Project Report on

Distributed Query Processing Plans generation using Teacher Learner Based Optimization

under the guidance of

J R Shruti

Submitted by

Nandini AV 16IT120

Nilita Anil Kumar 16IT122

Anmol Jindal 16IT222

in partial fulfillment for the award of the degree of

BACHELOR OF TECHNOLOGY

in

INFORMATION TECHNOLOGY

Department of Information Technology

National Institute of Technology Karnataka, Surathkal.

June 2020

Abstract

With the growing popularity, the number of data sources and the amount of data has been growing very fast in recent years. The distribution of operational data on disparate data sources impose a challenge on processing user queries. In such database systems, the database relations required by a query to answer may be stored at multiple sites. This leads to an exponential increase in the number of possible equivalents or alternatives of a user query. Though it is not computationally reasonable to explore exhaustively all possible query plans in a large search space, thus a strategy is required to produce optimal query plans in distributed database systems. The query plan with the most cost-effective option for query processing is measured necessary and must be generated for a given query. This paper attempts to generate such optimal query plans using a parameter less optimization technique 'Teaching-Learner Based Optimization' (TLBO). TLBO based algorithm is able to generate comparatively better quality Top-K query plans

TABLE OF CONTENTS

.

1 Introduction

A distributed database encompasses coherent data, spread across various operational autonomous sites of a computer network [1]. With the increase in the number of users or expanding organization requirement, the size of database networks also expands. A Distributed Database Management System (DDBMS) deals with managing such distributed databases. DDBMS provides access to users via a simple and unified interface over dispersed databases through different transparency mechanisms, due to which a user feels as if they were not distributed [2].

The query processing is a prime activity in the database and it is controlled by DBMS. The performance of a DBMS is determined by its ability to process queries in an effective and efficient manner. One common way to identify the top-k objects is scoring all database objects on some scoring function [3].

There are two alternatives of distribution of data in a distributed database: full replication or partition etc. due to which a given relation can be found in more than one sites [4]. These alternatives are equivalent in terms of outcome as they retrieve the same set of database objects or records. Thus selecting best alternatives for processing from the generated pool is a decisive task to query optimizers. In DDB systems, query processing and optimization is constrained and subjective by different cost parameters such as, communication cost, local processing cost, optimization cost, query localization cost etc.
Thus the DQP strategy aims to generate query processing plans that reduce the amount of data transfer between participant sites by selecting the appropriate copy of data for query result retrieval and thereby reduces the overall distributed query response time.
Optimality on query plans is based on the function of different cost models [5], [6].

Optimization approach used here is Teacher Learner Based Optimization (TLBO). The burden of regulation on control parameters is comparatively less in the TLBO, thusTLBO is simple, effective and involves less computational effort [7], [8]. There is another advantage of using TLBO, that it uses some well established benchmark functions to evaluate the final fitness of an alternative.

2 Literature Survey

In [9], an approach that generates “close” query plans with respect to the number of sites involved and the concentration of relations in the sites for a distributed relational query is given. As per [9],[10], query processing over a lesser number of sites would be more efficient and thus query plans involving fewer sites need to be generated. Such query plans, referred to as “close” query plans, are generated using the genetic algorithm (GA) [9] [10], without considering the communication and local processing cost on optimization of QEPs. None of the existing approaches considered the fundamental cost models for optimization.

In [11],[12], [13] optimal query plans are generated according to various customized cost models, in this paper we have accommodated the localization cost as integral part of query communication cost and subsequently local processing cost based on predicate selectivity of local operator. An optimization algorithm’s performance is entirely based on algorithmic parameters [13], in this paper we have applied TLBO, which does not require tuning of any algorithm specific parametric during generating of optimal query plans.

