A Query Optimization Algorithm of Database Based on Layered Backtracking

Weihua Wu , Jigui Liou and Yunfeng Li
Shandong Urban Construction Vocation College
Jinan Shandong , China
ise_yuann@ujn.edu.cn

***Abstract*—Based on basic ideas of DP algorithm and backtracking method, a new query optimization algorithm based on layered backtracking is proposed, in optimization of simple query,"optimal" solution can be provided; As for some complex applications, it achieve balance between complex enumerative algorithm and quality of solution based on algorithm, and get the "second best" optimal result to improve efficiency of algorithm and save resources needed in operating environment.**

Keywords-query optimization; DP algorithm; backtracking; bounding function

I. INTRODUCTION

To some degree, the success of commercialization of database systems should be attributed to mature query optimization techniques[1]. User defines query sentence with use of SQL or OQL languages, optimizer of database system finds the best way to execute the query. Optimization query means that when user consider query choices,he or she is provided with many solutions, he or she can use the cost model to assess cost of each solution, and select the least costly solution[2].

Core component of a query optimizer is its search strategy or enumeration algorithm. Enumeration algorithm of optimizer decide which solution are listed, the most typical enumeration method is based on the DP (Dynamic Programming)[3]. Because the problem for finding the optimal solution is NP unsolvable problem, when query optimizer determines which enumeration algorithm should be used, balance between complexity of the algorithm and quality of programs is considered. DP algorithm represents an extreme point: DP has complexity of exponential time and space, can generate optimal solution[4]. Other algorithms have lower complexity then DP algorithm, but these algorithms can not find the solution with minimum cost.

In DP algorithm, "solution" is defined as tree of query operational character, which means table scan, index scan, sorting, nested loop join and so on[5].The optimal solution in algorithm is the query operational character with the minimum cost.Assuming all solution is a space tree, the optimal solution can be found through searching space tree with certain searching strategy.

II. A NEW QUERY OPTIMIZATION ALGORITHM BASED ON LAYER BACKTRACKING

Backtracking is a recursive searching algorithm,like DP, it is also belong to exhaustive search method[6]. As for database with high complexity,exhaustive search method costs a lot of resources. Based on backtracking method, we propose a query optimization algorithm based on layer backtracking[7].

A. Definition of Solution Space in Query Optimization Problem

To describe the query optimization problem, we can regard solution space in SPJ query optimization progress as a tree. Like DP algorithm, we define access plan as one or two operational characters, query operational characters, table scan, index scan, sorting, nested loop, etc. And a table may have several different access plans. And solution space of query optimization problem can be defined as a tree of all access plans whose root nodes are virtual nodes.

B. Bounding Function of Query Optimization Problem

In order to avoid invalid search and improve system's efficiency, bounding function is used to eliminate the sub-node w without optimal solution. For example, when the algorithm search in the first sub-node, cost of connection between nodes in the second layer and sub-node s is regarded as the best solution. In the following searching process, cost of connection between root nodes and all access plans in the nodes are calculated, if the cost is lower than the best solution, the search process continues; If the cost greater than the best solution, the sub-tree is eliminated, then turn to upper node.

C. Query Optimization Algorithm Based on Layered Backtracking

Through depth-first search, backtracking method can find optimal solution in query optimization[8]. However, the performance of backtracking method is closely related with depth of solution space tree, when depth of the tree increases, the performance of the algorithm decline dramatically[9]. On the basis of DP algorithm and backtracking method, we propose a new query optimization algorithm based on layer backtracking. The idea behind this is that solution space tree of backtracking method is divided into many layers, height of each layer is *k*. At first, depth-first search is carried out in sub-tree of solution space whose depth is *k* to find best solution. Then, the sub-trees are ignored, and layered search continues. In this way, solution space tree with m layers are divided into sub-trees with height of less than *k*, and the query optimization problem is decomposed into several "sub-problems".

The query optimization algorithm based on layered backtracking is described as follows:

input: Search q in R_1, ……, R_n with SPJ, height of layer for each search is k (k<=m)

```
output: query solutions for q
1 for i=1 to n do
2   { optPlan({Ri})=accessPlans(Ri)
3     prunePlans(optPlan({Ri})
4   }
5 generataTree(optPlan({Ri}), ……, optPlan({Ri}))
6 t=1,s=m,
7 while t>0 do
8   { y=min(k,s)
9     if f(y,t)<=g(y,t)
10        for i=f(y,t) to g(y,t) do
11          { x[t]=h[i]
12            if Bound(t)
13              { if Solution(t) output(x)
14                else t++
15              }
16          }
17      else t--
18      if t=0
19        { s=s-k
20          if s>0
21            { generateTree2(optPlan({Ri})/x[t])
22              t=1
23            }
24        }
25   }
26 finalizePlans(x)
```

In the algorithm, *k* is the base for layer, *m* is height of the solution space tree, *t* represents depth of current expansion node in the solution space whose height is *k*. A collection of access plans for tables *R1, … …, Rn* is obtained in 1-4 lines, and second-class access plans are eliminated through the function *prunePlans()*. The function *generataTree()* generate permutation tree with all access plans as nodes, thus solution space of optimization query algorithm. The function *Solution()* is used to whether a viable solution for a "sub-problem" is obtained in the current node. If the return value is true, it means that *x[1:t]* is the viable solution for the current "sub-problem". At this point, the function *output(x)* records *x*. It value of *Solution()* is false, it indicates that *x[1:t]* is part of solution for the current "sub-problem", futher longitudinal search is needed. *h[i]* represents the ith optional value in the current extension node. The function *Bound(t)* indicates bounding function of current expansion node.

