Evaluation of gene value and heuristic function of alternate plans in multi database system using Genetic Algorithm

[1]Sambit Kumar Mishra and [2]Prof.(Dr) Srikanta Pattnaik

[1]Associate Professor, Dept. of CS & ENGG, Ajay Binay Institute of Technology, Cuttack
[2]Professor, Dept. of CS & ENGG, S.O.A. University, Bhubaneswar

Abstract

The major tasks in multiple query processing in multi database system are common operation or expression identification and global execution plan construction. Each query can have several alternative evaluation plans, each with a different set of tasks. Therefore the goal of multiple query processing is to choose the right set of plans for queries which minimizes the total execution time by performing common tasks only once. The objectives in the multiple query processing are to increase system throughput and decrease single query response time. I have retrieved the alternate plans and the tasks, estimated cost of plans and heuristic function of alternate plans in multi database system by applying the genetic algorithm technique.

Index Terms : *Plan , genes, query optimizer, query execution plan, dynamic programming*

1. Introduction

Database users may not need only their own data, but also data in other databases to solve a specific problem. Data may reside in different databases for many reasons such as ownership, security classification, performance or size. Data may be stored redundantly in different computers for reliability or survivability. In addition , hardware or software upgrades may create a need for integrated access to both old and new databases or for a tool to aid data migration from old to a new system. The multiple query processing is generally considered based on the functional dependency among the fragment attribute, the aggregate attribute, and the group by attribute. The main problem of multiple queries processing is the query redundancies produced by the different queries. The method of multiple queries processing realizes multiple queries optimization through identifying and eliminating the query redundancies among multiple queries. Before query processing can be undertaken, the query processor must translate the query into a suitable internal representation. In a relational database all information can be found in a series of tables. The most common queries are select-project-join queries .

DOI: 10.5121/ijdms.2011.3307

A query optimizer selects among the many alternative query execution plans (QEPs), the one with the least estimated execution cost, according to a given cost function. The objective functions of query optimization may take many different forms. One may try to find a query evaluation plan that optimizes the most relevant performance measures, such as the response time, CPU, I/O, and network time and efforts, memory or storage costs, resources usage (e.g. energy consumption for battery-powered mobile systems), a combination of the above, or some other performance measures.

Traditional query optimizers expect to deal with queries involving only a small number of relations, usually requiring less than 10 join operations and therefore have relied on the use of enumerative optimization strategies (e.g. dynamic programming) that consider most of the alternatives if not all. Dynamic programming (as well as other enumerative optimization techniques) finds an optimal plan.

2.Review of Literature

V Raman, V Markl, D. Simmen et.al[1] have discussed in their paper that the optimizer adjusts the plan based on feedback from runtime environment. This technique adds CHECK-points in the query plans. While executing the plan, when the checkpoint is reached, the estimated cardinalities are checked against the actual run-time cardinalities. If the check fails, the plan is re-optimized based on actual cardinalities .

Markl, Raman, Simmen et.al[2] have discussed in their paper that queries are optimized at compilation time. By the time the program is invoked and the plan is used the database state may change. This change in state may render the plan infeasible when some tables or indices referenced in the plan are deleted or there were bulk-loads which changes the data distributions.

T.Sellis et.al [4] have used a heuristic algorithm which performs a search over some state space defined over access plans. The search space is constructed by defining one state for each possible combination of plans among the queries.

S.Babu et.al[5] have elaborated in their paper that multi database systems use a query optimizer to identify the most efficient strategy called plan to execute declarative queries. Query optimizers often make poor decisions because their compile time cost models use inaccurate estimates of various parameters.

Mishra Sambit Kumar et.al[7] have discussed in their paper that the first step to represent this problem as a genetic algorithm problem is determining the chromosome , genetic algorithm operators and fitness function. For the crossover, one point in the selected chromosome would be selected along with a corresponding point in another chromosome and then the tails would be exchanged. Mutation process causes some bits to invert and produces some new information. The only problem of mutation is that it may cause some useful information to be corrupted. Therefore the best individual is used to proceed forward to the next generation without undergoing any change to keep the best information. Defining fitness function is one of the most important steps

in designing a genetic algorithm based method, which can guide the search toward the best solution.