3 Requirements Analysis

- In distributed database systems relations are spread across multiple sites, and multiple copies of a relation stored/ maintained in different sites. Here, we have taken a typical DDBS allocation of 8 database relations among 16 sites. eg. Relation R1 is stored in S1, S2, S3, S5, S6, S8, S10 and it is assumed that R1 is replicated entirely not in fragmented or partitioned form.
- SQlite3 uses python used to create the DDBMS.
- Inputs :
    - Relation-Site Matrix (RSM)- allocation schema
    - Subset of already generated query plans for a query
      Q1:SELECT a, m FROM R1, R2, R3, R4, R5, R6, R7, R8, WHERE R1.a=R4.t AND R4.p=R2.x AND R1.a=R7.q AND R2.x=R3.n AND R4.x=R5.s AND R8.w=R6.d AND R7.j=R6.k
- Output :
    - QEP (Query Execution Plans) are ranked based on the QC (Query Costs) value

4 System Design/Architecture

A typical DDBS allocation of 8 database relations among 16 sites is created. eg. Relation R1 is stored in S1, S2, S3, S5, S6, S8, S10 and it is assumed that R1 is replicated entirely not in fragmented or partitioned form.

Query optimization is done on the QP to find out the best QP. The query optimizer that follows this approach is seen as three components: A searchspace, a search strategy and a cost model. The search space is the set of alternative executions to represent the input query. These strategies are equivalent, in the sense that they yield the same result but they differ on the execution order of operations and the way these operations are implemented. The search strategy explores the search space and selects the best plan. It defines which plans are examined and in which order. The cost model predicts the cost of a given execution plan. Query optimization is done using the Teacher- Learner Based Optimization using cost parameters - Query Affinity Cost (QAC), Query Localization Cost (QLC) and Local Processing Cost (LPC).

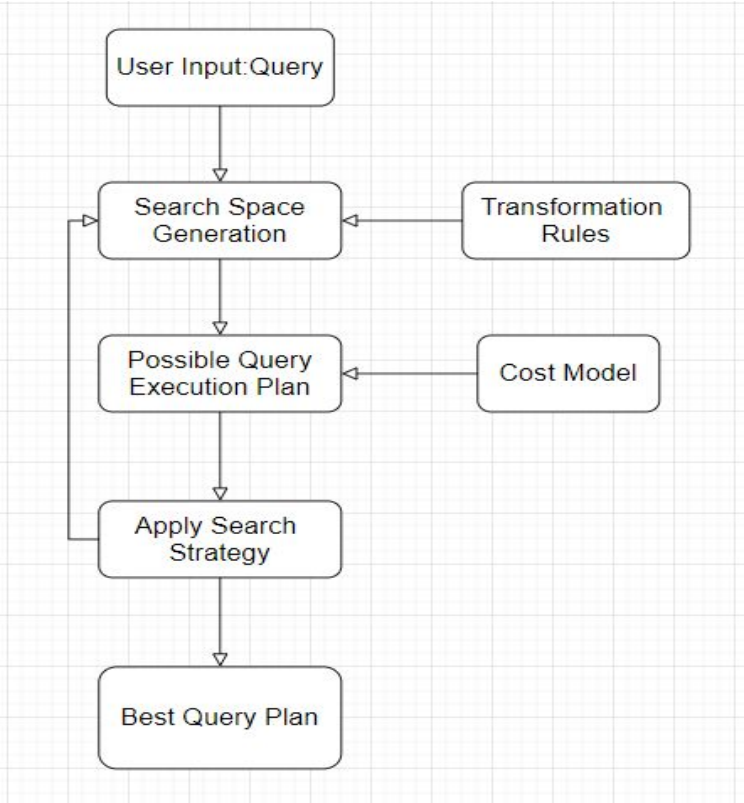


Fig 1: Query Optimizer Components

5 Methodology

- In order to process a user query in a distributed database system, the data required may have to be obtained from several sites distributed over a computer network. Furthermore, as the number of sites containing the relations accessed by the query increase, the number of possible valid query plans also increases.Thus, a large search space comprising all possible query plans needs to be explored in order to compute the optimal query plans.
- In this paper we propose cost model or design objectives for query processing based on which the optimality on QEPs are applied
- Three design objectives are applied as cost models on generated QEPs -
  - **Query Affinity Cost (QAC**): the number of sites accessed during result generation for given user query.

$$QAC = \sum_{i=1}^{M} \frac{Ki}{N}\left(1 - \frac{Ki}{N}\right)$$

M : total number of sites required in the query plan
N : total number of relations accessed by query plan
Ki : the number of times the ith site is accessed by query plan

  - **Query Localization Cost (QLC)**: QLC between two sites, S1 to S2 represents the ratio of size of relation at that site and sum of sizes of total relations in FROM clause of query.

$$QP_{Cost} = MIN_{i=1}^{Nr}\left[\sum_{j=1\&\&j\neq i}^{Nr} \frac{Size\left(R_{S_j}\right)}{\sum_{k=1}^{Nr} Size(R_k)}\right]$$

function MIN to evaluate the minimum value for i=(1 to Nr)
total number of relations accessed by the user query or number of relations in the FROM clause of user query is Nr
Size(Rsj) is the total number of data tuples in a relation present at site Sj Size(Rk) is number of tuples in relation Rk.

