Adaptive Optimization for Continuous Multi-Way Joins Using ACO System

G. Sakthivel
*PG Department of Computer Science,
Arignar Anna College (Arts & Science),
Krishnagiri, India*

P. Madhubala
*PG & Research
Department of Computer Science,
Don Bosco College, Dharmapuri, India*

Abstract- **The join operator is a core component of an ACO System. Query optimization, the process to generate an optimal execution plan for the posed query, is more challenging in such systems due to the huge search space of alternative plans incurred by distribution. . Due to the constantly updating nature of continuous queries, the query optimizer has to frequently change the optimal execution plan for a query. However, optimizing the join executing plan for every execution step might be prohibitively expensive; hence, dynamic optimization of continuous join operations is still a challenging problem so far. Therefore, this paper proposes the first adaptive optimization approach towards this problem in the ACO system. The approach comes with two dynamic cost-based optimization algorithms which use a light-weight process to search for the best execution plan for every execution step. The experimental results show that the proposed algorithm saves up to about 100% of optimization time with no significant difference in the quality of generated plans compared with the best existing genetic-based algorithm.**

Keywords - Ant Colony Optimization, Adaptive Optimization, Query Optimization, Continuous Multi-way joins, Join queries, Cost model.

1. INTRODUCTION

The query optimization might be applied at the logical level by rewriting the plan to improve efficiency, called algebraic optimization. The common rewriting rules such as reordering selection before joins and evaluating inexpensive predicates before complex ones were used in the Particularly for continuous queries, (Arvind Arasu, 2006) proposed rules on window-based operators such as commutative rules on time-based and count-based windows. Optimization by rescheduling physical query plans are similar to those used in relational databases, e.g., re-ordering a sequence of binary joins in order to minimize a particular cost metric. There has been work in join ordering for ACO System model (Shivnath Babu K. M., 2005), (Lukasz Golab, 2008). Furthermore, adaptive re-ordering of pipelined stream filters is studied in (Shivnath Babu R. M., 2004) and adaptive materialization of intermediate join results are considered in (Shivnath Babu R. M., 2004). Query optimization in different environment from single processor relational database systems to grid systems is discussed (H A. M., 2010) Hemeurlain 2009. Queries can be categorized according to (i) the type of operations which includes retrieve and update queries (ii)the number tuples retrieved ,which includes range and singleton queries (iii) the type of operator used which includes sets, aggregate and join queries (iv) the application type , which includes Online Transaction Processing (OLTP) and Decision Support System (DSS) queries.
The multi query optimization which tries to find same patterns in submitted queries and obtain a global plan for their optimization (Y.E, 1996)loannidis and rule based optimization which optimizes queries on set of rules ranked in the order of efficiency (C.E, 2005). The problem of optimizing multiple join queries has been proved to be computationally intractable with a large number of relations (T, 1984) ibaraki etal. 1984. In this Paper, an ant colony optimization (ACO) algorithm is introduced for the first time to tackle the problem of distributed join query optimization in a search space where relations are replicated and not fragmented. Ant algorithm had been previously applied for designing distributed database system (S.M.T, 2011), karimi adl et al.2009. Furthermore, Ant and bee colony algorithms have been applied for join query optimization in centralized database System which has led to better results compared to genetic algorithm (ALAA ALJANABY, 2005), (Luasz, 2008). Two cost models are introduced one based on the total time and the other on the response time. Our cost models consider both communication as well as local processing costs. We have compared our algorithm to two other genetic algorithms previously proposed in Rho et al.1997; sevinc et al.2010, the results of which show that our proposed algorithm could find reasonable QEPs with more Optimization problem.
The remainder of this paper is outlined as follows. First Section 2 covers related previous work. Section 3 explain more general concept of ACO Algorithm. The search strategy of our proposed ACO algorithm for distributed join query optimization for Section 4. The Section 5 will follow up with two adaptive optimization algorithms. The experiments of such algorithms are reported in the following Section 6. Finally, we conclude the paper in Section 7.

ISSN: 2349 – 6363

2. RELATED WORK

Algorithm used in the first distributed database system such as R*, SDD-1 and distributed ingress are widely studied and compared in (ozsu ,2011). Semi joins (Bernstein ,1981) are applied to reduce one of the relations involved in join through sending join attributes of one of the relations to the site of the other so as to extract matching tuples. Two way semi - joins (2SJ) Roussopoulous et al 1991 are the extension of semi joins to reduce both relations involved in a join. In 1997, Ribero and his colleagues proposed an algorithm with the following characteristics (Ribeiro, 1997). Its search space contains both bushy and linear execution plans. Unlike other algorithms that usually avoid unnecessary Cartesian products. Ribero et al. argue that Cartesian products may be worth doing in distributed environments, in case the relations are located at the same site. Their proposed algorithm does not benefit from semi joins and does the not benefit from semi joins and does the optimization in an environment where no relations are replicated. The cost model is based on the response time in terms of pipelining of join operations. The relevant original ideas came from cost models which are used for parallel database lanzelotte et al 1994, though may not be appropriate to be applied in distributed databases. The search strategy is based on tabu search metaheuristic algorithm and it uses a hashing based data structure for tabu search memory. Therefore, they are not comparable with our algorithm which its optimization model integrates all these decisions. In contrast to algorithms, we use two cost models- one based on the response time.

