ISSN: 2278 – 1323
International Journal of Advanced Research in Computer Engineering & Technology (IJARCET)
Volume 2, Issue 3, March 2013

ADAPTIVE SHARING METHOD FOR MULTIPLE CONTINUOUS QUERY

Mariat Thomas[1],Mr. G. Naveen Sundar[2]
Department of Computer Science and Engineering, Karunya University
Coimbatore, India

Abstract— **As the data management field has diversified to consider settings in which queries are increasingly complex, statistics are less available, or data is stored remotely, there has been an acknowledgment that the traditional optimize-then-execute paradigm is insufficient. The main objective of our project is to develop and produce an optimized global execution plan for the collective evaluation of a static set of multi-way continuous queries. Generally in a data stream environment, a considerable number of continuous queries are registrated in advance and should be processed continuously. For that there is a need of multiple query optimizations. For this purpose, this paper proposes a new method called A-SEGO that optimizes a set of multi-way join queries collectively by tracing a set of promising subplans. The proposed approach uses a cost profile that maintains statistical synopses of the cost of join operations in past global execution plans. Based on these cost statistics and on a user-defined cost-bound parameter, a set of promising subplans is determined to trace them concurrently. This paper focuses on producing an optimized global execution plan for the collective evaluation of a static set of multi-way continuous queries. In real data stream applications, a set of continuous queries registered in a DSMS can be changed as time goes. Therefore, it is necessary to devise an incremental optimization mechanism. With this approach, the system needs to store existing query computations, identify the common computations between the new query and the existing query plan, choose optimally among multiple sharing paths, and add unsharable new computations to the plan[4].**

Keywords— **Data Streams, Query optimization.**

I. INTRODUCTION

A data stream is massive unbounded sequence of data elements continuously generated. Query processing for data streams should be continuous and rapid, which requires strict time constraint. To guarantee this constraint, optimize the evaluation order of multiple join operations in a set of continuous queries using a greedy optimization strategy so that the order is re-optimized dynamically in run-time due to the time-varying characteristics of data streams. However, this method often results in a suboptimal plan because the greedy strategy traces only the first promising plan. This paper proposes a new multiple query optimization approach, Adaptive Sharing-based Extended Greedy Optimization Approach (A-SEGO). Given continuous queries with multiple join operations, they simultaneously trace a set of promising plans to reduce the possibility of producing a sub-optimal plan. Also it can control the time to optimize continuous queries depending the current processing load by controlling the number of traced plans.

The basic objective of multiple query optimizations is to reduce the overall evaluation cost of multiple queries by sharing the results of common sub-expressions. This paper proposes an improved scheme called an Adaptively Sharing-based Extended Greedy Algorithm (A-SEGO). Given continuous queries with multi-way join operations, they simultaneously trace a set of promising plans to reduce the possibility of producing a sub-optimal plan. Also it can control the time to optimize continuous queries depending on the current processing load by controlling the number of traced plans. Experiment results illustrate the performance of the A-SEGO in various stream environments[6].

When we optimize multiple queries in data stream environments, we should consider the optimality of the optimized plan as well as the time to optimize since continuous queries should be processed in real-time. Consequently, previous approaches have employed a greedy with respect to either the result size of a join predicate. Even if the strategy can optimize continuous queries very quickly with almost no overhead, it is apt to produce a sub-optimal plan because there is no backtracking mechanism to undo a non-optimal intermediate order of join operations. However, employing a backtracking mechanism in data stream environment is impractical because it has a huge overhead.

To minimize the time and storage complexities of the proposed algorithm, only the mean and standard deviation of the past costs of a join operation are employed. Various statistical models such as a poisson distribution and binomial distribution can enhance the cost model of the proposed algorithm. In addition, this paper focuses on producing an optimized global execution plan for the collective evaluation of a static set of multi-way continuous queries. In real data stream applications, a set of continuous queries registered in a DSMS can be changed as time goes. Therefore, it is necessary to devise an incremental optimization mechanism[15].

II. SYSTEM ARCHITECTURE

The proposed evaluation scheme is composed of four components: a query registration module, a query execution module, a run-time monitoring module and Incremental Multi-Query Optimization module (Fig. 1).

