (including both data delivery and query processing) in a shared-nothing parallel environment.

A rather simple overview of the River framework will serve our purpose, since we are not discussing parallelism here. In these database engines represented as river, "iterator"-style modules (query operators) communicate via a fixed dataflow graph (a query plan).

With the edges in the graph matching to finite message queues, each module runs as an autonomous thread. The faster thread blocks on the queue waiting for the slower thread to catch up, if the producer and the consumer are running at differing rates.

River is essentially multi-threaded. By reading from various inputs at self- determining rates, it can take advantage of barrier free algorithms. The River implementation used by here is derived from the work on Now-Sort. It features well-organised I/O mechanisms together with high-performance user-level networking, pre-fetching scans as well as avoidance of operating system buffering.

Pre-Optimisation: How to originally pair off relations into joins, with the constraint that each relation takes part in a single join has to be decided by a heuristic pre-optimiser, even though we will use eddies to reorder tables between joins,. This corresponds to choosing a spanning tree of a query graph, in which edges symbolise binary joins and nodes symbolise relations.

A chain of Cartesian products across tables known to be very small is formed by one rational heuristic for selection of a spanning tree. The small tables are chosen so as to handle "star schemas" when base-table cardinality statistics are available. The chain then selects as many random non-equijoin edges as necessary to complete a spanning tree. This is done only after selecting random equijoin edges (on the assumption that they are relatively low selectivity).

The pre-optimiser needs to decide join algorithms for every edge for a given spanning tree of the query graph. It can use either an index join if an index is available, or a hash ripple join, along each equijoin edge. It can use a block ripple join along each non-equijoin edge.

An Eddy in the river: An eddy is implemented via a module in a river containing a single output relation, an arbitrary number of input relations, and a number of participating unary and binary modules.

The participating operators of an eddy are encapsulated by the scheduling. The tuples entering the eddy can flow through its operators in a wide range of orders. In other words, based on the intuition that symmetries can be easily captured in an -ary operator, an eddy merges multiple unary and binary operators into a single -ary operator within a query plan.

A fixed-sized buffer of tuples that is to be processed by one or more operators is maintained by an eddy component. Each operator which participates in the eddy has an output stream that returns tuples to the eddy and either one or two inputs that are fed tuples by the eddy.

Eddies derive their name from the circular data flowing within a river. A tuple entering an eddy is related with Done bits and a tuple descriptor containing a vector of Ready bits. Those operators that are eligible to process the tuple and those that have already processed the tuple are specified by them.

A tuple was shipped by the eddy module ships, only to those operators for which the corresponding Ready was bit turned on. The operator returns the tuple to the eddy after processing it, and the corresponding Done bit is turned on. The tuple is sent to the eddy's output if all the Done bits are on; or else, it is sent to a different eligible operator so as to achieve continued processing.

An eddy zeroes the Done bits, and sets the Ready bits suitably when it receives a tuple from one of its inputs. In the uncomplicated case, the eddy sets all Ready bits on. The fact that any ordering of the operators is acceptable is signified by this.

As soon as there are ordering constraints on the operators, the eddy turns on only the Ready bits matching to operators that can be executed at first. The eddy turns on the Ready bit of any operator qualified to process the tuple when it returns a tuple to the eddy.

Binary operators produce output tuples that match to combinations of input tuples. In such cases, the Ready bits and Done bits of the two input tuples are read. In this manner the ordering constraints are preserved by an eddy while at the same time, maximising opportunities for tuples to follow diverse possible orderings of the operators.

Two properties of eddies deserve comment here. Firstly, note that eddies do not constrain reordering to moments of symmetry across the eddy as a whole. A given operator must carefully abstain from fetching tuples from certain inputs until its next moment of symmetry, for instance, a nested- loops join would not fetch a new tuple from the current outer relation until it has finished rescanning the inner.

Secondly, eddies represent the full class of bushy trees corresponding to the set of join nodes. For instance, it is possible that two pairs of tuples are united independently by two dissimilar join modules, and then routed to a third join to execute the 4-way concatenation of the two binary records.