3. GENERAL CONCEPTS OF ACO ALGORITHM

ACO algorithms are a Series of ant algorithms inspired by foraging behavior of ant colonies. To solve a problem with ACO algorithms, first the problem should be abstracted as a graph. For example the travelling salesman problem (TSP) can be represented by agraph G= (N,E) where a fixed set of vertices N represents the cities and a fixed set of edges E Shows the connection between the cities the objectives is to find hamiltonian path for G which gives the minimal length Dorigo et al. 2004. Ants build their solutions by moving on the problem graph and laying pheromone so as to guide other ants. Pheromone trails, which provides a positive feedback mechanism, permit ants to cooperate and exploit each other‟s experiences. A negative feedback mechanism. The first ACO algorithm, Ant System (AS), was developed by Professor Dorigo in 1992 (Dewitt,1985) (Dorigo, 1992). This algorithm was introduced using the TSP as an example application. AS achieved encouraging initial results, but was found to be inferior to state-of-the-art algorithms for the TSP. The importance of AS therefore mainly lies in the inspiration it provided for a number of extensions that significantly improved performance and are currently among the most successful ACO algorithms. In fact most of these extensions are direct extensions of AS in the sense that they keep the same solution construction procedure as well as the same pheromone evaporation procedure. These extensions include elitist AS, rank-based AS, and MAX − MIN AS. The main differences between AS and these extensions are the way the pheromone update is performed, as well as some additional details in the management of the pheromone trails. A few other ACO algorithms that more substantially modify the features of AS were also developed; those algorithms are the Ant Colony System (ACS), the Approximate Nondeterministic Tree Search and the Hyper-Cube Framework for ACO. Only the ACS will be briefly presented; for the others, we invite the reader to consult the reference Dorigo, M., Stutzle ̈ T. (2004). ACO algorithms are a Series of ant algorithms inspired by foraging behavior of ant colonies. To solve a problem with ACO algorithms, first the problem should be abstracted as a graph. For example the travelling salesman problem (TSP) can be represented by a graph G= (N,E) where a fixed set of vertices N represents the cities and a fixed set of edges E Shows the connection between the cities the objectives is to find hamiltonian path for G which gives the minimal length (Dorigo et al. 2004). Ants build their solutions by moving on the problem graph and laying pheromone so as to guide other ants. Pheromone trails, which provides a positive feedback mechanism, permit ants to cooperate and exploit each other‟s experiences. A negative feedback mechanism is also necessary to avoid stagnation (i.e premature convergence) which is implemented through pheromone evaporation. This indirect communication mediated by the environment is called stigmergy . Pheromone deposition and evaporation are two phases of Pheromone update that happen in the ACO. After all ants have completed the tour construction, the pheromone trails are updated. This is done first by lowering the pheromone trails by a constant factor (evaporation) and then by allowing the ants to deposit pheromone on the arcs they have visited. In particular, the update follows this rule.

$$T_{ij}(t+1) = \rho . T_{ij}(t) + \sum_{k=1}^{m} \Delta T_{ij}^{k}(t) \tag{1}$$

Where the parameter p (with 0≤p≤1)) is the trail persistence (thus, 1-p models the evaporation) and $T_{ij}^{k}(t)$ is the amount of pheromone ant k puts on the arcs it has used in its tour. The evaporation mechanism helps to avoid unlimited accumulation of the pheromone trails. While an arc is not chosen by the ants, its associated pheromone trail decreases exponentially; this enables the algorithm to "forget" bad choices over time. Research on ACO has shown that improved performance may be obtained by a stronger exploitation of the best solutions found during the search and the search space analysis in the previous section gives an explanation of this fact. Yet, using a greedier search potentially aggravates the problem of premature stagnation of the search. Therefore, the key to achieve best performance of ACO algorithms is to combine an improved exploitation of the best solutions found during the search with an effective mechanism for avoiding early search stagnation. MAX – MIN Ant System, which has been specifically developed to meet these requirements, differs in three key aspects from AS. To exploit the best solutions found during iteration or during the run of the algorithm, after each iteration only one single ant adds pheromone. (i) This ant may be the one which found the best solution in the current iteration (iteration-best ant) or the one which found the best solution from the beginning of the trial (global-best ant). (ii) To avoid stagnation of the search the range of possible pheromone trails on each solution component is limited to an interval [Tmin ; Tmax]. (iii) Additionally, we deliberately initialize the pheromone trails to max, achieving in this way a higher exploration of solutions at the start of the algorithm. In the next sections we discuss the differences between MMAS and AS in more detail and report computational results which demonstrate the effectiveness of the introduced modifications in improving the performance of the algorithm.

4. QUERY OPTIMIZATION

Query optimization is used for accessing the database in an efficient manner. It is an art of obtaining desired information in a predictable, reliable and timely manner. Formally defines query optimization as a process of transforming a query into an equivalent form which can be evaluated more efficiently. The essence of query optimization is to find an execution plan that minimizes time needed to evaluate a query. To achieve this optimization goal, we need to accomplish two main tasks. First one is to find out the best plan and the second one is to reduce the time involved in executing the query plan.

Kunal Jamsutkar et al. 2013 states a query passes through three different phases during the query processing in DBMS which are as follows:

- Parsing and translation
- Optimization
- Evaluation

Usually, user queries are submitted to DBMS as SQL queries. During the parsing and translation phase, the given query is translated into its internal form. In generating the internal form of the query, the parser checks the syntax of the user's query, verifies that the relation names appearing in the query are names of the relations in the database and so on. The system constructs a parse tree representation of the query, which it then translates into a relational algebra expression. For example let us consider the following SQL query:-

Select Sno from Student where Sno='101'

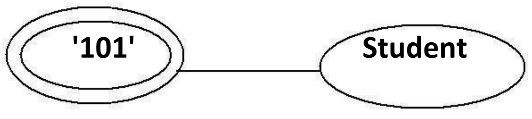
This query is then translated into either of the following relational algebra expressions as follows:-

σSno = '101' (πSno (Student))

πSno (σSno='101' (Student))

After parsing and translation into relational algebra expression, the query is then transformed into a form which is usually query tree or graph that can be handled by the optimization engine. For the above example, the relational algebra expression can be represented as either query tree or query graph which is shown in Figure 1.

