IOSR Journal of Computer Engineering (IOSR-JCE)
e-ISSN: 2278-0661, p-ISSN: 2278-8727, Volume 27, Issue 3, Ser. 2 (May. – June. 2025), PP 46-55
www.iosrjournals.org

Analysis Of INGRES Algorithm For Efficient Distributed Query Processing And Optimization For Performance Improvement

Mohammad Kamal Hossain Foraji, Md. Humayun Kabir, Md. Golam Moazzam
[1,2,3]*Department Of Computer Science And Engineering, Jahangirnagar University, Bangladesh*

Abstract:
This paper investigates distributed query optimization techniques and presents the features of centralized and distributed INGRES algorithms for efficient query processing and performance improvement of the algorithms. The original input query is analyzed and decomposed into a series of sub-queries. It uses detachment and substitution techniques to optimize the subqueries. In each step a monorelation query and a multirelation query are generated. The monorelation query can be evaluated using one variable query processor (OVQP). When the multirelation query cannot be further decomposed, the substitution method uses tuple substitution, and the INGRES algorithm terminates after evaluation of the resultant query. The goal of the distributed INGRES algorithm is to reduce communication time and response time. In a distributed INGRES system, the master site-the site where the user submits or initiates the query-is responsible for coordinating the execution of the distributed query. All monorelation queries that are detachable are initially handled locally. The original query is then processed using the reduction technique. In this research work, an input multirelation query is simplified using the detachment and substitution technique which can be automated to incorporate in the query compiler for optimization in future.

Keywords: *One Variable Query Processor, Query Execution Plan, Dynamic Query Optimization, Distributed Database, Search Strategy, Multirelation Query, Detachment, Substitution, Transmission Cost, Irreducible Query, Optimizer, Indexing, Replication, Rewriting.*

Date of Submission: 11-05-2025 Date of Acceptance: 21-05-2025

I. Introduction

Query optimization is a process of creating a query execution plan (QEP) that represents the query execution strategy. A query optimizer, or software module that optimizes query, consists of three parts: a search space, a search strategy, and a cost model [1, 2]. In query optimization, the search space refers to all the possible execution plans that a query optimizer might consider when trying to find the most efficient way to execute a query.

In this research, the performance of the centralized and the distributed INGRES algorithm is studied in distributed database environment. Distributed INGRES takes into consideration both communication costs and local processing costs, allowing relations to be fragmented among multiple sites [2]. Consider an input query Q of the following form, where p2 is a multirelation predicate.

select $r_2.a_2, \ldots r_m.a_m$
from $r_1, r_2, \ldots r_m$ ………………………… *(1)*
where $p_1(r_1.a'_1)$ *and* $p_2(r_1.a_1, \ldots r_m.a_m)$

The input query is processed using the reduction technique which uses detachment to seperate all irreducible and monorelation subqueries [3]. The reduction procedure generates a list of irreducible subqueries at most one relation in common. The INGRES algorithm divides a query into subqueries [2]. Distributed INGRES is the extension of centralized INGRES algorithm. It minimizes the size of intermediate results to create a monorelation and a multirelation query [3].

The query optimizer dynamically constructs query execution plan [4]. It also communicates the database management system (DBMS) execution engine to perform query operations. The dynamic query optimization in INGRES is designed to process queries efficiently, even in distributed environments where factors like network traffic, distributed data locations, and varying system resources affect performance. The system continuously evaluates the current environment and adjusts the query plan to ensure optimal execution [3].

The database consists of some tables, fragments and few replicas. An example distributed query is simplified by detachment and substitution technique. The replication of the table in distributed environment will ensure data availability to promote efficient management of university administration activities smartly and reliably which will contribute to society for better quality assurance of educational institution [5].

II. Background Study

There are so many techniques involved in query optimization in distributed databases. Some techniques are discussed below.

i. Data Replication

Data replication is normally desired since it allows to copy data across a network of various machines for the performance, dependability, and availability of data segment. Such replication improves performance by allowing for easier accommodation of various user requirements [6]. For example, data that can be stored on local systems as well as remote systems via a communication network so that remote users can access and update when necessary. This enhances the locality of references and even if one of the workstations fails, data is backed up on the other system on communication network. The example query is as shown below [7].

create snapshot replica as
select sname, cname
from student@dblink, courses@dblink;

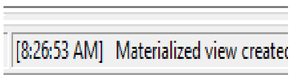
This SQL query creates a replica of the Campus1 database on Campus2 for extraction of the required data efficiently. For emergency purposes, this data will be available on remote sites and increase the query processing efficiency. It will also reduce the query processing time of the distributed system. Replication data is as shown in Figure 2 below.