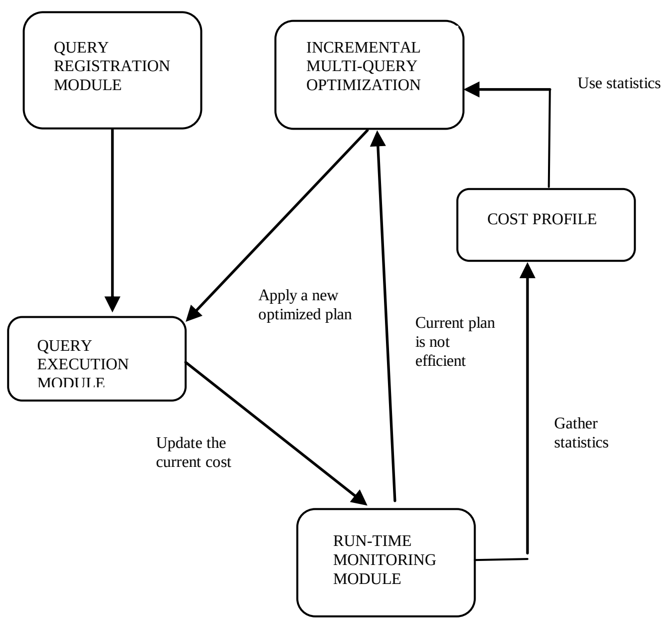
QUERY REGISTRATION MODULE
INCREMENTAL MULTI-QUERY OPTIMIZATION
Use statistics
COST PROFILE
Apply a new optimized plan
QUERY EXECUTION MODULE
Current plan is not efficient
Gather statistics
Update the current cost
RUN-TIME MONITORING MODULE

Fig.1: The overall system architecture

A continuous query is registered in the query registration module, and the query execution module evaluates the queries based on the optimized execution plan of the queries. While evaluating the queries continuously, the run-time monitoring module gathers the cost statistics of each join operation in the current execution plan and updates them into the cost profile[6]. Predicate Indexing for Incremental Multi-Query Optimization, examines the effectiveness of the current execution plan[15]. With incremental multi-query optimization approach, the system needs to store existing query computations, identify the common computations between the new query and the existing query plan, choose optimally among multiple sharing paths, and add unsharable new computations to the plan (Fig 2).

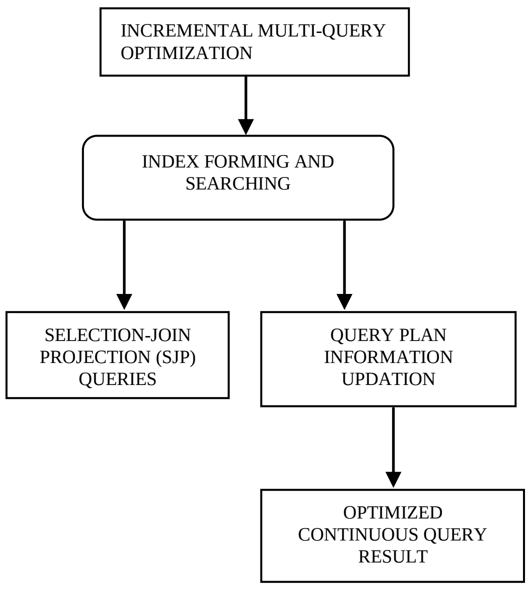
INCREMENTAL MULTI-QUERY OPTIMIZATION
INDEX FORMING AND SEARCHING
SELECTION-JOIN PROJECTION (SJP) QUERIES
QUERY PLAN INFORMATION UPDATION
OPTIMIZED CONTINUOUS QUERY RESULT

Fig.2 : The system architecture for Incremental Multi-Query Optimization

To minimize the time and storage complexities of the proposed algorithm we present a relational schema that stores the computations, which are the two essential parts of the Incremental Multiple Query Optimization (IMQO) framework we proposed to allow the efficient construction of the optimal evaluation plan for multiple continuous queries[11]. we propose a new approach, Incremental Multi-Query Optimization, by adding new query computation incrementally to the existing query plan with heuristic local optimization.

We focus on the Index and Search for selection-join projection (SJP) queries; this presents the most thoroughly investigated effort so far on the most common query types (SJP). Previous work discussed other query types, the sharing strategies and the actual continuous query plan construction .The constructed plan will match the stream data on the fly to produce continuous query results[11].

A. Query Registration Module

In this module a continuous query is registered in the query registration module, and the query execution module evaluates the queries based on the optimized execution plan of the queries. While evaluating the queries continuously, the run-time monitoring module gathers the cost statistics of each join operation in the current execution plan and updates them into the cost profile.