Falout C.Barber et.al [6] have evaluated the cost function in task allocation which is the sum of inter processor communication and processing cost and found that they are actually different in measurement unit.

Lynda Tamine et.al [8] have discussed in their paper that the idea of combining multiple representations of either queries or texts using different retrieval techniques is in order to improve the retrieval performance. The whole process of query evaluation is based on both general genetic optimization methodology and relevance feedback technique.

L Amsaleg et.al[3]have discussed in their paper that in large-scale systems, many queries can run for a very long time. As a result, there is interest in allowing users to control properties of queries while execution.With the advent of internet and distributed systems there is heterogeneity in the types of data sources that a DBMS is supposed to handle e.g. locally stored tables, data streams, sensor networks etc. This poses various challenges like different data rates, unknown statistics about data, and variation in the distribution of data leading to change in the selectivities of operators, delayed data sources.

3.Query Optimization challenges

The key constituents of the query evaluation component of an SQL database system are the query optimizer and the query execution engine. One aspect of optimization is where the system attempts to find an expression equivalent to the given expression, but more efficient to execute.

Another aspect is selecting a detailed strategy for processing the query. The task of an optimizer is computationally challenging since, for a given SQL query there can be a large number of possible execution plans – specifically, for a query with n base relations, the number of plans in the strategy space is at least O(n!). When queries are optimized at the time they are submitted by the user, the selection process can add a substantial overhead to the execution time of the query.

4.Efficient selection strategies

For the cost-based optimizers, the use of dynamic programming to efficiently find a good plan was proposed. The dynamic programming approach is based on the assumption of the principle of optimality , which states that the optimal solution to a problem is a combination of optimal solutions to its sub problems. While dynamic programming (DP) works very well for moderately complex queries with up to around a dozen base relations, it usually fails to scale beyond this stage in current systems due to its inherent exponential space and time complexity. Therefore, DP becomes practically infeasible for complex queries with a large number of base relations.

The goal here is to a priori identify the parametric optimal set of plans for the entire parameter space at compile time , and subsequently to use at run time the actual parameter settings to identify the best plan – the expectation is that this would be much faster than optimizing the query from scratch.

5.Runtime refinement of plan choices

Query optimizers often make poor decisions because their compile-time cost models use inaccurate estimates of various parameters. There have been several efforts in the past to address this issue, which can be categorized as – strategies that make decisions at the start of query execution and strategies that make decisions during query execution.

The following strategies are essential during query execution.

1. Perform query optimization just before query execution. This method is not very efficient, especially if the query is executed repeatedly.
2. Find the best execution plan for all possible values of the parameters and lookup the best plan for the current parameter values at runtime .
3. Perform part of the optimization at compile time and defer any decisions that are affected by the parameter values to execution time

6.Genetic Algorithm Procedure

The genetic algorithm starts with an initial population which is usually chosen at random and contains a wide variety of numbers. Each solution is evaluated according to an evaluation function. The population evolves from one generation to the next through the application of genetic operators, i.e. selection , crossover, and mutation. During selection operation, members of the population are selected in pairs to produce new possible solutions. The fitter a member of the population , the more likely it is to produce offspring. Crossover operator is then used to result in offspring inheriting properties from both parents. The offspring is evaluated and placed in the next population, possibly replacing weaker members of the last generation. Crossover operator is applied with a certain probability , crossover rate. Mutation operator is used to allow further variation of offspring. Mutation operator is also applied with a certain probability , mutation rate. The length of chromosome is equal to the number of operations in the query tree. Each integer at the particular position in chromosome represents the position selected for a particular operation. The initial population is generated by using a random number between one and the number of sites from the uniform distribution . The fitness of each individual member in the population is the parameter to query execution cost. Genetic algorithm keeps track of the best fitness chromosome in the population.