*select * from replica* (Q_{71})

ii. Indexing

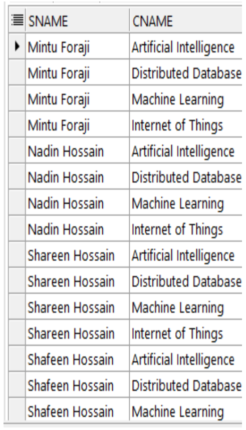
By indexing the query of distributed database, it also can minimize the query processing time. Figure 3 shows the CPU processing time. The following query Q_{81} is implemented for counting the number of attributes of the replica table. The queries Q_{91} and Q_{101} find the CPU cost for processing the query and compares the performance of CPU cost.

select count (*) from replica ……………… Q_{81}
*explain plan for select * from replica ………………..… Q_{91}*
*select * from table(dbms_xplan.display) ……….….. Q_{101}*



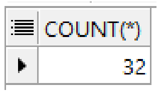
```
[8:26:53 AM]  Materialized view createc
```

Figure 1: Creation of Replication



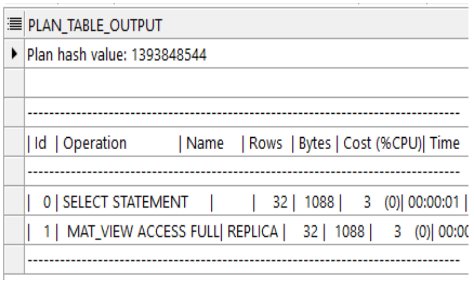
SNAME	CNAME
Mintu Foraji	Artificial Intelligence
Mintu Foraji	Distributed Database
Mintu Foraji	Machine Learning
Mintu Foraji	Internet of Things
Nadin Hossain	Artificial Intelligence
Nadin Hossain	Distributed Database
Nadin Hossain	Machine Learning
Nadin Hossain	Internet of Things
Shareen Hossain	Artificial Intelligence
Shareen Hossain	Distributed Database
Shareen Hossain	Machine Learning
Shareen Hossain	Internet of Things
Shafeen Hossain	Artificial Intelligence
Shafeen Hossain	Distributed Database
Shafeen Hossain	Machine Learning

Figure 2: Output of Replication of Q_{71}.



COUNT(*)
32

Figure 3: Record Count



```
PLAN_TABLE_OUTPUT
Plan hash value: 1393848544

-----------------------------------------------------------------------------
| Id  | Operation           | Name     | Rows  | Bytes | Cost (%CPU)| Time
-----------------------------------------------------------------------------
|   0 | SELECT STATEMENT    |          |    32 |  1088 |     3   (0)| 00:00:01 |
|   1 |  MAT_VIEW ACCESS FULL| REPLICA |    32 |  1088 |     3   (0)| 00:00
-----------------------------------------------------------------------------
```

Figure 4: Output of Indexing

iii. Rewriting Technique in Optimization

Query rewriting is a process that transforms a query into an optimized version that produces the same results. The goal of query rewriting is to improve the query's performance by making it easier to optimize. It can be especially useful for complex queries with many joins or subqueries. It rewrites the query in relational algebra. This is divided into two steps: 1) easy transfer of the query from relational calculus to relational algebra and 2) restructuring the relational algebra query for better performance [3].

III. Related Work

Distributed INGRES is the first dynamic approach of distributed query optimization. It makes a better solution to accurately capture the data distributions effectively based on the heuristic technique [8].

When joining operations do not use a foreign key, it is extremely difficult to assess their selectivity. Join selectivity factors can be used to solve the problem [9]. The prior commercial DBMSs introduced the transmission costs only for query processing. If faster communication networks are available on the distributed databases (DDBS), local query processing costs are more important factors for performance improvement. Semi-joins are generally used as they increase the local processing time of the query operation, and it is better than joining approaches for query optimization [10].

When a relation contains most of the tuples in a distributed database system, using a semi-join is often the most appropriate technique for reducing the amount of data that must be transferred across the communication network. The semi-join works by first transferring only the necessary data to filter out irrelevant tuples, thereby minimizing the size of the data involved in the actual join operation. So, the solution is to use the semi-joins [11], and it would be a very efficient technique to transfer multimedia data in a communication network [9]. In the INGRES algorithm, it gets some advantages of the communication networks. In [12], different optimal solutions have been provided, and analysis have been done based on the distributed INGRES algorithm. This method also improves communication time using the proposed optimization technique. Among the optimization techniques, enhanced semi-connection query optimization keeps more optimization efficiency than general semi-connected algorithms [13]. It drastically reduces the intermediate local cost as well as the total cost of communication networks.