  - **Local Processing Cost (LPC)**: categorized two components of LPC, first is due to local processing computation on relations at remote sites and second due to local processing computation on control sites.

Relation Processing Cost-

$$RPC = N_t * S_r / \sum_{k=1}^{Nr} Nt\,(k)$$

(a)LPC for Remote Site used in Query Plan

$$RLPC = Max_{i=1\ to\ Rs}[RPC(i)]$$

(b)LPC for Control Site used in Query Plan

$$CLPC = Max_{i,j=1toNr}[\, N_t(JOIN(R_i,R_j))* S_j((JOIN(R_i,R_j))/\sum_{k=1}^{Nr} Nt\,(k)]$$

LPC of a query plan QPi

$$= \sum_{k=1}^{Sqp-1}(RLPC(k)) + (CLPC\text{-}Max_{i=1toNr\&\&}R(i)CS\&JOIN\,[RPC(i)]$$

total number of tuples is Nt
total number of relation in user query is Nr
Sqp represents the total number of sites accessed by user query
Rs is total number of relations stored in local site
Sr is selectivity measure of relation R on local site
Sj is selectivity measure of Join relational operator
Nj is the number of Joins operations for a query plan.

5. 1 Teacher Learner Based Algorithm (TLBO):

A teaching learning process inspired algorithm. based on the effect or influence of a teacher on the output of learners or students in a class.consider a group of learners as a population and different subjects offered to the learners are considered as different design objectives.fitness of the learner is having analogy with the marks obtained by the learner on the specific subject. optimization is done on the basis of fitness value. The best solution in the entire population is considered as the teacher.design objectives are actually the parameters involved in the objective function of the given optimization problem and the best solution is the best value of the objective function. The working of the TLBO algorithm is alienated into two phases, 'Teacher phase' and 'Learner phase' A teacher attempts to increase the mean result of the class in the subject he teaches, depending on his or her capability. Since the teacher is usually considered a highly learned person who trains learners so that they can have better results, the algorithm considers the best identified learner as the teacher his best learner tries to improve the mean results of the entire set of learners in the subsequent phase of the algorithm. The identification of best learner's based on the final fitness value of the learner, fitness value is evaluated by benchmark function. A learner with minimum fitness values considered best in set, as in query optimization it is desired to select a solution with lesser cost values.

- Next phase is the student phase, in which students or learners raise their knowledge level by interaction among themselves. A learner or student interacts randomly with other learners to enhance his or her knowledge.

- After a number of sequential runs of teaching–learning cycles in which, the teacher disseminates knowledge to the learners and their knowledge level increases toward the teacher's level. Knowledge level of the entire class becomes smooth and the algorithm converges to an optimal solution.