7.Problem Formulation

Individual plan is represented as chromosome and individual task in a plan is represented as gene. Since a gene in a chromosome represents the plan selected for the query corresponding to the gene position, in the mutation operation the plan number is only replaced with randomly selected valid plan's number for that query. Therefore a mutation operation always generates valid solutions.

8.Function Evaluation & Algorithm

total_generation=50

relation=50

querysize=20

plan_chrom=7

crossover_pc=0.07

mutation_pm=0.001

sum_plan=0;

sum_plan1=0;

fitness_plan=0;

for i=1: querysize

planselect(i)=plan(i)/querysize*plan_chrom;

real_cost(i)=planselect(i)/querysize+cputime;

est_cost(i)=real_cost(i)/querysize;

sum_plan1=sum_plan1+real_cost(i);

weight(i)=(plan(i)*querysize)/querysize-plan(i));

fitness(i)=1+((querysize*weight(i))/((weight(i)^2)+querysize^2));

fitness_plan=fitness_plan+fitness(i);

selection_plan(i)=fitness(i)/fitness_plan;

sum_plan=sum_plan+planselect(i);

end

average_association_ degree_coefficient between plans= sum_plan/(relation*querysize);

averagesum=sum_plan/querysize;

heuristic function, fsk_est=

sum_plan1+min_est_cost;

```
for i=1 : querysize

if(averagesum > est_cost(i))

genevalue(i)=( averagesum-(real_cost(i))+(est_cost(i));

else

genevalue(i)=(real_cost(i)-averagesum)+ est_cost(i));

end

end
```

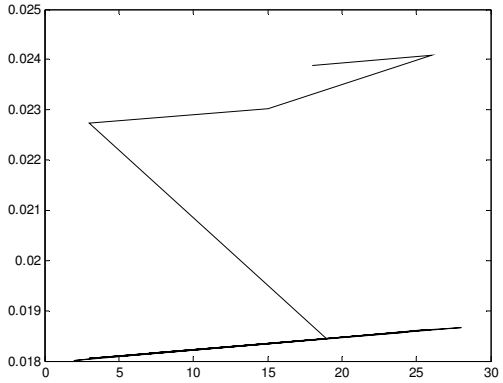
Table –I

Plan	Est_cost	Real_cost	Gene_val	Fitness
10	0.018219	0.36438	0.020669	1.5
18	0.018419	0.36838	0.016869	1.1098
26	0.018619	0.37238	0.013069	0.7809
25	0.018594	0.37188	0.013544	0.80769
15	0.018344	0.36688	0.018294	1.3
3	0.018044	0.36088	0.023994	1.1711
5	0.018094	0.36188	0.023044	1.3
24	0.018569	0.37138	0.014019	0.83784
7	0.018144	0.36288	0.022094	1.4174
28	0.018669	0.37338	0.012119	0.73585
17	0.018394	0.36788	0.017344	1.1711
19	0.018444	0.36888	0.016394	1.0525
4	0.018069	0.36138	0.023519	1.2353

Average association degree coefficient=0.07397
Heuristic function of alternate plan, fsk_est=7.3545

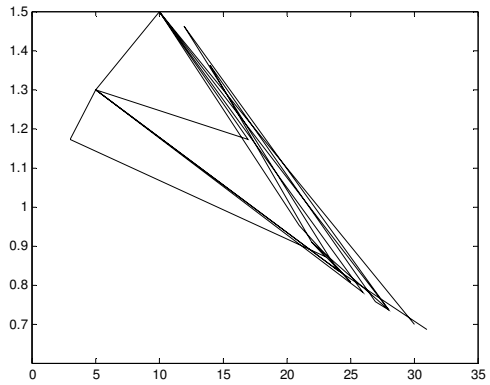
Figures :