However, there is no obligation that all operators (apart from the one that is fetching a new tuple) in the eddy are at a moment of symmetry when this happens. Thus eddies are a little flexible both in the scenarios in which they can rationally reorder operators, and in the shapes of trees that they can produce.

SELF-ASSESSMENT QUESTIONS – 2

3. An eddy zeroes the Done bits, and sets the Ready bits suitably when it receives a tuple from one of its inputs. (True/ False)
4. Two properties of eddies are _____

Activity 1

Discuss how Eddy Architecture allows extreme flexibility in query processing.

4. PROPERTIES OF QUERY PROCESSING ALGORITHMS

Different properties of query processing algorithms are:

- Reorderability plans
- Synchronisation barriers in query processing
- Moments of symmetry
- Joins and indexes
- Physical properties, predicates and commutativity
- Join algorithms and re-ordering

Now let us discuss them in detail.

Reorderability plans: An essential challenge of run-time re-optimisation is to reorder pipelined query processing operators while they are in flight. To alter a query plan on the fly, a great deal of state in a variety of operators has to be well thought-out, and subjective changes can necessitate important processing and code complexity to warranty accurate results. For instance, the state that is maintained by an operator like hybrid hash join can grow as large as the size of an input relation.

Besides, it also requires modification or re-computation if the plan is reordered while the state is being constructed. We can keep this work to a minimum by restraining the scenarios in which we reorder operators. Since in a highly variable setting, the best-case scenario rarely exists for a significant length of time period, it is better to favour adaptivity over best-case performance. In idealised query processing, therefore it is better to sacrifice marginal improvements.

Synchronisation barriers in query processing: A significant state is usually captured by binary operators similar to joins. In such operators, the interleaving of requests for tuples from different inputs relates to an exact form of state used.

As an example, let us consider the case of a merge join on two duplicate- free, sorted inputs. The next tuple is always consumed, during processing, from the relation whose last tuple had the lower value. This constrains the order in which tuples can be consumed significantly.

As an example, let us consider the case of a slowly-delivered external relation 'slow-lo' with a high-bandwidth but large local relation 'fast-hi' with only high values in its join column and numerous low values in its join column. The processing of fast-hi is postponed for a long time while consuming many tuples from slow-lo.

We describe this phenomenon as a synchronisation barrier by using terminology from parallel programming. One table-scan produces a value larger than any seen before, while the other table-scan waits.

Generally, barriers limit performance when two tasks take different amounts of time to complete (i.e., to "arrive" at the barrier) by limiting concurrency. It is noteworthy here that concurrency arises even in single-site query engines, which can carry out disk I/O, network I/O and computation simultaneously.

Therefore, in a dynamic (or even heterogeneous but static) performance environment, it is desirable to minimise the overhead of synchronisation barriers. There are two issues which affect the overhead of barriers in a plan. They are: a) the gap between arrival times of the two inputs at the barrier; and the frequency of barriers. b).

Moments of symmetry: It is noteworthy that the synchronisation barrier in merge join is declared in an order-independent manner. It does not discriminate between the inputs based on any property other than the data that they convey.

Since its two inputs are treated uniformly, a Merge join is often described as a symmetric operator. Let us take the example of a traditional nested-loops join. In a nested-loops join, the "inner" relation is synchronised with the "outer" relation, but not vice versa. A barrier is set until a full scan of the inner is completed after each tuple (or block of tuples) is consumed from the outer relation.

Performance benefits can often be obtained by reordering of the inputs for asymmetric operators like nested-loops join.

A join algorithm declares the end of a scheduling dependency involving its two input relations when it reaches a barrier. Without altering any state in the join, the order of the inputs to the join can often be changed in such scenarios. When this is true, the barrier is referred to as a moment of symmetry.

Let us again consider the example of a nested-loops join, with inner relation S and outer relation R. (See Figure 6.3) Having joined each tuple in a subset of R with every tuple in S, the join has completed a full inner loop at a barrier. Without affecting the join algorithm, reordering the inputs at this point can be done.