B. Developing Cost Profile for Global Plan Generation

In this module from the given a set of multi-way join queries Q= {q1, q2…qn}, is denoted as J(Q) the total number of distinct join predicates in Q. To find the optimized global execution plan of the queries, the cost of every join operation in all the possible subplans for Q should be monitored in eqn (2). However, because the number of possible subplans can be very large, such monitoring requires huge run-time overhead. Although the cost of a join operation may be changed unpredictably in run-time, it does not vary randomly. Instead, because it depends on the characteristics of a target application domain, it varies within a frequently occurring range. Accordingly, the statistics of the past cost of a join operation are used to estimate the current cost of join operation in eqn (3). When a join operation of the current plan has never been executed before, a new entry for the operation is inserted in the cost profile. If the past cost statistics of a join operation Jx in the cost profile are monitored again in the current global plan, the newly monitored actual cost v (cost (Jx)) of Jx is reflected in the cost profile as follows:

$$\mu(cost(J_X))^{new} = \frac{\mu(cost(J_X))^{old} \times count(J_X) + \vartheta(cost(J_X))}{count(J_X) + 1} \qquad (1)$$

$$\sigma(cost(J_X))^{new} = \frac{\mu(cost(J_X)^2)^{old} \times count(J_X) + \vartheta(cost(J_X)))^2}{count(J_X) + 1} - \mu(cost(J_X)^2)^{new} \qquad (2)$$

$$count(J_X) = count(J_X) + 1 \qquad (3)$$

Given a global execution plan PQ for a set of queries$Q = \{q1, q2, \ldots, qn\}$, let $Q(Jx)\ (pQ)$ denote the set of queries that reuse the result of a join operation Jx in PQ. Furthermore, let widenote the window size of a query qi. The sharing degree $\xi(Jx|PQ)$ of a join operation Jx in PQ is defined in eqn (4).

$$\varepsilon((J_X/P_Q) = \frac{1}{W_{max}} \sum_{i=1}^{|Q(J_X)|} W_{i,} q_i \in Q(J_X) \text{ and } W_{max} = \max(W_i) \tag{4}$$

C. Developing Cost Profile for Subplans

In this module from the given the value of a cost-bound parameter k for a set of multi-way join queries Q, let SP^m Q denote the set of all possible m-subplans for Q, and let $sp^m{}_{min}=\psi[l^m{}_{min}, u^m{}_{min}]$ denote an m-subplan with the smallest cost upper bound um min among all the m-subplans in $SP^m{}_Q$. The set of candidate m-subplans CSP^m traced simultaneously by A-SEGO is defined as follows:

$$CSP^m = \{sp^m{}_x | u^m{}_{min} < sp^m{}_x \in sp^m{}_Q\} \tag{5}$$

C.1 Extending CSPS

In this the set of all possible 1-subplans SP1 is {sp11, sp12, sp13} where sp11={J1}, sp12={J2} and sp13={J3}. First, the effective cost of each subplan in SP1 is calculated. As a result, the average effective cost of sp1 1 with the sharing operation J1 and its standard deviation.

C.2 Cost-Bounds of CSPS

In this step fix the greater the value of a cost-bound parameter k, the wider the cost-bound of a subplans becomes; this can potentially increase the cardinality of CSP. As a result, the optimality of A-SEGO can also be improved, and the time to generate the final plan may also be increased[11]. Depending on the current workload of a DSMS, the cardinality can be adaptively adjusted by varying the value of k. The proposed algorithm can examine the entire search space exhaustively when $k=\infty$; when $k=0$, it can merely simulate a typical greedyoptimization approach.

D.Parallel Join Methods

In this module one of the major tasks of a query optimizer is to determine the method for each join to be performed since there are usually a number of ways to perform a particular join with different costs. In the multiprocessor environment, the selection of join methods becomes more complex. First, the number of join methods increases. For uniprocessor system, the sort-merge join, nested loops join and hash-based join are three major join methods. In a multiprocessor system, each of these methods has a number of variations with different performance. Second, there are more parameters that affect the cost of a join in multiprocessor systems than in uniprocessor systems, such as number of processors participating in the join and the architecture of the system[15]. This module proposes a new method called A-SEGO that optimizes a set of multi-way join queries collectively by tracing a set of promising subplans. The proposed approach uses a cost profile that maintains statistical synopses of the cost of join operations in past global execution plans. Based on these cost statistics and on a user-defined cost-bound parameter, a set of promising subplans is determined to trace them concurrently.

E. Predicate Indexing For Incremental Multi Query Optimization

In this module We present a relational schema that stores the computations of a shared query evaluation plan, and tools that search the common computations between new queries and the schema, which are the two essential parts of the Incremental Multiple Query Optimization (IMQO) framework we proposed to allow the efficient construction of the optimal evaluation plan for multiple continuous queries. As part of the IMQO framework, a comprehensive computation indexing schema and a set of searching tools were presented. The schema stores shared plan computations in relational system catalogs, and the tools search the common computations between new queries and the system catalogs.We implemented the schema and tools on ARGUS to support efficient processing of a large number of complex continuous queries. The empirical evaluation on ARGUS demonstrated up to hundreds of times speed-up for multiple query evaluation via the shared plan construction.