σSno='101'	**πSno**
πSno	**σ Sno='101'**
……………	**……………..**
Student	**Student**



a) Query tree **b) Query graph**

Figure 1. Query representation

During the optimization phase, the optimization engine performs various analyses on the query data. It applies various rules to the internal data structures of the query to transform these structures into equivalent and efficient representation. It then generates valid evaluation plans based upon the rules applied. From the generated evaluation plans, the best evaluation plan to be executed is determined and passed onto the query execution engine.The final phase in processing a query is the evaluation phase. During the evaluation phase, the best evaluation plan generated by the optimization engine is selected and then executed. Avi Silbershatz et al. 2002 describes the steps involved in query processing are shown in Figure 2.

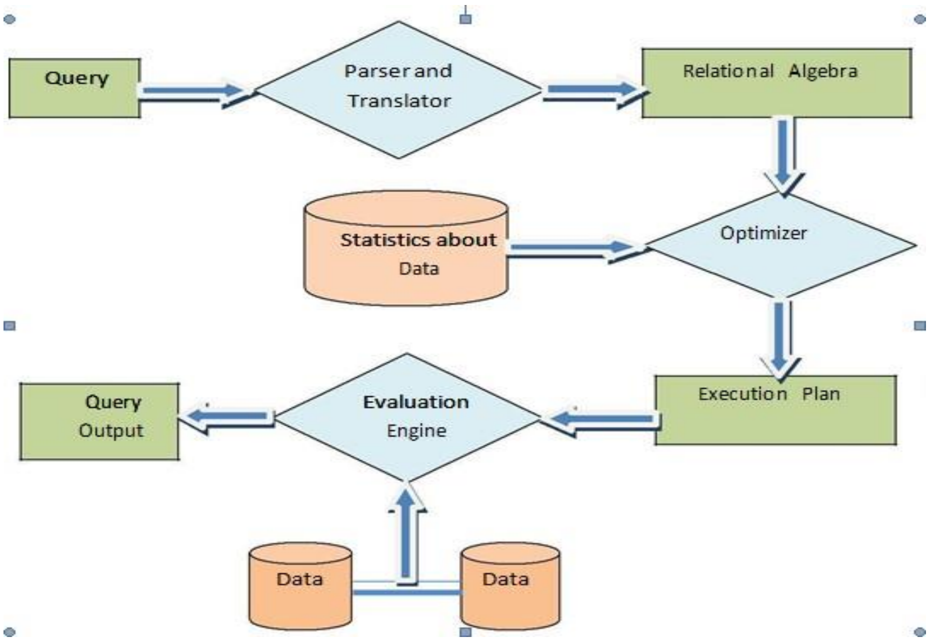


Figure 2. Steps in query processing

Deepak Sukheja and Umesh Kumar Singh 2011 states query processing and optimization process work together to execute any kind of queries. Query processing is concerned with execution of a query or refers to the activities involved in extracting data from a data warehouse. On the other hand, query optimization process deals with the efficiency of the query. It defines the execution plans, the strategy of execution of the query and chooses the best execution plan.
(Kosmann, 2000) states the query optimization is the process of identifying an efficient way in which the query could be executed with less time complexity to produce better results. In this process, when a query in a high level language is first submitted, it is first scanned and parsed to determine if the query consists of appropriate syntax. If the query passes the parsing checks for correct syntax, then it is converted into query tree or query graph. Here we determine different ways of representing a query which are then passed on to the query optimizer. The way by which a given query is optimized plays a vital role in the enhancement of query performance.
(Sunitha Mahajan and Vaishali Jadhav 2012) states that the goal of the query optimizer is to find a reasonably efficient strategy for executing the query. It considers the possible query plans for a given input query and determines the most efficient query plan. Once the query optimizer has determined the execution plan, the code generator writes out the actual access routines to be executed. With an interactive session, the query code is interpreted and passed directly to the runtime database processor for execution. During this section, we estimate the various cost factors for executing each of the execution plans. The execution plan which results in least cost estimate is chosen as best optimal execution plans.
(Alaa Aljanaby, 2005) in the query optimization process, user given query is first scanned, parsed and validated. The scanner identifies the language tokens such as SQL keywords, attribute names and relation names in the text of the query, whereas the parser checks the query syntax to determine whether it is formulated according to the syntax rules of the query language. An internal representation of the query is then created. A query expressed in relational algebra is usually called initial algebraic query and can be represented as a tree data structure called query tree. It presents the input relations of the query as leaf nodes of the tree and represents the relational algebra operations as internal nodes.
The next step is an optimization step that transforms the initial algebraic query using relational algebra transformation into other algebraic queries until the best one is found. A query execution plan is then founded

which represented as a query tree includes information about the access method available for each relation as well as the algorithms used in computing the relational operations in the tree. The next step is called code generator, where we generate code for the selected query execution plan. This code is then executed by the run time database processor to produce the query result. The run time database processor has the task of running the query code, whether in compiled or interpreted mode, to produce the query result. If a run time error results, an error message is generated by the run time database processor. Figure 3 shows the different steps of query processing**.**