An easy operational prototype is proposed for effective query improvement. It combines portability and workability to execute an operative query optimizer for the new creation of distributed databases [14]. Using the heuristic algorithm, the cost of query optimization can be reduced for distributed queries. Their queries are distributed into tree queries and cyclic queries. The best approach for turning cyclic queries into tree queries is horizontal and vertical decomposition of relations [13]. Communication cost is one of the main parameters to reduce query relevant data and processing cost in distributed query optimization [15]. The query optimization approach decreases the communication cost and hence reduces the distributed query response time. Finally, the distributed database confirms the faster local query processing time for its smooth operation.

IV. Query Optimization Using Ingres Algorithm

There are two techniques of query optimization based on the database management environment which are centralized and distributed query optimizations. Centralized query optimization minimizes the query response time and maximizes the system throughput. The query execution plan minimizes an objective cost function [1-3]. Distributed query optimization is an extension of centralized query optimization. Query optimization emphasizes the following features: Search Space, Search Strategy and Cost Model

As an example, we clarify the features using an example query in distributed environment.

select student. sname
from student, assignedcourses, courses
where student.rno=assignedcourses.rno (Q_1)
and assignedcourses.cno=courses.cno
and courses.cname='Machine Learning'

Search Space

The search space represents the set of possible execution plans for the original query. For any given SQL query, the search space is the set of all equivalent operator trees that can be derived from the original query using a series of transformation rules [3]. There are three join trees such as linear trees, bushy trees and Cartesian products are involved in this regard. Join trees which start with Cartesian products may be substantially more expensive than alternative join trees. Two types of change using transformation rules in a tree are possible: i) operand position change and ii) operator change. The combination of associativity and permutation of join orders dramatically increases the number of possible operator trees, and this is what makes query optimization such a complex problem-especially as the number of joins grows. Restructuring operator trees using transformation rules may result in equivalent operator trees as given in Figure 5 below. Permutation of join order also affects the performance of the relational queries.

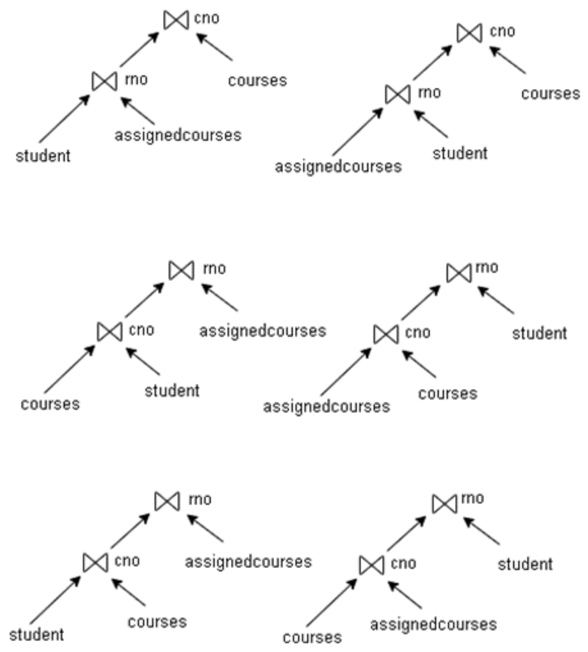


Figure 5: Different Types of Linear Trees

In this scenario, the operands are changed according to the cyclic order. It is proved that for the operands number 1, 2, 3, 4 and 5, the number of trees is 1, 2, 6, 12 and 20 respectively. In a distributed database, when a query is executed, it may need to search multiple nodes or data partitions. It refers to all the possible locations of data that the query could access. Optimizing query execution often involves minimizing the search space by determining which nodes or partitions to access based on the query, data location, and indexing mechanisms.

Search Strategy

Search Strategy in the context of distributed databases refers to the methods or approaches used to locate and retrieve data efficiently across multiple distributed nodes. Since data is spread across different machines or locations in a distributed system, the strategy must account for factors like data partitioning, query routing, replication, fault tolerance, and load balancing. A deterministic search approach widely used by query optimizers - especially in traditional relational database systems - is indeed dynamic programming. This method is especially effective for join order optimization, where the goal is to find the most cost-efficient plan among many possibilities [3]. Deterministic strategies follow a systematic and layered approach to plan generation starting with base relations and adding one additional table in every stage till the plans are completed, shown in Figure 6. It is a powerful technique for solving optimization problems where the outcome is fully predictable based on previous actions. By breaking a complex problem into simpler sub problems, it can efficiently find the optimal solution. However, it requires careful consideration of the state space and memory requirements, especially for large or complex problems. We can see that only one tree is found in the following step as given in Figure 6.

Cost Model

The cost model of an optimizer relies on cost functions for predicting operator costs, statistics and base data, and algorithms for determining the sizes of intermediate outcomes. The cost function represents the execution time of a query. The cost of a distributed execution approach can be represented in terms of either total time or response time. The total time is the sum of all time units, where the response time is the absorbed time from the initiation to the completion of the query. A general formula for determining the total time can be specified as follows [18]:

Ttotal = TCPU * #insts + TIO * #IOs + TMSG * #msgs + TTR * #bytes

Here, TCPU and TIO measure the local processing time and TMSG and TTR measure the communication time to transfer a data unit from one location to another. Let's see the Figure 7.