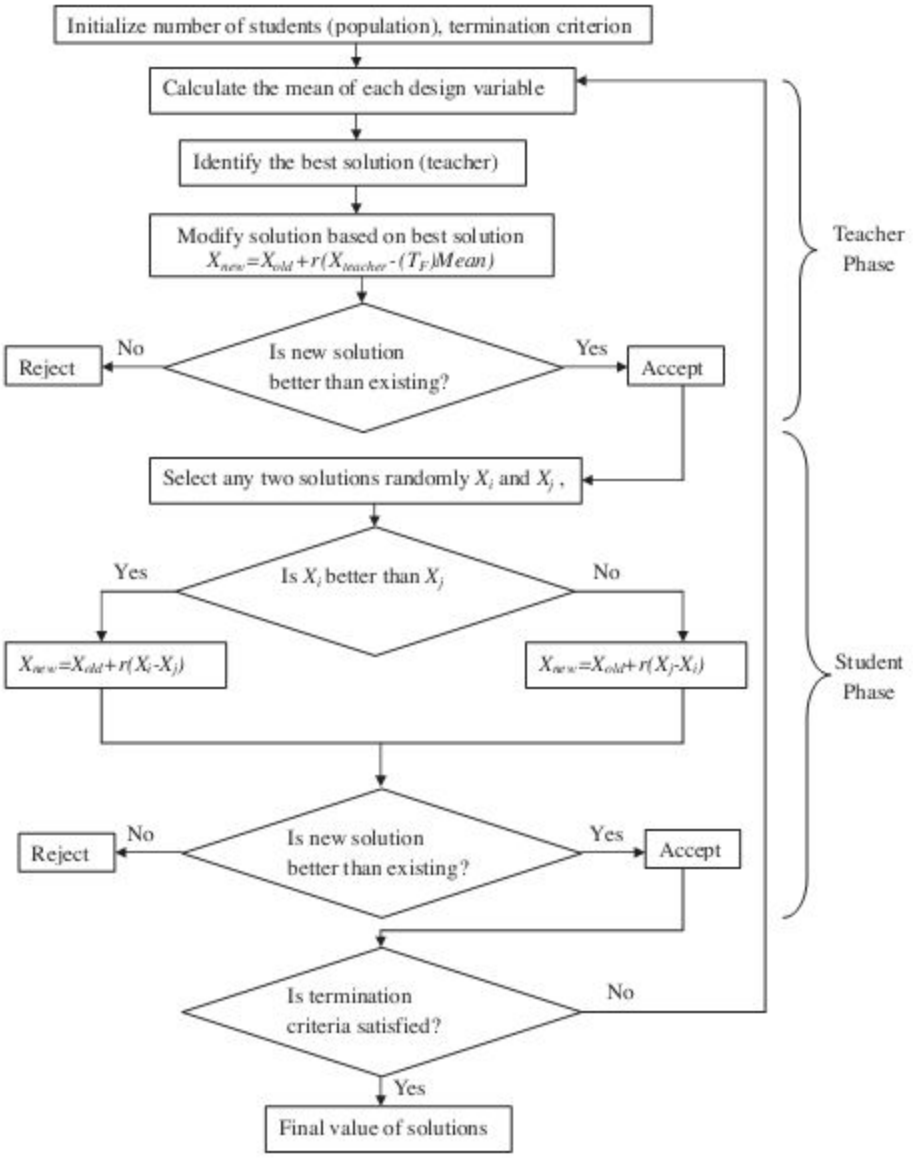


Fig 2: Schematic TLBO for Optimal Query Plan generation in Distributed Database Systems : Teacher's Phase and Learner's Phase

6 Implementation

6.1 Heuristics of Design Objective

- The initialization of query equivalents is according to the allocation schema (Relation Site Matrix).
  Eg: RSM [0] = [1, 1, 1, 1, 1, 0, 0, 0]
  Site 1 consists of R1, R2, R3, R4 and R5

```
RSM = [[1, 1, 1, 1, 1, 0, 0, 0],
       [1, 1, 1, 1, 1, 0, 0, 0],
       [1, 0, 0, 0, 0, 1, 0, 1],
       [0, 0, 0, 0, 0, 1, 0, 0],
       [1, 1, 1, 0, 0, 1, 1, 0],
       [1, 0, 0, 0, 0, 1, 1, 0],
       [0, 1, 1, 1, 1, 1, 0, 0],
       [1, 1, 1, 1, 0, 0, 1, 1],
       [0, 0, 0, 0, 0, 0, 0, 1],
       [1, 0, 1, 0, 0, 0, 0, 0],
       [0, 1, 1, 1, 0, 0, 1, 0],
       [0, 1, 0, 0, 0, 0, 0, 1],
       [0, 0, 1, 0, 0, 0, 0, 0],
       [0, 0, 0, 1, 0, 0, 0, 1],
       [1, 0, 1, 1, 1, 0, 0, 0],
       [0, 1, 1, 0, 1, 1, 1, 0]]
```

- All the generated query equivalent plans are semantically valid and retrieve the similar results for the user. Valid QEPs are listed for the user query (Q1) based on allocation schema.

```
Q1: SELECT a, m
FROM R1, R2, R3, R4, R5, R6, R7, R8,
WHERE R1.a=R4.t
AND R4.p=R2.x AND R1.a=R7.q
AND R2.x=R3.n AND R4.x=R5.s
AND R8.w=R6.d AND R7.j=R6.k
```