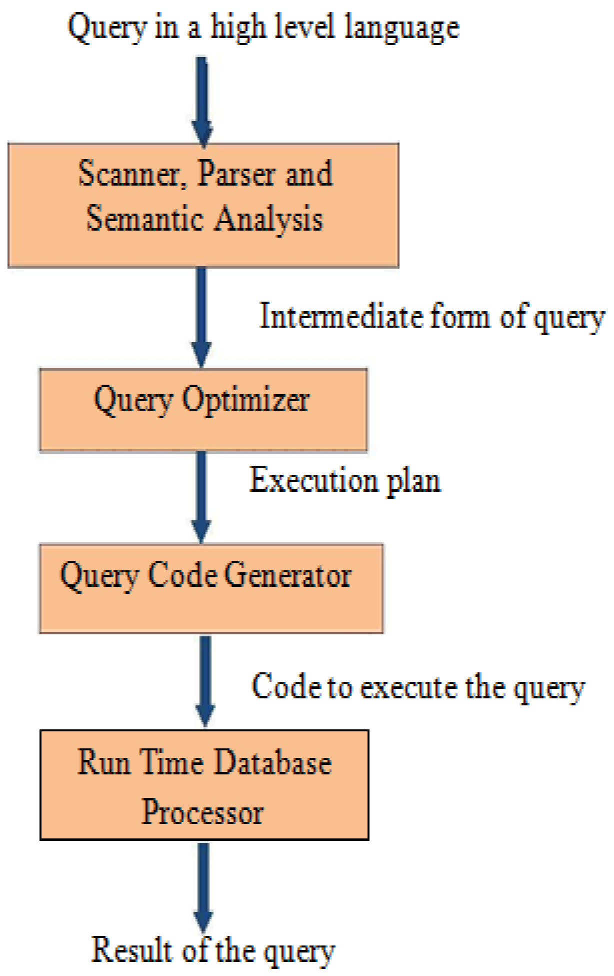

Query in a high level language
Scanner, Parser and Semantic Analysis
Intermediate form of query
Query Optimizer
Execution plan
Query Code Generator
Code to execute the query
Run Time Database Processor
Result of the query

Figure 3. Different steps of query processing**.**

Query optimization process becomes a complex task as query complexity increases with new applications. Significant research work has been done in developing efficient query optimization techniques for processing complex queries in a cost effective manner.

```
Input: Let the input be Dependency Rule Set DRS
Output: Let the output be Final resultant SEquence selected SEq.
Step1: Initialize Query Completion Probability Set QCPS.
Step2: For each rule Ri from DRS
Compute Query completion probability Qcp
                    $Q_{cp} = N *^{\log(R_i(N_i)) + (R_i(N_L)) * \log(R_i(NDTR))}$
          QCPS = ∑QCPS(i)+Qcp
End.
Step3: Choose the most probable Rule.
Step4: Return selected Dependency Rule Set DRS(i).
Step5: Stop.
```

4.1. Optimization of Multi-Way Join

Intra-join algorithm can be achieved by assigning more than one processor to a join operation as in all proposed parallel join algorithms DeWi85, KitsgO, Schn89, Vald84,Lu90, wolfYO.