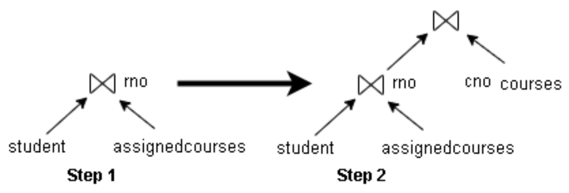


Figure 6: Equivalent Join Trees

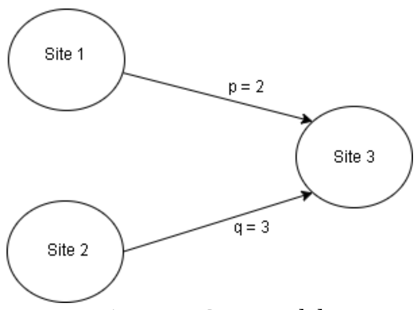


Figure 7: Cost Model

Assume TMSG and TTR are measured as time costs. Total time required to move p data units to site 3 from site 1 and q data units from site 2 that would be Ttotal = 2xTMSG + TTR (p+q). So, response time for identical query is estimated by Tresponse=max {TMSG+TTR*p, TMSG+TTR*q} as parallel data transfers are a key optimization in distributed databases.

If the TMSG and TTR are 2 seconds and 3 seconds respectively and the value of x and y are 1 and 2, the total time is Ttotal = 2x2 + 3(1+2) = 13 seconds and the response time T response = max{2+3x1, 2+3x2} = max{5, 8} = 8 seconds. The system may then choose to perform a distributed join, where part of the join is performed locally at each node, or may choose a different execution plan based on the cost model. Finally, a cost model in distributed databases is a critical part of the system's optimization strategy. It helps balance factors such as latency, bandwidth, disk access, replication overhead, and computational resources to achieve high performance and efficient resource usage.

There are two algorithm techniques used here to solve the query optimization in distributed databases. They are Centralized and Distributed INGRES Algorithms. Now, we can see the Centralized INGRES Algorithm first.

A) Centralized INGRES Algorithm

INGRES applies the query optimization techniques dynamically for recurrently splitting the query into shorter segments [2]. Consider that an m-relation query *Q* is divided into m subqueries $Q_1 \rightarrow Q_2 \rightarrow \ldots \rightarrow Q_m$. Every query relation r_i is a monorelation, and hence ri+1 contains the output of ri relation. Two basic approaches, detachment and substitution, are used for decomposing the main query ri. A query is initially divided into a set of subqueries that have a common relation. The monorelation queries are then processed by a one-variable query processor [1, 2, 3].

The detachment splits a query *Q* into $Q' \rightarrow Q''$. The above SQL query *Q* even in (1) is split by the detachment technique into two subqueries. In the first subquery *Q'*, attribute a1 of the relation r_1 builds a new intermediate relation ir'_1 based on the predicate p1. The other subquery *Q''* may contain this temporary relation in special faster memory to increase its efficiency [2, 3].

i. Database and Network Configuration

The distributed database configured among four campuses of ABC University located in four different cities, and the databases are distributed among Campus1, Campus2, Campus3 and Campus4 in a communication network as shown in Figure 8.

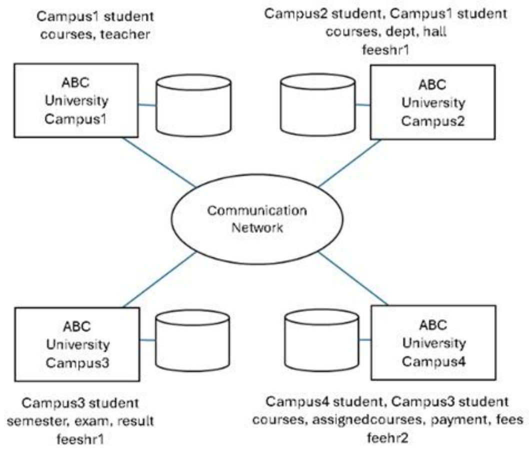


Figure 8: DDBS Environment of ABC University.