F. Final Execution Plan

In this module the incomplete subplans will be reexecuted. In this find all the subplans and check the condition that if the subplans is in cost profile then get the corresponding cost of that subplans from the table. If not then estimate the cost otherwise for each subplans find all i-subplans if this is exist in cost profile then get the cost of the subplans from the table. If not then estimate the cost for the subplans.

III. PERFORMANCE ANALYSIS

Continuous queries are registered in query registration module and the query execution module evaluates the queries based on the optimized execution plan of the queries. While evaluating the queries continuously, the run-time monitoring module gathers the cost statistics of each join operation in the current execution plan and updates them into the cost profile. In addition, it examines the effectiveness of the current execution plan. When the current plan is no longer effective, the query optimization module seeks a new optimized execution plan. In developing cost profile module from the given a set of multi-way join queries Q= {q1, q2…qn}, is denoted as J(Q) the total number of distinct join predicates in Q. To find the optimized global execution plan of the queries, the cost of every join operation in all the possible subplans for Q should be monitored[6].

Develop the cost profile plan from the given the value of a cost-bound parameter k for a set of multi-way join queries Q, let SP^m Q denote the set of all possible m-subplans for Q, and let $sp^m{}_{min}=\psi[l^m{}_{min}, u^m{}_{min}]$ denote an m-subplan with the smallest cost upper bound um min among all the m-subplans in $SP^m{}_Q$. The set of candidate m-subplans CSP^m traced simultaneously by A-SEGO is defined in eqn (5) .

In final execution plan the incomplete subplans will be re executed. In this find all the subplans and check the condition that if the subplans is in cost profile then get the corresponding cost of that subplans from the table 1. If not then estimate the cost otherwise for each subplans find all i-subplans if this is exist in cost profile then get the cost of the subplans from the table 1. If not then estimate the cost for the subplans. This process will be end when the entire subplans will be inserted in table 1. Then the cost profile process will be done until the user specified value.

Table 1 Candidate Subplan

Pattern	Window Size (sec)	Total Record	Newly Added Record
Pattern 1	3697	13	23
Pattern 2	4305	13409	50
Pattern 3	3697	1802304	13400
Pattern 4	5429	1802304	13400

The pattern types and its window size of each pattern is shown in the table 1.Each pattern consist of records and total number of records is mentioned. And also include newly added records. We have to find each record cost value by using formula. Cost values of each record is shown in table 2.

Table 2 Cost of Candidate Subplan

Pattern	Window Size (sec)	Cost value using AND operator	Cost value using OR operator
Pattern 1	3697	23.78	123.0
Pattern 2	4305	850.0	107600.0
Pattern 3	3697	375200.0	2.47632E7
Pattern 4	5429	549400.0	3.63676E7

Various sub plans are considered for getting minimal cost. After evaluation of many subplan the algorithm select the minimal cost plan .We will get reduced cost. Because the cost of a global execution plan for a set of continuous queries Q can vary dynamically with time, it should be monitored continuously to keep the plan as efficient as possible. If the current plan is no longer efficient, a newly optimized plan should be generated in a timely manner by a re-optimization process. However, invoking a re-optimization process too frequently can degrade the overall performance of continuous query evaluation. The below fig 3 shows the execution time for each pattern .

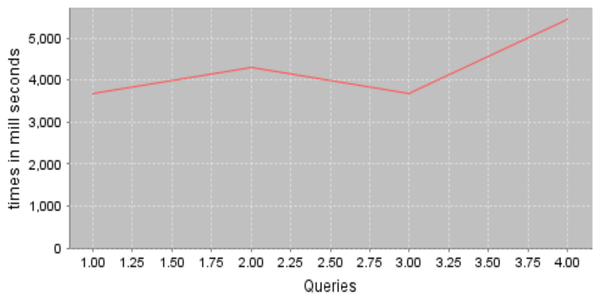

Fig 3 : Queries vs Time

IV. CONCLUSION

In real data stream applications, a set of continuous queries registered in a DSMS can be changed as time goes. Therefore, it is necessary to devise an incremental optimization mechanism for the enhancement work.In addition, this paper focuses on producing an optimized global execution plan for the collective evaluation of a static set of multi-way continuous queries. To minimize the time and storage complexities of the proposed algorithm we present a relational schema that stores the computations of a shared query evaluation plan, and tools that search the common computations between new queries and the schema, which are the two essential parts of the Incremental Multiple Query Optimization (IMQO) framework we proposed to allow the efficient construction of the optimal evaluation plan for multiple continuous queries.