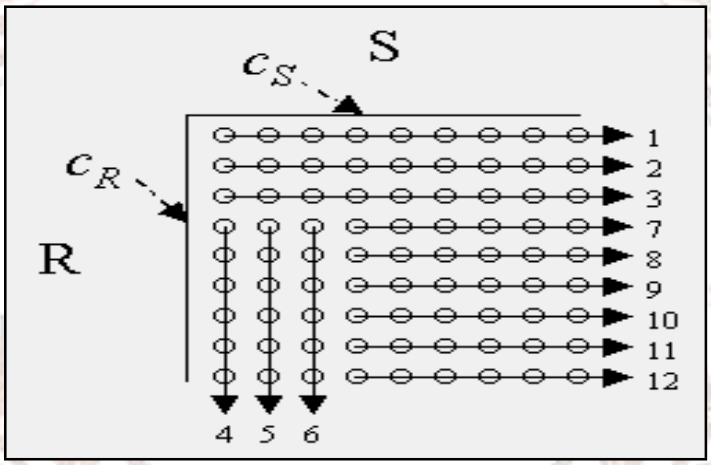

Fig 6.3: Tuples Generated by Nested-Loops Join

Joins and indexes: Benefit of indexes on the inner relation can be taken by Nested-loops joins. These result in a fairly efficient pipelining join algorithm. Since one input relation has been pre-indexed, an index nested-loops join or an "index join" is essentially asymmetric..

Changing the alternative of inner and outer relation "on the fly" is difficult even when indexes exist on both inputs. Therefore, it is simpler to think of an index join as a type of unary selection operator on the unindexed input for the objective of reordering.

The fact that with respect to the unindexed relation, the selectivity of the join node may be greater than one is the only difference between an index join and a selection. We can reorder an index join and its indexed relation as a unit among other operators in the plan tree, even though one cannot swap the inputs to a single index join.

It must be noted that the logic for indexes can be useful to external tables that need bindings to be passed. Such tables may be gateways to web pages.

Physical properties, predicates and commutativity: Undoubtedly, the possible join orderings are constrained by a pre-optimiser's choice of an index join algorithm. In the join view, the unindexed join input is ordered before (though not necessarily directly before) the indexed input since an ordering constraint must be obligatory.

This constraint develops because of a physical property of an input relation. Indexes cannot be scanned but can be probed. Hence, they cannot appear before the corresponding probing tables. More complex but similar constraints can arise in preserving the ordered inputs to a merge join (i.e. preserving "interesting orders").

Additional constraints are raised by the applicability of certain join algorithms. Many join algorithms will not work on joins other than equijoins. Since they always require all relations mentioned in their equijoin predicates to be handled before them, such algorithms constrain reordering on the plan tree as well.

Join algorithms and re-ordering: Join algorithms with frequent moments of symmetry, adaptive or nonexistent barriers, and minimal ordering constraints have to be most favourable, for an eddy to be most successful.

These algorithms present the best opportunities for re-optimisation. The need to avoid blocking, rules out the use of hybrid hash join, minimises ordering constraints and barriers excluded merge joins. Nested loops joins are undesirable because they have imbalanced barriers and occasional moments of symmetry.

Ripple joins have moments of symmetry at every "corner" of a rectangular ripple, i.e., whenever all tuples in a prefix of input stream join a prefix of the input stream and vice versa.

This scenario occurs between each consecutive tuple consumed from a scanned input for hash ripple joins and index joins. Recurrent moments of symmetry are thus offered by the ripple joins.

Ripple joins are also attractive with respect to barriers. In order to allow changing rates for each input, ripple joins were designed. The reason was to proactively expend more processing on the input relation with more statistical influence on intermediate results.

The same mechanism also allows reactive adaptivity in the wide-area scenario. A barrier is reached at every single corner, whereas the next corner can adaptively be a sign of the relative rates of the two inputs.