- queryPlan[0] = [1, 1, 2, 2, 3, 5, 5]
  R1 in site 1, R2 in site 1, R3 in site 2, R4 in site 2 and so on.

```
queryPlan = {1: [1, 1, 2, 2, 2, 3, 5, 3],
             2: [3, 5, 7, 15, 4, 6, 8],
             3: [6, 7, 8, 11, 16, 6, 8, 9],
             4: [10, 11, 11, 15, 16, 11, 16, 14],
             5: [8, 8, 10, 14, 16, 11, 11, 14],
             6: [15, 12, 13, 11, 15, 15, 11, 12],
             7: [1, 2, 5, 7, 2, 4, 6, 8],
             8: [3, 5, 7, 8, 15, 3, 5, 3],
             9: [2, 2, 2, 2, 3, 5, 3],
             10: [2, 1, 13, 2, 15, 3, 5, 3],
             11: [8, 8, 16, 14, 16, 15, 16, 14],
             12: [3, 7, 7, 8, 16, 7, 16, 3],
             13: [2, 2, 2, 2, 15, 16, 14],
             14: [1, 1, 8, 8, 2, 7, 8, 8],
             15: [8, 8, 8, 2, 7, 8, 9],
             16: [5, 16, 15, 15, 16, 6, 8, 9],
             17: [1, 1, 1, 1, 15, 15, 16, 14],
             18: [10, 11, 11, 8, 7, 3, 5, 3],
             19: [15, 16, 15, 15, 15, 16, 14],
             20: [1, 1, 8, 8, 1, 7, 8, 8] }
```

- Cost calculations

```
def queryAffinityCost(queryPlan):
    ret = []
    for i in range(1, Qeps + 1):
        QAC = 0
        sitesRequired = set(queryPlan[i])

        for relations in sitesRequired:
            Ki = queryPlan[i].count(relations)
            QAC = QAC + (Ki * (N - Ki)) / (N * N)
        QAC = round(QAC, 4)
        ret.append(QAC)
        # print(QAC)
    return ret
```

```
def queryLocalizationCost(QEP):
    total = sum(R)
    ret = []
    for i in range(1, len(QEP) + 1):
        ret.append(QEP_util(QEP[i]))
    return ret
```

```
def localProcessingCost(queryplan):
    total_ = []
    for i in range(1, len(queryplan) + 1):
        lpc = RLPC(queryplan[i]) + CLPC
        total_.append(round(lpc, 4))
    return total_
```

```
def RPC(relation):  # the relation number
    # selectivity of relation (Number of tuples participating in join or semi
join) is required
    total = sum(R)
    Sr_ = Sr[relation - 1]
    rpc = R[relation - 1] * Sr_ / total
    return rpc


def RLPC(qp):  # qp are the qep in query plan
    sum_ = 0
    for i in qp:
        currentsite = RSM[i - 1]  # array of qp in sites
        values = []
        # print('currentsite', currentsite)
        for j in range(len(currentsite)):
            if currentsite[j] == 1:
                values.append((RPC(j + 1)))
        sum_ = sum_ + max(values)
    return sum_
```

6.2 Teacher Learner Based Algorithm

- Inspiration: Knowledge Transfer in a classroom
- Required parameters: Population size and number of iterations

```
# Step 1: Fix the population size
population_size = no_of_QEPs

# Step 2: Fix the maximum number of iterations
max_iteration = 10
```

- Algorithm consists of two phases:
  Teacher's Phase:
  - New solution is generated using the best solution and mean of the population (class).
  - Greedy selection: Accept the new solution if better than the previous solution which is used to generate it else discard it.

  Learner's Phase
  - New solution is generated from the partner's solution (a randomly selected member from the class.
  - Greedy selection

- Each solution undergoes the teacher's phase followed by the learner's phase.

  Iteration 1: Student 1- T1 L1, Student 2- T2 L2, Student 3- T3 L3
  Iteration 2: Student 1- T1 L1, Student 2- T2 L2, Student 3- T3 L3

  .
  .
  .