If the lowest cost is found with the algorithm within the subtree whose height limit is *k(y)*, the the function *output(x)* is uesed to record values, with sub-node of the sub-tree as root, the function *generateTree2()* is used to regenerate solution space tree for remaining nodes, and the above processes are repeated. After while loop is finished with the algorithm, the whole backtracking search progress end. Finally the algorithm connet all best solutions at every layers with the function *finalizePlans(x)*, after processing and use of *Project, Sort, Group-by* and other operational characters, the whole query solution is generated.

III. Performance Analysis Layered Backtracking Algorithm

A. Time Complexity

When the query optimization algorithm run the first ground backtracking, the number of sub-nodes need traversing are $n(n-1)\ldots\ldots(n-k+1)$; In the second round backtracking, because some sub-trees are ignored, the number of sub-nodes need traversing are $(n-k)(n-k-1)\ldots\ldots(n-2k+1)$, and complexity is far smaller than the first ground. Taking into computing time of the function *Bound(t)*, time complexity of the algorithm is $O(n(n-1)\ldots(n-k+1))$.

B. Space Complexity

Because in the algorithm, the tree of solution space is stored, so the space complexity for the whole algorithm is $O(n^k)$.

C. Some Explanations on Layered Backtracking

Layered backtracking may not be able to find the optimal solution, but it reduces time complexity and space complexity of the algorithm, and achieve trade-off between performance and efficiency[10].

- Layered backtracking requires many iterative backtracking to optimize query. .For example, if $m=10$, $k=4$, three round iterative backtracking are needed. After each iteration, layers of the solution space tree reduces by *k*. After the last iteration, the solution space has only two layers.
- When $k=m$, the layered backtracking method is equivalent to normal backtracking method, when the algorithm is in fact is a exhaustive algorithm, and can find the optimal solution. When processing query optimization problem of small solution space, system resources have not limit, we can set $k=m$.
- When $k=1$, the layered backtracking method is equivalent to greedy algorithm. In each backtracking, the algorithm only search in one layer of solution space, when quality and performance of the algorithm depends largely on bounding function, quality of solution is not good. It is recommended that *k* should be set according to features of dababase, in general, $k>2$. Parameter *k* can also be set by user to limit optimization time or as an optimized level. Obviously, greater *k*(more resources) means that better plan is generated by layered backtracking, because, the algorithm is the closest to classic backtracking method.

IV. Conclusion

The process to find the optimal solution is a NP unsolvable problem,layered backtracking method is a query optimization algorithm which is suitable for simple query and complex database[11]. In this paper, the author only makes experimental study on the algorithm, and does not give detailed feasibility studies on its performance in large database, as for setting of k, he also only gives qualitative discussion and simple recommendations. Next, more extensive testing will be made to analyse query status of different databases to find the most scientific layered approach, so as cost of obtained solution is close to that of solution with exhaustion method

REFERENCES

[1] Donald Kossmann and Konrad Stocker. Iterative Dynamic Programming:A New Class of Query Optimization Algorithms. ACM Transactions on Database Systems, Vol.25, No.1, March 2000, pp.151--155.

[2] Franklin, M.Jonsson and Kossmann. Performance Tradeoffs for Client-server Query Processing. In Proceeding of the ACM-SIGMOD Conference on Management of Data, ACM, New York, 1996, pp.149-160.

[3] S.Chaudhuri and K.Shim. Optimization of Queries with User-defined Predicates. In Proceeding of the 22nd International Conference on Very Large Data Bases,1996, pp.87-98.

[4] S.Ganguly, W.Hasan and R.Krishnamurthy. Query Optimization for Parallel Execution. In Proceeding of the ACM-SIGMOD Conference on Management of Data, ACM, New York, 1992, pp. 9-18.

[5] Eppstein D. Small maximal independent sets and faster exact graphcoloring. Lecture Notes in Computer Science 2125 : Proc 7th Worksh Algorithms and Data Structures. Springer-Verlag, 2001, pp. 462-470.

[6] Fomin F V, Grandoni F, Kratsch D. Measure and conquer: Domination- A case study. Proceedings of the 32nd International Colloquiumon Automata, Languages and Programming(ICALP 2005). Springer, 2005:191-203.

[7] Deb K. Apopulation-based algorithm-generator for real-parameter optimization, KanGAL Rep.200303[R].2003.

[8] Eppstein D. Quasiconvex analysis of backtracking algorithms. Proceedings of the 15th ACM-SLAM Symposium on Discrete Algorithms(SODA 2004), 2004:781-790.

[9] Beigel R. Finding maximum independent sets in sparse and generalgraphs[C/OL]//Proc 10th ACM-SIAM Symp Discrete Algorithms,1999:S856-S857.http://www.eecs.uic.edu/soda.PS.gz.

[10] GOEL P. An implicitenum eration algorithm to generatetests for combinational logic circuits. IEEE Transactions on Computers, 1997, 30(3),pp.215-222.

[11] Garey M, Johnson D. Computers and intractability: A guide to the theory of NP-completeness. San Francisco:Freeman,1979.