In case of block ripple join, the next corner is selected upon reaching the previous corner. This can be done adaptively to reflect the relative rates of the two inputs over time.

At a modest overhead in performance and memory footprint, the ripple join family offers attractive adaptivity features. Therefore, they fit well with the principle of sacrificing marginal speed for adaptability. As a result, there is focus on these algorithms in Telegraph.

SELF-ASSESSMENT QUESTIONS – 3

5. Ripple joins were designed to allow changing rates for _____.
6. At a modest overhead in performance and memory footprint, the ripple join family offers attractive adaptivity features. (True/ False)
7. The possible join orderings are constrained by a pre-optimiser's choice of an index join algorithm. (True/ False)

Activity 2

The next tuple, during processing, is always consumed from the relation whose last tuple had the lower value. Explain this statement with the help of suitable examples and figures if necessary.

5. NEED AND USES OF ADAPTIVE QUERY PROCESSING

Now let us study what is adaptive query processing and where it is most appropriately used.

Declarative queries were key value proposition of relational model, in which the user chooses the data to be queried and the database management system will work out the correct algorithm for retrieving the data from the data store. The normal method of doing this is cost based query optimisation. Figure 6.4 demonstrates the common Traditional Query Processing.

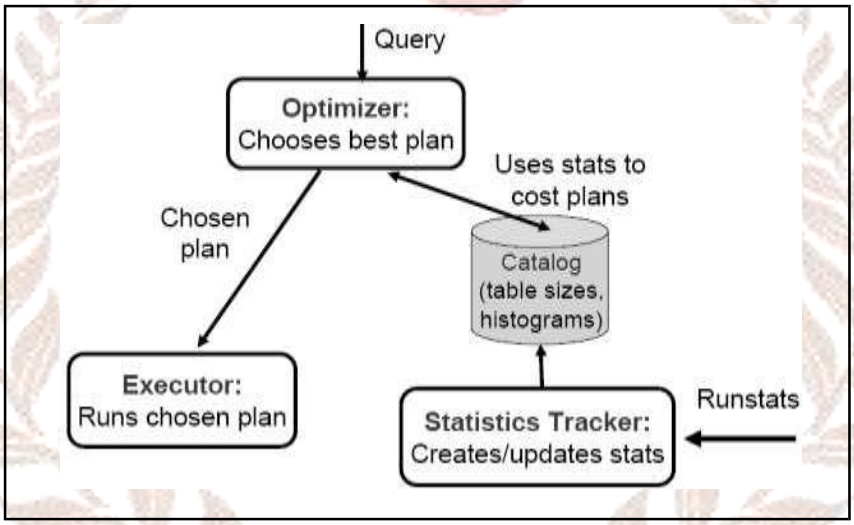

Fig 6.4: Traditional Query Processing

Following three tasks are involved in Traditional Query Processing.

- ***Optimisation:*** Optimiser selects a plan to accomplish a query using available statistics.
- ***Execution:*** Executor executes the plan to get query results.
- ***Statistics Tracking:*** A statistics tracker keeps the statistics utilised by the optimiser.

Adaptive systems are also known by "self-tuning" or "dynamic" systems; i.e. the systems that modify their behaviour by the use of "introspection", "learning", etc. The query processing system is said to be adaptive if it has three features:

- It obtains information from its environment
- It applies this information to decide its behaviour,
- This process repeats again and again, resulting in a feedback loop among environment and behaviour.

Static optimisation has the first two of these features. The feedback required in an adaptive system is solution to its effectiveness. Figure 6.5 demonstrates the general structure of the adaptive query processing.