Figure-1 : Est_cost



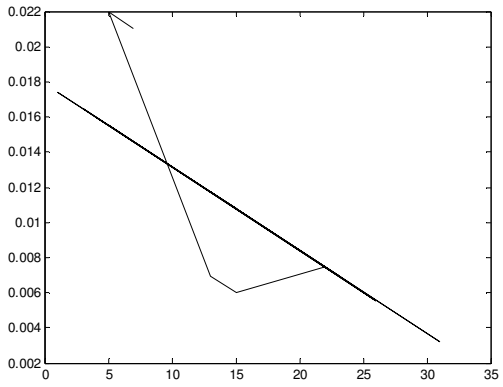
Plan

Figure-2 : Fitness



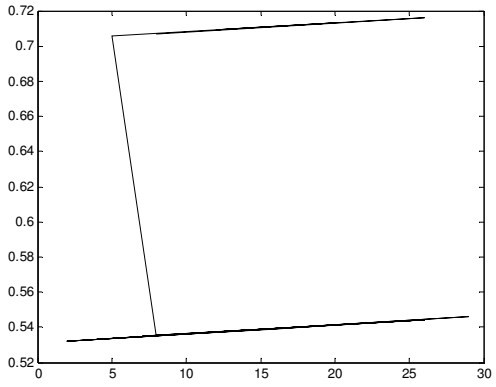
Plan

Figure-3 : Genevalue



Plan

Figure-4 : Real_cost



Plan

9.Conclusion

Clearly original multi query processing problem in multi database will be NP-hard if the above decision problem is NP-Complete. It is easy to see that multiple query processing belongs to NP since a nondeterministic algorithm needs only guess one plan for each query and check whether the cost of the global access plan obtained by merging the guessed local access plans is less than or equal to combining the access plans can be easily done in polynomial time and therefore the checking steps takes only polynomial time.

Each query has a number of possible solution plans , and each plan of a query contains a set of tasks, which when executed in a certain order and produce the answer for the query. Each task also has an associated cost , and for convenience the cost value is represented by a positive integer number. Alternative plans of a query , and other queries in the query set , may contain the

same task. Therefore it is required to determine a set of tasks, with minimal total cost that contains all the tasks of at least one plan of each query.

Reference :

[1] V Raman, V Markl, D. Simmen, G. Lohman, Demo,Progressive Optimization in Action, VLDB-2004

[2] Markl, Raman, Simmen, Lohman ,Robust Query Processing through Progressive Optimization, SIGMOD-04

[3] L Amsaleg, M Franklin, A Tomasic, T Urhan, Scrambling Query Plans to CopeWith Unexpected Delays, 4th International Conference on Parallel and Distributed Information Systems (PDIS-1996).

[4] Sellis, "Multiplequery optimization",IEEE transactions on knowledge and data Engineering , Vol-2, June-1990.

[5] Shivnath Babu , Pedro Bizarro , David DeWitt, proactive re-optimization with Rio, SIGMOD-2005.

[6] Falout C, Barber, R.Flicker.M, Journal of intelligent Information Systems 2000.

[7] Mishra Sambit Kumar , Pattnaik Srikanta(Dr.),retrieval of average sum of plans and degree coefficient between genes in distributed query processing November -2010,IJCSI –Vol-7, Issue-6 , pp -337-342.

[8] Lynda Tamine , Claude Chrisment , Mohand Boughanem , University of Toulouse, France , IPM-2003.

[9] Alfons Kemper , Christian Wiesner ,Dynamic Distributed query processing "Proceedings VLDB Conference ,Rome,Italy2003.

[10] Murat Ali Bayir, Ismail H.Toroslu , and Ahmet Cosar , "Genetic algorithm for the multiple query optimization problem", IEEE transactions on Systems , Vol-37, No.1, January 2007.

[11] Zehai Zhou , Department of FACIS, University of Houston , USA, Journal of Information of Computing science vol-2, No.4, 2007.

[12] Bindu M.Hima , Department of Computer Sc., JIIT , Chennai , International Journal of Computer Science Issues , Vol-2, Issue-5, June-2007.