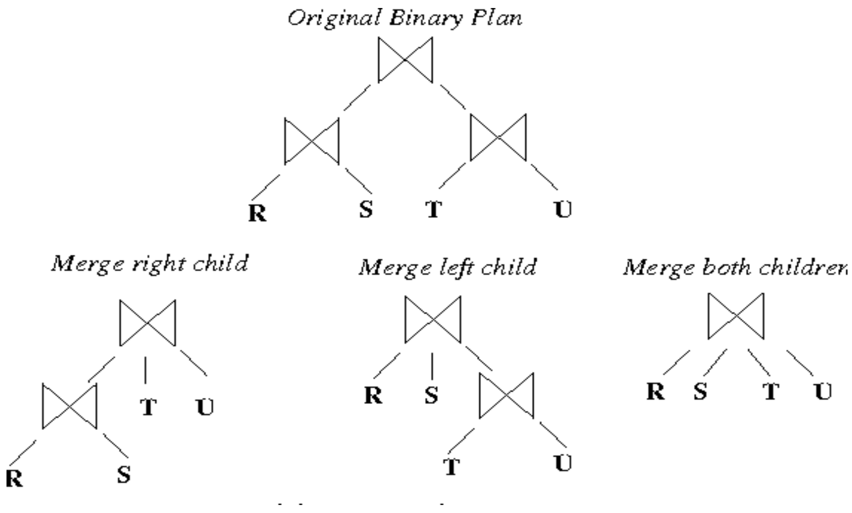


Figure 4. Building a Multi-Way Join via Merging

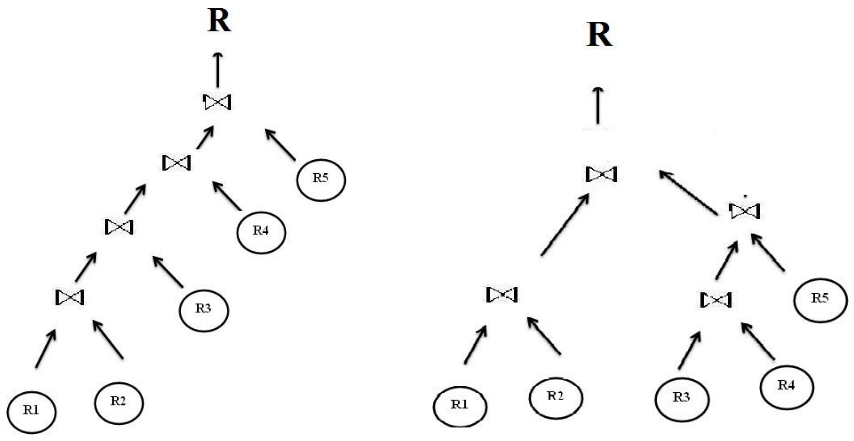


Figure 5. (a)Linear QEPs (b)Synchronized Bushy QEP

Inter-join parallelism among multi-way join queries can be realized by generating query execution plans with bushy structure. The difference between such bushy structured QEPs and the linear structured QEPs is shown in Figure 4. In a linear QEP Figure 5(a). Joins in a multi-way join query are performed one by one. The result relation from the first join of two relations, say R1 and R2, is joined with the third relation, R3 the result of which is then joined with the fourth relation, R4 and so on. In a bushy structured QEP, a number of pairs of relations may be joined in parallel. In Figure 5 (b), two pairs of relations, (R1t, R 2) and (R 3, R4), are joined in parallel. The result of R3 $\bowtie$ R4 is then joined with R5, the result of which is again joined with the result of R 1 $\bowtie$ R2 When the bushy structured QEPs are included in the search space of a query optimizer, the number of feasible QEPs increases dramatically. To limit the increase of QEPs in the search space of our multiprocessor query optimizer, we divide QEPs into two groups, synchronized and asynchronized. By a synchronized QEP, we mean that the whole multi-way join process is divided into synchronized steps. For each step, a number of joins are executed concurrently. The joins to be performed at the following step will not start to execute until all joins in the previous step have been completed. In this section, we are going to propose a greedy multi-way join optimization algorithm which explores inter-join parallelism by considering such synchronized QEPs during optimization. By limiting QEPs to synchronized ones, the cost estimation of a QEP is easier. However, there are two possible side effects: (1) the possible pipeline among steps is not taken into account. Instead, the costs of storing and retrieving the intermediate results are added to the plan cost, and (2) some processors that complete one join earlier than others have to wait and the CPU utilization will decrease. As a result, some better plans may be excluded from the search space. However, the second effect could be minimized by carefully allocating processors to the joins to be concurrently executed according to their workload. Furthermore, since linear QEPs are still in the search space, the new optimization algorithm should be at least as good as those that do not consider bushy QEPs.

4.2. Algorithm GP: a greedy multi-way join query optimization algorithm

The Greedy Parallel multi-way join optimization algorithm, GP is an iterative algorithm that generates one step in a synchronized QEP during each iteration. It is a greedy algorithm since it always tries to join as many pairs of relations as possible in parallel for the current step. At the beginning, all relations to be joined are included in the working set T. A set of relation pairs, R , is selected for the first step by calling function Select_ Rel_Pairs. For subsequent steps i, the same procedure is applied to the reduced working set that consists of the intermediate relations from the last step, step i - 1, and the relations that have not been joined so far. Graphically, this reduced working set is represented by a reduced join RTU@I that is obtained by replacing the relations joined in step i - 1 by their result relations and merging the edges accordingly. When the working set contains less than four relations, function Two_way_seq is called to determine the sequence of sequentially joining those relations.

4.3. Selecting pairs of relations

Function Select_rel_pairs in Algorithm GP select k pairs of relations from the working set to be joined in parallel for the current step. Select_rel_pairs determines concurrently executed relation pairs with given working set (or join graph). The algorithm uses an iterative approach starting with k = 1. During each iteration, it computes the costs of QEPs which concurrently join k pairs of relation at the first step and find the minimum cost by calling function Minimum_cost. It terminates when either k is equal to the number pairs in the join graph or such k is found that the minimum cost of QEPs concurrently joining ktl pairs first is greater than the minimum cost of QEPs having k joins evaluated concurrently first.

Algorithm GP

```
Input :.A join graph G = (T, E)  relations and edge set E represents the join conditions.
Output : S , the join sequence consisting of relation pairs S <--- ø;
while Size(T) > 3 do
  {
    R t Select_rel_pairs (G); S<--- R;
    S<---S∪ R
   G <---G with each pair of relations in R replaced by their join results;
  }
  R <---Two_way_seq (G); S<---S∪ R
```

Algorithm for Multi-way join optimization algorithm GP

Algorithm Select-rel_pairs

```
Input : G , a join graph
Output : R , a set of relation pairs to be joined concurrently begin k <---0
repeat
k<---k+l;
CK<--- t Minimum-cost (G, k, RK);
if (R, does not contain all relations in G) then Ck+1 t Minimum-cost (G, k+1, RK+1);
until Ck+1>CK or RK+1, contains all pairs in G if Ck+1>CK then
return Rk else
return RK+1, end;
```

Function Minimum-cost is the core part of the algorithm. It takes the reduced join graph G, and the number of relation pairs to be joined concurrently first, k , as input and returns the minimum cost of those plans that joins k pairs first. At the same time, it determines those k pairs of relations and join methods for each pair of relations. The computation complexity of this function comes from the large number of feasible QEPs that join k pairs in parallel during the first step; and (2) a large number of combinations of join methods supported and possible processor allocation strategies for a chosen QEP. To simplify the cost evaluation of QEPs and hence to reduce the optimization overhead, we propose two heuristic cost functions that lead to two versions of Algorithm GP: GP_T, an optimization algorithm based on total Cost and GP_P, an optimization algorithm based on partial COSL As the name implies, algorithm GP_T, estimates the total cost of a QEP, C_{plan}, each step i $(1 \leq I \leq m)$, the m-step QEP. On the other I. which is the sum of the cost of hand, Algorithm GP_p uses only the cost of the first step (may plus one more join as explained later) Cost i as the approximation of Cost. We discuss the details of these two algorithms in the next two subsections.

5. PERFORMANCE STUDY

To evaluate the algorithms described in the above section, an experimental study is conducted with the following purposes: (1) to compare the four criteria for selection of pairs of relations (used in the algorithm of pairing relations) md (2) to evaluate and compare the effectiveness of algorithm GP with both heuristics, GP, and GPP in generating optimal plans. The optimization algorithm, algorithm GP, is implemented in our study. However, the input queries are randomly generated according to chosen parameters and execution costs of generated QEPs are calculated according to the developed cost models. Therefore, the results presented here are basically simulation results since no multiprocessor database system is available in our organization yet. We hope that these results can give us some insight into our algorithm and provide us with some experience to implement it in real systems. Though recent work [Kris86, Swam88, Swam891 have emphasized on large number of joins, we believe that for most traditional applications in a well-designed relational database system, most of the queries will require only a small number of joins. Therefore, we study the proposed algorithm on a small number of joins (S 10). We vary the join selectivity, the sizes of the relations, the number of processors and the number of tuples per page. However, our algorithm is also applicable for large number of joins (> 10). We define the following measure to study the performance of our algorithm: cost,, Cost-Multiplier (A1, A2) = lost A; where Cost,,,, (i = 1,2), represents the cost of executing the QEP generated by algorithm A1. Cost-Multiplier (A1, A3) is thus a measure of the relative performance of algorithm A1 over algorithm A2. For the experiments with small number of joins, we are able to compute the optimal solution by enumerating all possible combinations. We therefore use the optimal solution generated by exhaustively trying all possibilities as our basis for comparison. Hence, we have Cost-Multiplier (GP, OPT) = costfip =costGP/OPT where CostOPT and CostGp are the costs of plans generated by the exhaustive search and the algorithm GP used respectively. It is clear that Cost-Multiplier (GP, OPT) and a lower bound value of one implies that algorithm GP generates the optimal answer.