  Iteration T: Student 1- T1 L1, Student 2- T2 L2, Student 3- T3 L3

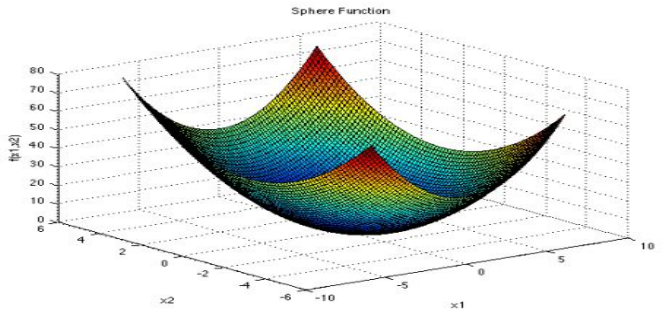
```
QAC: [0.7188, 0.7656, 0.8125, 0.75, 0.7812, 0.7188, 0.8438, 0.75, 0.5469, 0.8125,
0.7188, 0.7188, 0.5781, 0.6562, 0.5781, 0.8125, 0.6562, 0.8125, 0.5469, 0.5938]
QLC: [0.1931, 0.2187, 0.2246, 0.1944, 0.1931, 0.2187, 0.154, 0.2228, 0.1009, 0.2246,
0.1931, 0.1944, 0.1009, 0.1931, 0.1568, 0.154, 0.1069, 0.2228, 0.1902, 0.1224]
LPC: [0.1427, 0.1251, 0.1239, 0.1193, 0.1247, 0.1167, 0.1324, 0.1395, 0.1354, 0.1373,
0.1202, 0.1224, 0.1269, 0.1395, 0.1251, 0.1248, 0.1342, 0.1332, 0.1215, 0.1395]
```

- QAC, QLCand LPC calculated for each Query Execution Plans are given as decision variables to a fitness function.

- A fitness function is a particular type of objective function that is used to summarise, as a single figure of merit, how close a given design solution is to achieving the set aims. Fitness function used here is the Sphere function.

```
# Step 3: Find out the fitness value for each member in the population
def fitness_func(member):
    result = 0
    for m in range(len(member)):
        result += math.pow(member[m], 2)
    return result
```

- Sphere Function : The function is usually evaluated on the hypercube $x_i \in [-5.12, 5.12]$, for all i = 1, …, d.



$$f(\mathbf{x}) = \sum_{i=1}^{d} x_i^2$$

6.3 Teacher's Phase: Generation of a new solution

- New solution is generated with the help of the teacher and means of the solution.
- Teacher: Solution corresponding to the best fitness value. Since it is a minimization problem we consider the best solution as the solution with the least fitness value.

```
# Step 4: selecting the best learner or teacher from the population based on minimum
fitness value
def sel_teacher():
    print("fitness value of each cost row= ", np.apply_along_axis(fitness_func, 1,
costs_array))
    global fitness_array
    fitness_array = np.apply_along_axis(fitness_func, 1, costs_array)
    pos_teacher = np.where(fitness_array == np.amin(fitness_array))[0][0]
    print("position of teacher = ", pos_teacher)
    return pos_teacher
```

```
# Step 5: determine mean of each cost of the population
```

```
def mean_column_costs():
    means = np.array(costs_array).mean(axis=0)
    return means
```

- Each variable in a solution is modified as:

$$X_{new} = X + r(X_{best} - T_f X_{mean})$$

$X - Current\ Solution$
$X_{new} - New\ Solution$
$X_{best} - Teacher$
$X_{mean} - Mean\ of\ the\ population$
$T_f - Teaching\ factor,\ either\ 1\ or\ 2,\ same\ for\ all\ variables\ of\ a\ solution$
$r - Random\ number\ between\ 0\ and\ 1\ ,\ to\ be\ selected\ for\ each\ variable$

- Evaluate the fitness value of the new solution generated in the teacher's phase. Perform greedy selection to update the population.