Campus 1, 2, 3 and 4 all are server PCs, containing distributed databases. In Figure 8, all campuses are located in different places or cities in a country. According to the above configuration, the distributed databases contain the tables or relations student, teacher, dept, hall, courses, assignedcourses, fees, semester, exam, result, payment and accounts in different campuses of the ABC University as depicted in Figure 8. Some data are stored in the tables of ABC University databases where every campus has its own students. From any campus, students information can be inserted, deleted, modified and updated from other remote sites. Here, Campus1 is the master site. To maintain the consistency of the database, triggers or procedures are written executing at the master site which automatically maintains the safe state of the database stored in all the sites. The attributes of the relations are given below.

student (*rno, sname, dept, nid, dob, brn, origin*)
teacher (*tid, tname, origin*)
dept (*did, dept, origin*)
hall (*hid, hall, origin*)
courses (*cno, cname, cfees, origin*)
assignedcourses (*rno, cno, title, field, dur, origin*)
fees (*cno, tfees, origin*)
semester (*sid, sem, origin*)
exam (*edate, cno, cname, origin*)
result (*rno, dept, sid, gpa, cgpa, grade, origin*)
payment (*rno, sid, year, cno, origin*)
accounts (*rno, account_id, balance, origin*)

ii. Application of Detachment Technique

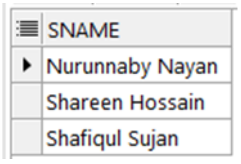
Different distributed databases have their own data in different campuses. The detachment technique is explained with an example below.

Example 1

Find the students' names who take "Machine Learning" courses.
This query is defined in SQL language using the query Q_1 as follows.

select student.sname
from student, assignedcourses, courses
where student.rno=assignedcourses.rno (Q_1)
and assignedcourses.cno=courses.cno
and courses.cname='Machine Learning'

The output of query Q_1 is given below in Figure 9.



SNAME
Nurunnaby Nayan
Shareen Hossain
Shafiqul Sujan

Figure 9: Output of query Q_1.

The QEPs differ in the sequence in which operations are executed and implemented and hence improve the performance of the query operations. A cost model is a framework used by a query optimizer to estimate the cost of different execution strategies for a query. The goal is to choose the strategy with the lowest estimated cost, ideally leading to the best performance. It must include a thorough understanding of the distributed domains. Search strategy uses cost models to navigate the search space and select the optimal plan [3].

Figures 10 (a) and (b) show the query graph and join graph for query Q_1.

.

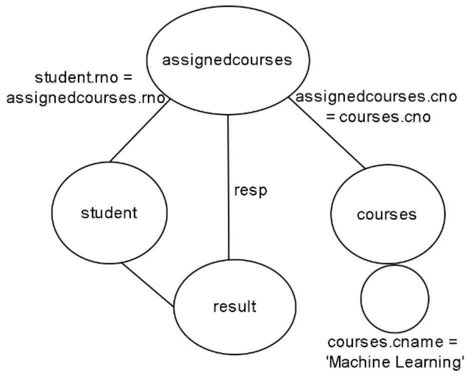

assignedcourses
student.rno = assignedcourses.rno
assignedcourses.cno = courses.cno
resp
student
courses
result
courses.cname = 'Machine Learning'

Figure 10 (a): Query Graph.

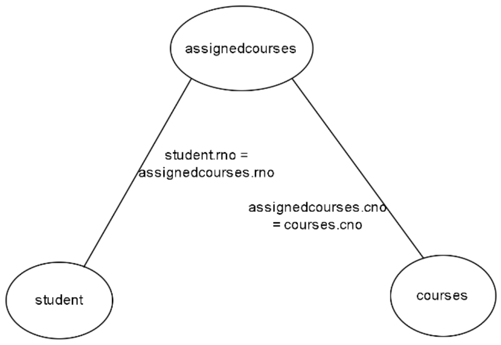

assignedcourses
student.rno = assignedcourses.rno
assignedcourses.cno = courses.cno
student
courses

Figure 10 (b): Join Graph

The search trees generated for query Q_1 are shown in Figure 11 (a), (b) and (c).

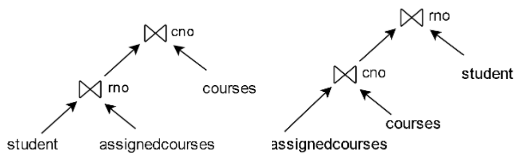


Figure 11 (a): Linear Trees.

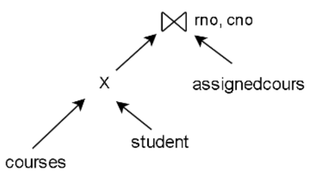


Figure 11 (b): A tree generated with Cartesian operator.

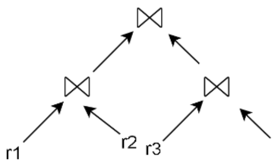


Figure 11 (c): Bushy Trees.