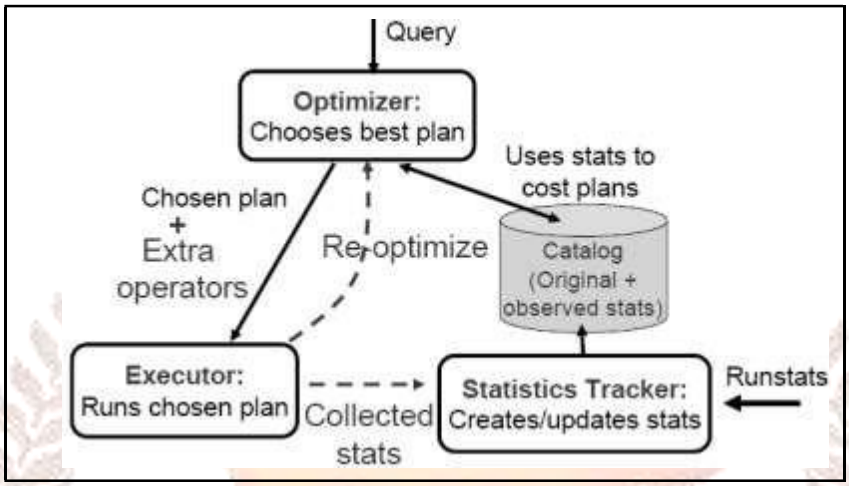

Fig 6.5: Adaptive Query processing in Adaptive Environment

Now let us study the main applications of Adaptive Query Processing. These are:

Adaptive query processing in data grids: The data grid integrates wide- area autonomous data sources and provides users with a unified data query and processing infrastructure. Adaptive data query and processing is essentially used by data grids to offer better quality of services (QoS) to users and applications in spite of dynamically changing resources and environments.

Adaptive query processing in internet applications: Data management in the Internet has gained a lot of popularity. The recent focus is on efficiently dealing with transfer rates and unpredictable, dynamic data volumes by making use of adaptive query processing techniques. For various query processing domains, an equally vital consideration is the high degree of variability in performance needs.

Adaptive query processing in web based data integration: Mediators for web-based data integration require the capability to handle multiple, often conflicting objectives such as cost, coverage and execution flexibility. This requires the development of query planning algorithms as well as techniques for automatically gathering the obligatory cost/coverage statistics from the independent data sources.

To detect and correct optimiser: Adaptive query processing has been utilised to detect and correct optimiser errors due to wrong statistics or simplified cost metrics.

Adaptive query processing in wide-area database systems: In broad- area database systems, that run on changeable and volatile environments (such as computational grids), it is difficult to produce efficient database query plans depending upon information available only at compile time. A potent solution to this difficulty is by adjusting the query plan to varying conditions during execution as well as by making use of information that becomes available at query run-time.

SELF-ASSESSMENT QUESTIONS – 4

8. Adaptive query processing has been utilised to detect and correct optimiser errors due to wrong statistics. (True/ False)
9. Adapt data query and processing is required by _____to provide better quality of services (QoS) to users.

6. COMPLEXITIES

In the database systems, since very large query engines function in changeable and unpredictable environments, using traditional techniques for adaptive query processing takes the engine to the breaking point. This volatility is common in large-scale systems, on account of increased complexity. These complexities are:

Hardware and workload complexity: Variabilities are frequent in the busty performance of servers and networks in wide-area environments. These systems often serve as big communities of users whose aggregate behaviour can be really hard to guess, with the hardware mix being quite heterogeneous.

Analogous performance variations can be displayed by large clusters of "shared-nothing" computers, owing to a mix up of heterogeneous hardware evolution and user requests.

Hardware performance can be volatile even in entirely homogeneous environments. For example, the inner tracks of a disk might demonstrate just half the bandwidth of outer tracks.

Data complexity: Selectivity estimation for static alphanumeric data sets is rather understood very well. Besides, through complex methods and types, there has been preliminary work on estimating statistical properties of static sets of data. However, federated data hardly comes with any statistical summaries. Multifaceted non-alphanumeric data types are now broadly in use both on the web and in object-relational databases. Selectivity estimates are usually imprecise in these scenarios as well as in traditional static relational databases.

User Interface Complexity: Many queries can run for a very long time in large-scale systems. Consequently, there is interest in Online Aggregation and other methods that permit users to "Control" properties of queries at the time they execute depending upon refining approximate results.