6. EXPERIMENT

Criteria for selecting the joining pairs In this experiment, we conducted several tests to study the criteria used for selection of joining pairs (see Section 3.3). The main parameters of the queries used in the experiment are shown in Table 1.

Table 1. Queries used in the experiment

Parameters		**Relation size (in pages)**	
		750 – 850	**600 – 1000**
Join Sel	0.0008 – 0.002	Test 1	Test 3
	0.0007 – 0.004	Test 2	Test 4

For example, in test 1, the join selectivity is varied from 0.0008 to 0.002 while the relations sizes are in the range of 750 to 850 pages. These are varied according to the uniform distribution such that the final relation size is also in the range of 750 to 850 pages. The other tests are similar except for the parameter settings. The numbers of processors are varied from 5 to 32 for the tests. For each test, more than 2500 multi-way join queries with different number of joins, relation sizes, join selectivity and numbers of processors are generated. A query generator is used to generate queries. The QEPs of these queries are generated by applying algorithm GP with all the four criteria. The average costs by using different criteria were compared with that of criterion 1 and Table 2 summarize the results. Those numbers greater than one means that the criterion protons worse than the first criterion.

Table 2. Comparison of criteria (A) Performance for GPT

Experiment Set	**GP_T**			
	C1	**C2**	**C3**	**C4**
Test 1	1.0000	1.0078	0.9912	0.9899
Test 2	1.0000	1.0000	0.9872	0.9872
Test 3	1.0000	1.0037	1.0023	0.9958
Test 4	1.0000	0.9993	0.9782	0.9734

Table 3. Comparison of criteria (B) Performance for GP_P

Experiment Set	**GP_p**			
	C1	**C2**	**C3**	**C4**
Test 1	1.0000	1.0004	0.9976	1.0003
Test 2	1.0000	1.0003	0.9963	0.9963
Test 3	1.0000	1.0024	1.0005	0.9943
Test 4	1.0000	1.0012	0.9919	0.9922

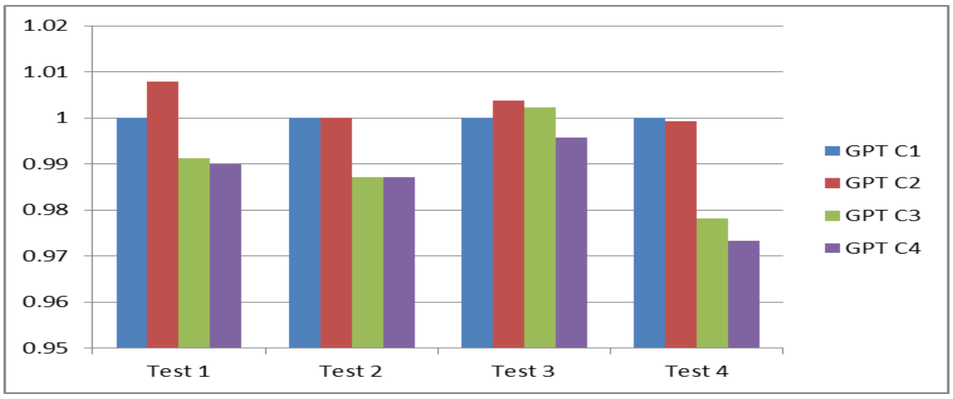


Figure 6. Representations of criteria (A) Performance for GPT

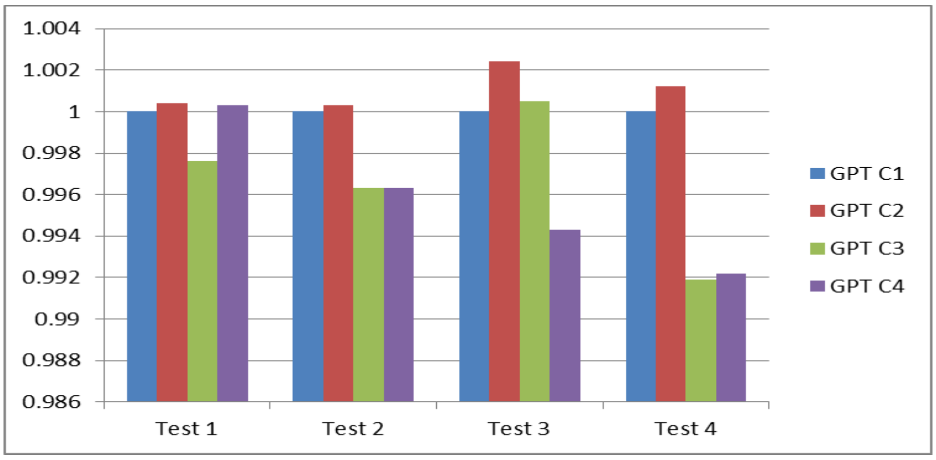


Figure 7. Representations of criteria (B) Performance for GP_P

Increase the number of joins From experiments 2 to 4, we see the effectiveness of the proposed algorithm GP. The purpose of this experiment is to see the relative performance of GP, and GPp for large number of joins. Since an exhaustive enumeration of the join orderings is computationally expensive, we compare them with one another. We vary the join selectivity's and the sizes of the relations. Tables 2 and 3 show the relative performance of heuristic CPT over GP,, with parameter settings from experiments 2 and 3 respectively. Figure 5 and 6 shows the representations of criteria (A) Performance for GPT and criteria (B) Performance for GP_P.