From the Figures 11 (a), (b) and (c), linear trees can be thought of as a special case of a tree or a sequence of decisions. This structure can arise when there's only one feasible action or choice at each decision point in a search problem. It is with one path; data conflicts and version inconsistencies are minimized. The bushy tree is a general and may include operators with two intermediary operands. In a distributed context, bushy trees help to show parallelism. A Cartesian tree in a search space is a specialized binary tree often used for efficient searching, sorting, and range queries. This is beneficial in tasks requiring frequent segment-based queries, such as finding minimums in a sliding window or interval-specific calculations making them efficient for interval problems, like finding minimum or maximum values in subarrays. In this case, it produces the cross products between courses and student data based on the assignedcourses where rno is equal to cno [3].

Query optimizers usually limit the extent of the search space they evaluate. Common heuristics are selection, projection and avoidance of Cartesian products. An essential restriction is the geometry of the join tree [4, 5, 16]. Join trees are often divided into two types: linear trees and bushy trees. By focusing on linear trees, the search space decreased to O(2N). Bushy trees, on the other hand, can be beneficial in demonstrating parallelism in a distributed context.

The detachment technique decomposes a query Q_1 into Q_{11} and Q' with the intermediate relation *ir1*. Q_{11} is defined as follows.

select courses.cno
from courses (Q_{11})
where courses.cname='Machine Learning';

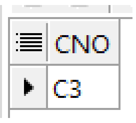
Query Q_1 is decomposed into Q_{11} and Q_{12} subqueries. It splits the single table into multiple tables ensuring consistency in the data. By the query Q_{11}, only the course number "C3" processed from the given dataset. Figure 12 shows the output of query Q_{11} by using the following select query.

*select * from* ir_1*;*

The other subquery Q' is defined as follows.

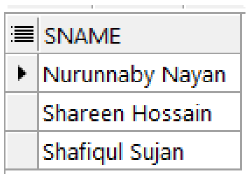
select *student.sname* (Q')
from *student, assignedcourses, ir1*
where *student.rno = assignedcourses.rno*
and *assignedcourses.cno = ir1.cno;*

The output is of *Q'* as follows.



CNO
C3

Figure 12: Output of query Q_{11}.



SNAME
Nurunnaby Nayan
Shareen Hossain
Shafiqul Sujan

Figure 13: Output of query *Q'*.

The detachment of *Q'* generates the subqueries Q_{12} and Q_{13}.

```
select assignedcourses.rno into ir2
from assignedcourses, ir1                    (Q12)
where assignedcourses.cno = ir1.cno
```

Note that the intermediate relations ir1 and ir2 must be created before executing the subqueries Q11 and Q12. These intermediate relations are created using the following SQL expressions.

```
create table ir1 as
(select courses.cno
from courses
where courses.cname = 'Machine Learning')

create table ir2 as
(select  assignedthesis.sno
from  assignedthesis, ir1
where  assignedthesis.cno=ir1.cno);
select *  from ir2                                   (Q12)
```

The other subquery Q_{13} is defined as follows and its output as shown in Figure 15.

```
select student.sname
from student, ir2                             (Q13)
where  student.sno=ir2.sno;
```

Here, Q_{11} is a single-relation query and can be performed by OVQP where Q_{12} and Q_{13} are multirelation queries. These queries cannot further be reduced by detachment [3] technique as they are not monorelation queries.

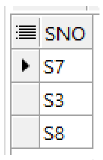
iii. Application of Substitution Technique

Multirelation queries, which cannot be further detached, are irreducible. Tuple substitution converts irreducible queries to monorelation queries. Given an *m-relations*' query *Q*, the substitution technique produces a set of (*m-1*) variable queries. In query processing using views, especially in query rewriting, the tuple substitution technique generates new queries by substituting tuples from a known result or intermediate relation [3].

Assume that ir_2 relation has the tuples {S1, S2}. Now, rewrite Q_{13} with tuple substitution into Q_{131} and Q_{132}.

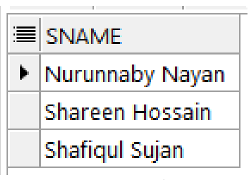
```
select student.sname
from student                       (Q131)
where student.rno='S1';

select student.sname
from student                       (Q132)
where student.rno='S2';
```

As Q_{12} is a multirelational query, it is also decomposed into Q_{13}, and it finds the student's name from the relation student and ir2. The query Q_{13} is decomposed into Q_{131} and Q_{132} until the mono-relation queries are produced. After the query execution of Q_{131} and Q_{132}, different student's names are selected using the select, from and where conditions. The output of the query Q_{131} and Q_{132} are shown in Figures 16 and 17.



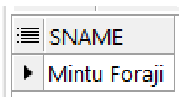
SNO
S7
S3
S8

Figure 14: Output of query Q_{12}.



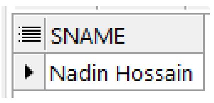
SNAME
Nurunnaby Nayan
Shareen Hossain
Shafiqul Sujan

Figure 15: Output of query Q_{13}.



SNAME
Mintu Foraji

Figure 16: Output of query Q_{131}.