SELF-ASSESSMENT QUESTIONS – 5

10. Many queries can run for a very long time in large-scale systems. (True/ False)
11. In wide-area environments, variabilities are frequent in _____.

7. ROBUST QUERY OPTIMISATION THROUGH PROGRESSIVE OPTIMISATION

In some cases the query optimiser has a choice between a "conservative" plan that is likely to perform reasonably well in many situations, or a more aggressive plan that works better if the cost estimate is accurate, but much worse if the estimate is slightly off.

The requisite probability distributions over the parameters can be calculated by use of histograms or query workload information. It is evidently a more robust optimisation objective, with an assumption that only one plan can be selected and the required probability distributions can be obtained.

Error-aware optimisation (EAO) utilises intervals over query cost estimates, rather than specifying the estimates for single. EAO focuses mainly on memory usage ambiguity. A later work, it provides several features including the use of intervals. It generates linear query plans (a slight variation of the left-linear or left-deep plan, in that one of the two inputs to every join – not necessarily the right one – must be a base relation) and uses bounding boxes over the estimated cardinalities in order to find and prefer robust plans.

Another way of making plans more robust is to utilise more sophisticated operators, for example, n-way pipelined hash joins.

SELF-ASSESSMENT QUESTIONS – 6

12. The needed probability distributions over the parameters can be calculated by use of _____or _____ information.
13. _____ utilises intervals over query cost estimates, rather than specifying the estimates for single.

8. QUERY EVALUATION TECHNIQUES FOR LARGE DATABASES

In this section we will focus on various query evaluation techniques for large databases.

Structural design of query engines: Query processing algorithms iterate on components of input sets; algorithms are algebra operators. The physical algebra is the collection of operators and data representations apart from associated cost functions that the database execution engine supports, and the logical algebra which is more associated to expressible queries of the data model (e.g. SQL) and the data model.

Transfer between operators and synchronisation between them is the key. Primitive methods consist of using one process per operator and using IPC or creation of temporary files/buffers. Implementation of all operators as a set of procedures (open, next and close), and having operators schedule each other within a single process via simple function calls is an important practical technique. An operator calls its data input operator's next function to produce another piece of data ("granule"), every time an operator needs one, *Iterators* are operators structured in such a way.

Query plans *iterators* can be symbolised as trees and are algebra expressions. The three common structures are: Bushy (arbitrary), Left-deep (every right subtree is a leaf), and right-deep (every left-subtree is a leaf) structures. In a left-deep tree, every operator draws input from only one input, whereas an inner loop iterates over the other input.

Sorting: Merging techniques are used by all sorting in "real" database systems. The structure of iterators must be followed by sorting modules' interfaces.

Exploit the duality of merge sort as well as quicksort. Sort proceeds in *divide* stage and the *combine* stage. One of the two phases is based on logical keys (indexes), the physically arranged data items (this phase is logical and is particular to an algorithm).

There are two types of sub algorithms: one for sorting a run within main memory and another one is meant for managing runs on disk or tape. Degree of fan-in (which basically refers to the number of runs merged in a given step) is a key factor.

For creation of the set of initial (level-0) runs, quicksort and replacement- selection are the two algorithms of choice. Replacement assortment fills memory in a priority heap in which the smallest key is written to a run and replaced from the next input.

This replacement may be bigger than the just written item, so we can then iterate; unless put mark replacement for next run file. RS (replacement- selection) has smooth alternates between read and write operations. In contrast, Quicksort has bursty I/O pattern.

Level-0 runs and level-1 runs are mixed together. Buffer space must be dedicated to each input run as well as the merge output. A cluster is the unit of I/O.

Hashing: In general, hashing must be considered for equality matches. Hashing-based query processing algorithms utilise in-memory hash table of database objects. Hash table overflow occurs if the data in hash table is larger than main memory (common case).

Assigning hash buckets to partitions desires to be optimised so that disk accesses result in clustered buckets both logically as well as physically.