Table 4. Cost Multiplier (GPT, GPP)

Query type	COST MULTIPLIER (%)			
	0.5-0.7	0.7-0.8	0.8-0.9	0.9-1.0
10 –R	1.4.6	21.8	70.8	1.6
20 –R	7.65	25.75	63.25	1.15
30 –R	4.5	25.0	66.0	1.4
40-R	3.0	37.25	55.5	1.15
50-R	1.75	32.5	62.0	0.50

Table 5. Cost Multiplier (GPT, GPP)

Query type	COST MULTIPLIER (%)			
	0.5-0.7	0.7-0.8	0.8-0.9	0.9-1.0
10 –R	4.6	16.2	74.4	0.8
20 –R	3.75	27.75	64.00	0.5
30 –R	4.5	29.5	61.75	0.25
40-R	2.0	29.00	64.75	0.25
50-R	3.75	36.25	56.25	0.50

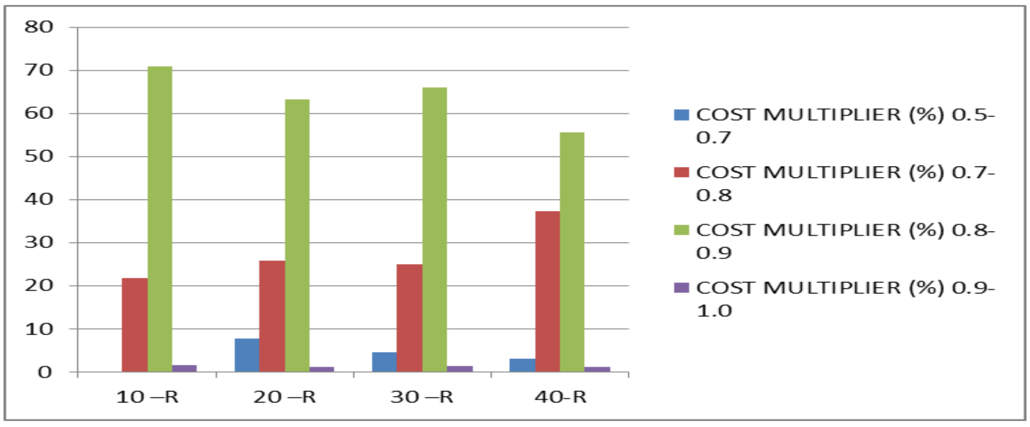


Figure 8(a). Representations of Cost Multiplier (GPT, GPP)

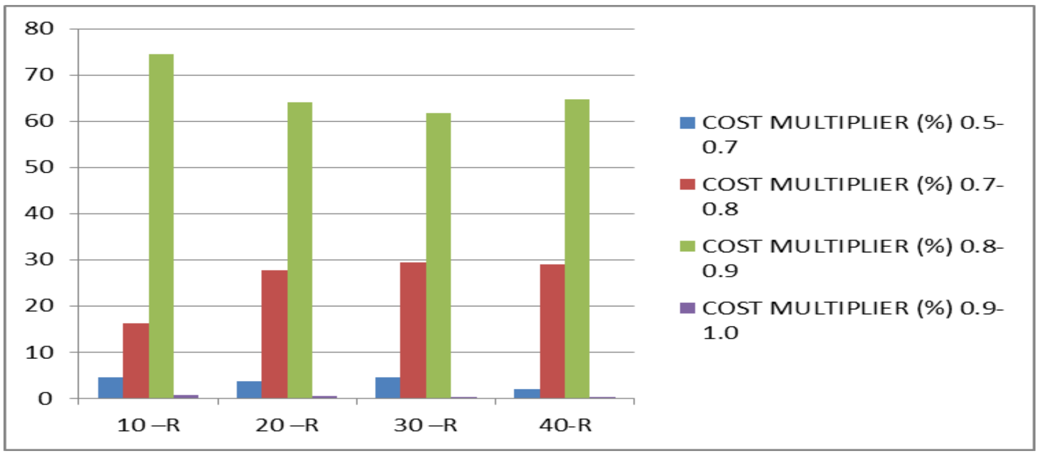


Figure 8(b). Cost Multiplier (GPT, GPP)

Tables 4 and 5 discusses the GPT outperforms GPP most of the time (> 98%). For more than 50% of the time, GP, produces results that are close to GP,. Up to 90% of the results generated by GP, are 80%-near-GPT. Figure 8(a), 8(b) shows the representations of Cost Multiplier (GPT, GPP).

7. CONCLUSIONS

The paper presents our effort on addressing a very challenging problem of dynamic optimization of join operation in continuous query ACO system which we believe is a very central performance problem of RSP engines. We modeled the join operations on ACO a continuous multi- way join which help us to introduce a general recursive cost model. Via this cost model, we propose two approximation versions which help us to introduce two light-weight adaptive optimization algorithms. While traditional optimizers (which do not generate parallel plans) deal with choosing an appropriate join method and the best join ordering, our optimizer that generates parallel plans, must also select the pairs of relations to be joined in parallel and allocate processors to the join operations. We proposed an algorithm, algorithm GP, which employs the greedy paradigm to generate parallel QEPs for multi-way join queries. The plan generated exploits parallelism at two levels: intra-join parallelism where several processors may be assigned to a join operation and inter-join parallelism where several joins may be performed concurrently. Our experimental results on simulated data show very encourage performance gain in controlled settings. This opens up interesting and potential options for implementing dynamic query optimizers for ACO System.