SNAME
Nadin Hossain

Figure 17: Output of query (Q_{132}).

Now, the above queries Q_{131} and Q_{132} both are monorelation queries, and hence they can be evaluated by OVQP.

B) Distributed INGRES Algorithm

The distributed INGRES divides every query *Qi* into more subqueries that use fragmentations. All fragments are to be mapped into separate database locations.

To run the following PLSQL query, a 'dblink' is created among the four campuses Campus1, Campus2, Campus3 and Campus4. The following SQL statement is used for creation of database link on Campus1so that the user of Campus2, Campus3 and Campus4 can access and update the master site Campus1 data.

create database link dblink connect to student1 identified by abcd using 'campus2';

After creating this database link, logout from the Campus1 database site and login into student1@Campus2 for the following query processing. The following result is as shown on PLSQL result grid.

*select * from fees@dblink* (Q_{21})

iii) Fragmentation

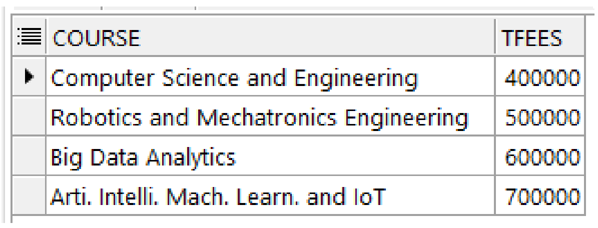
Fragmentation is a typical practice that divides each database relation into smaller fragments that are treated as individual database objects or relations. This is often done for reasons of performance, availability, and dependability.

For applying horizontal fragmentation, a partition is created as table 'hf1' on Campus2 database as given below.

create table hf1 as
*select * from fees@dblink*
where tfees>=500000;

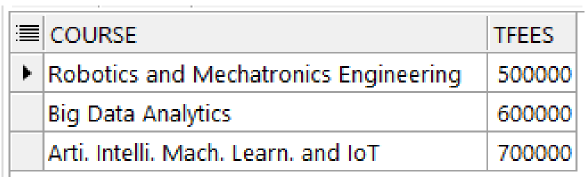
The output of this fragment (hf1) is shown by the following query Q_{22}.

*select * from hf1* (Q_{22})



COURSE	TFEES
Computer Science and Engineering	400000
Robotics and Mechatronics Engineering	500000
Big Data Analytics	600000
Arti. Intelli. Mach. Learn. and IoT	700000

Figure 18: Output of the query Q_{21}.



COURSE	TFEES
Robotics and Mechatronics Engineering	500000
Big Data Analytics	600000
Arti. Intelli. Mach. Learn. and IoT	700000

Figure 19: Output of the query Q_{22}.

For horizontal fragmentation, another partition is created as table 'hf2' on Campus2 database as given below.

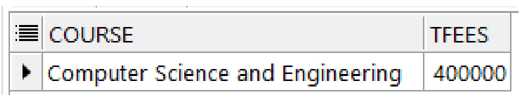
create table hf2 as
*select * from fees@dblink*
where tfees<500000;

The value of this fragment (hf2) is shown by the following query Q31.

*select * from hf2* ……………….... (Q_{31})

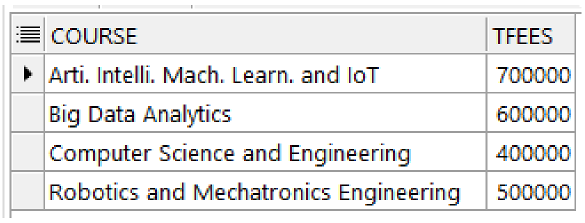
Using union operator, Q_{21} and Q_{31} are merged and created a table 'hf' on Campus2 database as shown below.

create table hf as
*select * from hf1*
union
*select * from hf2;*
*select * from hf* ………..………………. (Q_{41})



COURSE	TFEES
Computer Science and Engineering	400000

Figure 20: Output of the query Q_{31}.



COURSE	TFEES
Arti. Intelli. Mach. Learn. and IoT	700000
Big Data Analytics	600000
Computer Science and Engineering	400000
Robotics and Mechatronics Engineering	500000

Figure 21**:** Output of the query Q_{41}.

V. Result Analysis

All the monorelation queries and subqueries which are detached from the multirelations are processed by the PLSQL query engine. The queries and subqueries are run on Oracle SQL engine and get the required result by decomposition of queries [17]. The detachment technique reduces the relation size and query Q_1 result is found by query processing on SQL. Only three names are shown for the Q_1 query. The simple query finds student names from *student*, *assignedthesis* and *courses* tables.

Data replication increases the performance of local and distributed data processing costs. There are some techniques which are involved in query processing that improve query processing performance in distributed databases [18]. The techniques of data localization, inherent parallelism, query optimization, memory cost-reduction, fragmentation, replication of data, indexing, networking, and network transparency etc. are involved to improve the performance of distributed INGRES algorithm [19].