Disk Access: File scans can be made quickly with read-ahead (track-at-a- crack). This will require contiguous file allocation, so may need to bypass OS/file system.

Advantages of disk access:

a) It is possible to scan an index without ever retrieving records, i.e. if just salary values are needed and the index is on salary.
b) Multiple indices can be joined to satisfy query requirements, even if none of the indices is enough by itself.
c) Take union/intersection of two index scans if two or more indices apply to individual clauses of a query.
d) Joining of two tables can be achieved by joining indices on the two join attributes and then doing record retrievals in the underlying data sets.

Buffer management: Cache data in I/O buffer. LRU is not right for a lot of database operations. Iterator implementations can take benefit of buffer management mechanisms which typically provide fix/unfix semantics on a buffer page, when passing buffer pages amongst themselves.

SELF-ASSESSMENT QUESTIONS – 7

14. In-memory hash table of database objects are utilised by hashing- based query processing algos. (True/ False)
15. Iterator implementations can take benefit of buffer management mechanisms which _____.
16. It is _____ to scan an index without ever retrieving records.

9. QUERY EVALUATION PLANS

A query evaluation plan (or simply plan) consists of an extended relational algebra tree, with extra annotations at every node indicating the implementation method to use for each relational operator and the access methods to utilise for each table.

SELECT S.sname
FROM Reserves R, Sailors S
WHERE R.sid = S.sid
AND R.bid = 100 AND S.rating > 5

This query can be expressed in relational algebra as follows:

$\pi sname(\sigma bid = 100 \wedge rating > 5(Reserves \rhd\lhd_{sid=sid} Sailors))$

In Figure 6.6, this expression is illustrated in the form of a tree. The algebra expression partly specifies how to evaluate the query, we first calculate the natural join of Reserves and Sailors, then performs the selections, and finally projects the sname field.

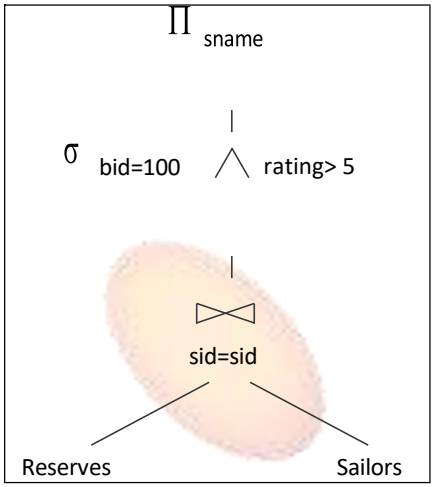

Fig 6.6: Query Expressed as a Relational Algebra Tree

To obtain a fully specific evaluation plan, we should decide on an implementation for each of the algebra operations involved. For example, we can use a page-oriented simple nested loops join by means of Reserves as the outer table and apply selections and projections to every tuple in the result of the join as it is formed. The result of the join previous to the selections and projections is never stored entirely. This query evaluation plan is illustrated in Figure 6.7.

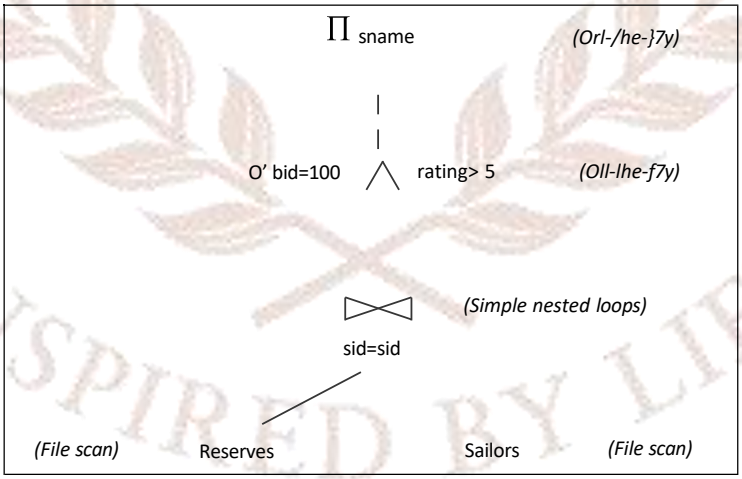

Fig 6.7: Query Evaluation Plan for Sample Query

In drawing the query evaluation plan, we have utilised the convention that the outer table is the left child of the join operator. We adopt this convention henceforth.