REFERENCES

A, D. M. (1996). System optimization by a colony of cooperating agents systems,Man, and cybernetics . *IEEE Transactions*, (pp. 29-41).

A, H. (2009). evolution of query optimization methods from centralized database system to data grid systems. *Proceedings of the 20th International Conference on Database and expert system Applications Linz,*.

Alaa Aljanaby, E. A. (2005). A Survey of Distributed Query Optimization. *International Arab Journal of Information Technology*, *2*(1), 48-57.

Arvind Arasu, S. B. (2006). The CQL continuous query language: semantic foundations and query execution. *The VLDB Journal* , 121– 142.

Avi Silbershatz, H. K. (2002). Database System Concepts. *McGraw-Hill*.

C.E, C. T. (2005). Database Systems. A practical Approach to design Implementation and Management. *Addison – Wesley*.

Dewitt, D. J. (1985). Multiprocessor Hashed-Based Join Algorithms,. *Proc. VLDB 85*, 151- 164.

H, A. M. (2010). Multi join query optimization using Bees Algorithm.

H, R. N. (1991). A pipeline N-way join algorithm based on the 2 way semi-join program. *Knowledge and data Engineering IEEE Transaction*, (pp. 486-495).

Kitsuregawa, M. A. (1990). Bucket Spreading Parallel Hash: A New, Robust, Parallel Hash Join Method for Data Skew in the Super Database Computer (SDC). *Proc. VLDB 90*, 210- 221.

Kosmann, D. (2000). The State of art in Distributed Query Processing. *ACM Computing Surveys*, 422-469.

Kunal Jamsutkar, V. P. (2013). Query Processing Strategies in Distributed Database. *Journal of Engineering, Computers and Applied Sciences*, 71-77.

Lu, H. J. (1990). Hash-based Join Algorithms for Multiprocessor Computers with Shared Memory. *Proc. VLDB 90*.

Lukasz Golab, T. J. (2008). Prefilter: predicate pushdown at streaming speeds. *In Proceedings of the 2nd international workshop on Scalable stream processing system (SSPS "08)*, (pp. 29–37).

M, L. R. (1994). Industrial –strength parallel query optimization: issues and lessons. *information system*, (pp. 311-330).

MW, B. P. (1981). using joins to solve relational Queries. *J.ACM*, (pp. 25 -40).

R.S.G, R. C. (1997). query optimization in distributed relational database. *Journal of heuristics*, (pp. 5-23).

S.M.T, G. M. (2011). A new vertical fragmentation algorithm based on ant collective behavior in distributed database systems. *Knowledge and information systems,*.

S.T, R. S. (1997). optimizing distributed queries : A genetic algorithm approach Annals of operations Research., (pp. 199-228).

Schneider, D. A. (n.d.). A Performance Evaluation of Four Parallel Join Algorithms in a Shared-Nothing Multiprocessor Environment. *Proc. SIGMOD 89*.

Shivnath Babu, K. M. (2005). Adaptive Caching for Continuous Queries. *In Proceedings of the 21st International Conference on Data Engineering (ICDE "05)*, (pp. 118– 129).

Shivnath Babu, R. M. (2004). Adaptive ordering of pipelined stream filters. *In Proceedings of the 2004 ACM SIGMOD international conference on Management of data (SIGMOD "04).*

Shivnath Babu, U. S. (2004). Exploiting k-constraints to reduce memory overhead in continuous queries over data streams. *ACM Trans. Database Syst*, 545–580.

Singh, D. S. (2011). A Novel approach of Query Optimization for Distributed Database Systems. *International Journal of Computer Science,, 8*(1), 307-312.

Sunita Mahajan, M. &. (2012). General Framework for Optimization of Distributed Queries. *International Journal of Database Management System, 4*(3), 35-47.

T, I. T. (1984). On the optimal nesting order for computing N- Relational joins. *ACM Trans. Database Syst*, (pp. 482-502).

Valduriez, P. A. (1984). Join and Semijoin Algorithms for a Multiprocessor Database Machine. *ACM Trans. Dafabase Syslems, 9*(1), 133- 161.

Valduriez.P, O. M. (2011). *Principles of distributed database system 3rd ed.,springer*.

Wolf, J. L. (1990). An Effective Algorithm for Parallelizing Sort Merge Joins in the Presence of Data Skew. *Proc. 2nd Intl. Symp. Databases in Parallel and Distributed Systems*, 103-115.

Y.E, I. (1996). Query Optimization . *ACM Comput.Sur28*, (pp. 121-123).

Authors Profile

Dr. P. Madhubala pursued Ph.D. in Computer Science from Mother Teresa Women"s University, kodaikanal in the year 2017. She is currently working as Head & Assistant Professor in PG & Research Department of Computer Science, Don Bosco College, Periyar University, Salem since 2007. She has published more than 15 research papers in reputed international journals and participated in conferences including IEEE and it"s also available online. Her main research work focuses on Cloud Security and Privacy, Cryptography Algorithms, Network Security, and Big Data Analytics. She has 17 years of teaching experience and 5 years of Research Experience.

G.Sakthivel pursued Bachelor of Science from Sacred Heart College, Madras University, Master of Computer Science from Thiruvalluvar University and M.phil of computer science in the year 2009. He is currently pursuing Ph.D. and working as Assistant Professor in PG Department of Computer Science, Arignar Anna College (Arts & Science) since 2010. He has published more than 5 research papers in reputed international journals and presented papers in National and International conferences. His main research work focuses on Set containment Joins, Cryptography Algorithms, Big Data Analytics and Data Mining. He has 10 years of teaching experience & 2 years of Research Experience.