ACKNOWLEDGMENT

The authors would like to thank all the professors and lecturers who helped in completing this paper as well we thank the reviewers for their valuable comments and suggestions.

REFERENCES

[1] Avnur. R, Hellerstein.J,“Eddies: Continuously adaptive query processing”, *Proceedings of ACM International Conference on Management of data SIGMOD*, pp. 261–272,2000.

[2] Babcock. B, Babu. S, Datar. M, Motwani. R, Widom. J, “Models and issues in data stream systems”, *Proceedings of International Symposium on Principles of database systems (PODS)*, pp. 1–16,2002.

[3] Babu. S, Motwani. R, Munagala.K,“Adaptive ordering of pipelined stream filters”, *Proceedings of ACM International Conference on Management of data (SIGMOD)* , pp. 407–418,2004.

[4] Buccafurri.F, Lax.G, “Approximating sliding windows by cyclic tree-like histograms for efficient range queries”, *Data & Knowledge Engineering* 69 (9) 979–997,2010 .

[5] Chen. J, DeWitt. D.J, Naughton.J.F, “Design and evaluation of alternative selection placement strategies in optimizing continuous queries”, *Proceedings of International Conference on Data Engineering (ICDE)*, pp. 345–356,2002.

[6] Hong Kyu Park, Won Suk Lee,”Multiple Continuous Queries Evaluation over Data Streams”, Department of Computer Science, Yonsei University, 134 Shinchon-dong Seodaemun-gu, Seoul, 120-749, Republic of Korea, proceedings of the 8th wseas international conference on applied computer science (acs'08)

[7] Cohen. E, Cormode. G ,Duffield. N, “Structure-aware sampling on data streams”, Proceedings of the ACM SIGMETRICS joint international conference on Measurement and modeling of computer systems (SIGMETRICS '11), ACM, New York, NY, USA, pp. 197–208,2001.

[8] Dalvi.N.N, Sanghai.S.K, Roy. P, Sudarshan.S,“Pipelining in multi-query optimization”, *Proceedings of ACM International Conference on Management of data (SIGMOD)*, pp. 59–70,2001.

[9] Das.A, Gehrke. J, Riedewald. M,“Approximate join processing over data streams”, *Proceedings of ACM International Conference on Management of data (SIGMOD*), pp. 40–51,2003.

[10] Deshpande.A ,Hellerstein. J.M, “Lifting the Bufden of history from adaptive query processing*”, Proceedings of International Conference on Very large data bases(VLDB)* , pp. 948–959,2004.

[11] Hong Kyu Park, Won Suk Lee,” Adaptive optimization for multiple continuous queries”, Department of Computer Science, Yonsei University, 134 Shinchon-dong Seodaemun-gu, Seoul, 120-749, Republic of Korea, Data & Knowledge Engineering 71 (2012) 29–46

[12] Golab .L, Özsu.M.T, “ Issues in data stream management”, *ACM SIGMOD Record* 32 (2) 5–14,2003.

[13] Golab. L, Ozsu. M.T, “Processing sliding window multiway joins in continuous queries over data streams”, *Proceedings of International Conference on Very large data bases(VLDB*), pp. 500–511,2003.

[14] Grand.J, Minker. F, “On optimizing the evaluation of a set of expression”, *International Journal of Parallel Programming* 11 (3) 179–191,1982.

[15] Chun Jin and Jaime Carbonell,"Predicate Indexing for Incremental Multi-Query Optimization",Language Technologies Institute School of Computer Science Carnegie Mellon University Pittsburgh, PA 15213 USA,2008

[16] Hammad.M.A, Aref.W.G, Elmagarmid.A.K, "Stream window join: tracking moving objects in sensor-network databases", *Proceedings of International Conference on Scientific and statistical database management(SSDBM)*, pp. 75–84,2003.

[17] Madden. S, Shah. M, Hellerstein. J.M., Raman.V, "Continuously adaptive continuous queries over streams", *Proceedings of ACM International Conference on Management of data (SIGMOD*, pp. 49–60,2002.

[18] Park. H.K, Lee. W.S, "Adaptive continuous query reoptimization over data streams", *IEICE Transactions 92* (7) 1421–1428,2009.

[19] Park. J, Segev .A, "Using common subexpressions to optimize multiple queries*", Proceedings of International Conference on Data Engineering(ICDE)*, pp. 311–319, 1988.

[20] Raman. V, Deshpande. A, Hellerstein. J.M, "Using state modules for adaptive query processing", *Proceedings of International Conference on Data Engineering (ICDE*), pp. 353–364, 2003.