$if\ f_{new} < f\ else\ remains\ the\ same$
$X = X_{new}$
$f = f_{new}$

```
# Step 6: Generation of new solution - Teacher phase of each student
def gen_new_soln():
    global costs_array
    global fitness_array
    X_best = np.array(costs_array[sel_teacher()])
    X_mean = mean_column_costs()

    Tf = round(random()) + 1  # Tf same for all variables of a solution
    for k in range(population_size):
        X = costs_array[k]
        r = np.random.rand(decision_var)  # r to be calculated for each variable in the
solution
        X_new = X + r * (X_best - (Tf * X_mean))

        # Step 7: Before calculating fitness value for x_new, checking whether values in
x_new are in between the lb
        # and ub condition of the fitness func
        X_new[(X_new < -5.12)] = -5.12
        X_new[(X_new > 5.12)] = 5.12
        # Step 8: Calculating the fitness value of the bounded solution
        fitness_x_new = fitness_func(X_new)
        # Step 9: Perform greedy selection to update the population
        fitness_teacher = fitness_func(X_best)
```

```
        if fitness_x_new < fitness_teacher:  # X_new is the best solution
            costs_array[k] = X_new
            fitness_array[k] = fitness_x_new
```

6.4 Learner's Phase: Generation of new solution

- New solution generated with the help of the partner solution, randomly selected solution from the population.

$$X_{new} = X + r(X - X_p) \; if \; f < f_p$$
$$X_{new} = X - r(X - X_p) \; if \; f \geq f_p$$

X – *Current Solution*
X_{new} – *New Solution*
X_p – *Partner Solution*
f – *Fitness of current solution*
f_{new} – *Fitness of partner solution*
r – *Random number between* 0 *and* 1 , *to be selected for each variable*

```
# Learners Phase
        xl = np.array(costs_array[k])
        while True:
            y = np.random.randint(population_size)
            if k != y: break
        partner = costs_array[y]
        fitness_partner = fitness_func(partner)
        rl = np.random.rand(decision_var)
        if fitness_array[k] < fitness_partner:
            Xlnew = xl + rl * (xl - partner)
        else:
            Xlnew = xl - rl * (xl - partner)
        Xlnew[(Xlnew < -5.12)] = -5.12
        Xlnew[(Xlnew > 5.12)] = 5.12
        fitness_learner = fitness_func(Xlnew)
        if fitness_learner < fitness_array[k]:
            costs_array[k] = Xlnew
            fitness_array[k] = fitness_learner
```

- The whole process is iterated for the maximum number of iterations.

7 Results and Analysis

From the table we can see that QEP 19 is the most optimized QP with the least cost.

Rank	QEP	QC Cost
1	19	1.69037e-07
2	15	2.40324e-07
3	7	4.10356e-07
4	20	4.43097e-07
5	11	4.62553e-07
6	8	4.99287e-07
7	12	8.225e-07
8	3	9.19931e-07
9	17	1.36816e-06
10	10	1.37384e-06
11	5	1.41171e-06
12	4	1.4165e-06
13	13	1.87671e-06
14	14	1.8939e-06
15	16	2.00152e-06
16	2	2.88056e-06
17	9	2.91388e-06
18	1	3.19192e-06
19	6	3.42605e-06
20	18	7.67976e-06