SELF-ASSESSMENT QUESTIONS – 8

17. The algebra expression partly does not evaluation of the query-owe. (True/ False)
18. To get a fully specific evaluation plan, we should decide on a _____ for each of the algebra operations involved.

10. SUMMARY

Let us recapitulate the important points discussed in this unit:

- Eddy is a query processing mechanism. It constantly reorders operators in a query plan as it runs.
- The techniques that we described in this unit can be used with any operator, but algorithms with recurrent moments of symmetry allow for more recurrent re-optimisation.
- Reorder of pipelined query processing operators during flight is the basic challenge of run-time re-optimisation.
- The data grid integrates wide-area autonomous data sources and provides users with a unified data query and processing infrastructure.
- The required probability distributions over the parameters can be computed using histograms or query workload information.
- Naive methods like creation of temporary files/buffers, using one process per operator and using IPC are used.
- A query evaluation plan (or simply plan) contains an extended relational algebra tree.

11. GLOSSARY

- ***Binary Operator:*** Binary operators similar to joins often capture significant state.
- ***EAO:*** Error-aware optimisation. EAO considers intervals of estimates and proposes heuristics to identify robust plans. However, the techniques in EAO assume a single uncertain statistic (memory size) and a single join.
- ***Index nested-loops:*** One input relation has been pre-indexed because an index nested-loop join (henceforth an "index join") is inherently asymmetric.
- ***LRU:*** Least Recently Used. This rule may be used in a cache to select which cache entry to flush. It is based on temporal locality - the observation that, in general, the cache entry which has not been accessed for longest is least likely to be accessed in the near future.
- ***QOS:*** Quality of services. It refers to several related aspects of telephony and computer networks that allow the transport of traffic with special requirements.
- ***Ripple Joins:*** The purpose of ripple joins is to allow changing rates for each input.
- ***XML:*** Extensible Markup Language. It is a markup language that defines a set of rules for encoding documents in a format that is both human- readable and machine-readable.

12. TERMINAL QUESTIONS

1. Explain the eddy architecture and how it allows for extreme flexibility.
2. What are the basic properties of query processing algorithms?
3. Discuss where the adaptive query processing is most widely used?
4. Explain the query evaluation techniques for large databases.

13. ANSWERS

Self-Assessment Questions

1. True
2. Eddy
3. False
4. Ordering
5. Arrival times
6. False
7. True
8. True
9. Data grids
10. False
11. Federated
12. Histograms, Query Workload
13. Error-aware optimisation (EAO)
14. True
15. Hash function
16. False
17. False
18. Implementation

Terminal questions

1. In eddy architecture, we implemented eddies in the context of River, a shared-nothing parallel query processing framework. Refer Section 2 for more details.
2. Reorder ability plans and moments of symmetry are the properties of query processing. Refer Section 4 for more details.

3. Adaptive query processing is used in internet application and data grids. Refer Section 5 for more details.
4. Query evaluation techniques for large databases are hashing and sorting. Refer Section 8 for more details.

14. REFERENCES

References:

- Raghu Ramakrishnan, Johannes Gehrke, *Database Management Systems*, (3rd Ed.), McGraw-Hill
- Peter Rob, Carlos Coronel, *Database Systems: Design, Implementation, and Management*, (7th Ed.), Thomson Learning
- Silberschatz, Korth, Sudarshan, *Database System Concepts*, (4th Ed.), McGraw-Hill
- Elmasari Navathe, *Fundamentals of Database Systems*, (3rd Ed.), Pearson Education Asia

E-reference:

- http://db.cs.berkeley.edu/papers/sigmod00-eddy.pdf
- http://www.it.iitb.ac.in/