VI. Conclusion And Future Work

This paper investigates distributed query optimization using optimization algorithms, e.g., INGRES and R*. We simplified a distributed query using detachment and substitution technique of INGRES algorithm. It shows that the simplification reduces to an expected result, but it needs automation in a generalized approach. In the distributed INGRES algorithm, a query is splitted on relation into different subqueries. The subqueries are decomposed into different smaller subqueries. The queries are splitted until they reach into monorelation query. These new subqueries are operated on horizontal fragmentation consisting of a sub-set of the tuples (row) of a relation that is also shown in this research work. In this research work, only horizontal fragmentation is implemented. But future research will work on vertical fragmentation using distributed INGRES algorithm and effect of network topologies, media and protocols on cost model in distributed query optimization.

References

[1] C. T. Yu And C. C. Chang. Distributed Query Processing. Computing Surveys. Vol. 16, No. 4, December 1984.

[2] E. Wong And K. Youssefi. Decomposition: A Strategy For Query Processing. Acm Trans. Database Syst., 1976. 1(3): 223–241.

[3] M. Tamer Ozsu And Patrick Valduriez. Principles Of Distributed Database Systems, Springer, 4th Edition, 2019.

[4] Luc Bouganim, Francoise Fabret And Patrick Valduriez. Dynamic Query Processing Architecture For Data Integration Systems. Bulletin Of The Ieee Computer Society Technical Committee On Data Engineering, April 2001.

[5] Kishor Yadav Kommanaboina. Creating Smart City Infrastructure Using Integrated Data Pipelines. Iosr-Jce, Vol. 26, Issue 4, Series 3, 2024.

[6] Dr. Sunita M. Mahajan And Ms. Vaishali P. Jadhav. General Framework For Optimization Of Distributed Queries. International Journal Of Database Systems (Ijdms) Vol. 4, No. 3, June 2012.

[7] Vamsi Krishna Myalapalli And Bhupati Lohith Ravi Teja. High Performance Pl/Sql Programming. International Conference On Pervasive Computing (Icpc), Ieee. 8-10 Jan. 2015.

[8] Epstein, R., Stonebraker, M. And Wong, E. Query Processing In A Distributed Relational Database System. Proc. Acm Sigmod Int. Conf. Management Of Data, 1978. Pages 169–180.

[9] Mackert, L. F. And Lohman, G. R* Optimized Validation And Performance Evaluation For Local Queries. In Proc. Acm Sigmod Int. Conf. On Management Of Data, 1986. Pages 84–95.

[10] Valduriez, P. And Gardarin, G. Join And Semi-Join Algorithms For A Multiprocessor Database Machine. Acm Trans. Database Syst., 1984. 9(1):133–161.

[11] Valduriez, P. Semi-Join Algorithms For Distributed Database Machines. In Schneider, J.-J., Editor, Distributed Data Bases. North-Holland, 1982. Pages 23–37.

[12] Epstein, R. And Stonebraker, M. Analysis Of Distributed Database Processing Strategies. In Proc. 5th Int. Conf. On Very Data Bases, 1980. Pages 92–101.

[13] Fan Yuanyuan And Mi Xifeng. Distributed Database System Query Optimization Algorithm Research, Ieee, 2010. 978-1-4244-5540-9/10.

[14] Tejy Johnson And Dr. S. K. Srivatsa. Study On Optimization Techniques And Query Execution Operators That Enhances Query Performance. International Journal Of Advanced Research In Computer Science, Vol. 3, No. 3, May-June 2012.

[15] Abhijeet Raipurkar And Dr. G. R. Bamnote. Query Optimization In Distributed Database System. International Journal Of Computer Science And Applications, Vol. 6, No. 2, April 2013.

[16] Shyam Padia, Sushant Khulge, Akhilesh Gupta, Parth Khadilikar. Query Optimization Strategies In Distributed Databases. International Journal Of Computer Science And Information Technologies (Ijcsit), Vol. 6 (5), 2015, 4228-4234.

[17] Abhayanand And Dr. M. M. Rahman. An Overview Of Query Optimization In Distributed Relational Databases. International Journal Of Creative Research Thoughts (Ijcrt), Vol. 12, Issue 2, Feb 2024.

[18] Lohman, G., Mohan, C., Haas, L., Daniels, D., Lindsay, B., Selinger, P., And Wilms, P. (1985). Query Processing In R*. In Kim Et Al. [1985], Pages 31–47. 250, 277.

[19] Mohammad Kamal Hossain Foraji And Md. Humayun Kabir. Study On Performance Improvement Of Distributed Query Optimization Algorithms. Iosr Journal Of Computer Engineering, Vol. 24, Issue No. 4, 2024.