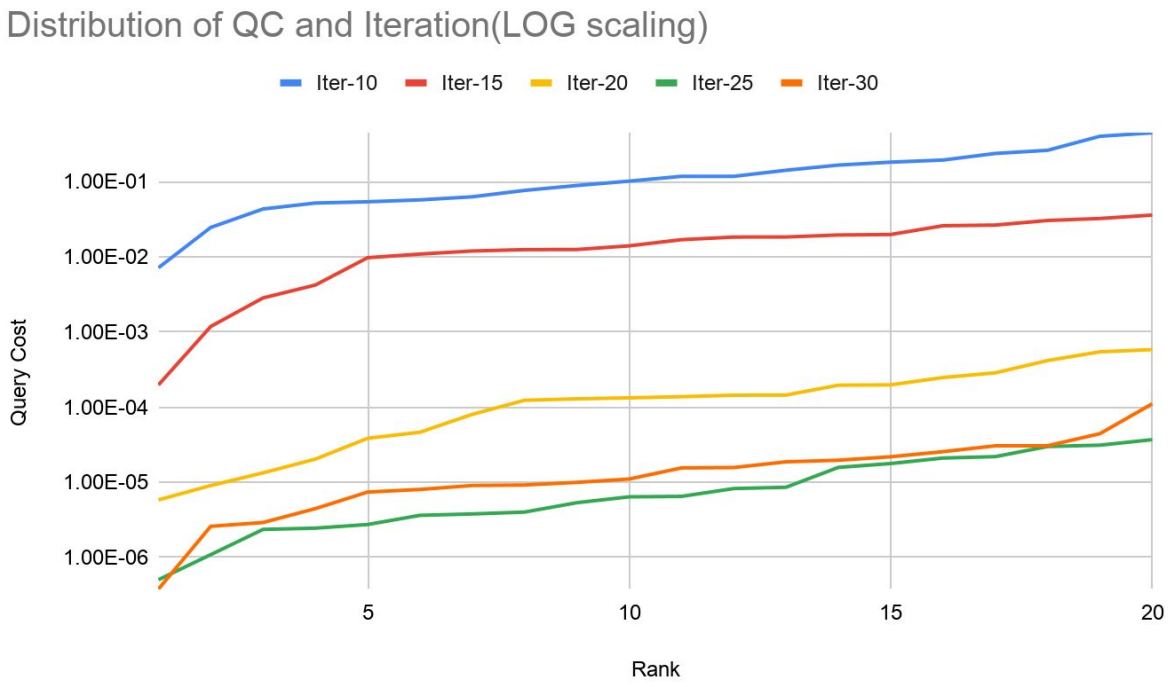


Fig Distribution of QC for various iteration

The Graph is an indication that the optimiser is working efficiently. As the number of iterations increases the query cost decreases. As the rank increases the cost of query increases.

8 Conclusion

In distributed database system data is dispersed over the multiple sites, this distribution of data is based on partition or replication based due to which a given relation can be found in more than one sites. Query processing in such an environment is a difficult task for query processors, as multiple equivalent alternatives. Query processing in such an environment, major objective design are CPU, I/O and the site-to-site communication cost, among these, the site-to-site communication cost is the dominant cost. This requires an optimization mechanism to generate an optimal set of query plans to retrieve results for user query. The results show that the TLBO algorithm is effective in reducing the cost and it produces the optimal query plan with less number of parameter tuning.

9 References

[1] Stefano Ceri, Giuseppe Pelagatti. Distributed Databases principles and systems, McGraw-Hill Inc., 1984

[2] Tamer Ozsu M, Patrick Valduriez. Principles of Distributed Database System, Prentice-Hall, Inc.,1999

[3] Singh Vikram, Mishra Vikash. Distributed Query Plan generation using Aggregation based Multi-Objective Genetic Algorithm. In ACM ICTCS Proceedings, pages 1-8, November 2014.

[4] Hongbin Dong, Yiwen Liang. Genetic algorithms for large join query optimization, In 9th ICGEC Proceedings, pages 07-11, 2007.

[5] Chengwen Liu, Hao Chen, Warren Krueger, A Distributed Query Processing Strategy Using Placement Dependency, In IEEE 12th ICDE Proceedings, pages 477-484, February 1996.

[6] Tamer Ozsu M, Patrick Valduriez. Principles of Distributed Database System, Prentice-Hall, Inc.,1999.

[7] Rao R V, Patel V. An elitist teaching–learning-based optimization algorithm for solving complex constrained optimization problems, Int. J. Ind. Eng. Computation, 3 (4):535–560, 2012

[8] Niknam T, Fard A K, Baziar A. Multi-objective stochastic distribution feeder reconfiguration problem considering hydrogen and thermal energy production by fuel cell power plants, Energy 2012, p.563–573.

[9] Vijay Kumar T V, Singh V, Verma, A K. Distributed query processing plans generation using Genetic Algorithm. International Journal of Computer Theory and Engineering, 9(31): 38-45, 2011 19.

[10] Singh Vikram, Mishra Vikash. Distributed Query Plan generation using Aggregation based Multi-Objective Genetic Algorithm. In ACM ICTCS Proceedings, pages 1-8, November 2014.

[11] Vijay Kumar T V, Singh V, Verma, A K. Generating Distributed query plans Using Genetic Algorithm, In IEEE ICDSDE Proceedings, pages 173-177, March 2010 28. Vijay Kumar T V, Panikar Shina.

[12] Generating Query Plans for Distributed Query Processing Using Genetic Algorithm, In Springer ICICA Proceedings, pages 765-772, October 2011. 29.

[13] Mishra Vikash, Singh Vikram. Generating Optimal Query Plans for Distributed Query Processing using Teacher-Learner Based Optimization, In 11th ELSEVIR ICDMW Proceedings, pages 181-290, August 2014.