International Journal of Information, Control and Computer Sciences
ISSN: 2517-9942
Vol:2, No:1, 2008

Weight-Based Query Optimization System Using Buffer

Kashif Irfan, Fahad Shahbaz Khan, Tehseen Zia, and M. A. Anwar

***Abstract*—**Fast retrieval of data has been a need of user in any database application. This paper introduces a buffer based query optimization technique in which queries are assigned weights according to their number of execution in a query bank. These queries and their optimized executed plans are loaded into the buffer at the start of the database application. For every query the system searches for a match in the buffer and executes the plan without creating new plans.

***Keywords*—**Query Bank, Query Matcher, Weight Manager.

I. Introduction

QUERY optimization is well-known to be a computationally intensive process since a combinatorially large set of alternatives has to be considered and evaluated in order to find an efficient access plan [1]. With the increase of IT involvement in an organization data is increasing every year, making its storage and retrieval a challenging issue. Therefore if the organization is geographically dispersed around the world then data storage and it's sharing become more complex. To overcome the storage and sharing challenge, grid databases promise to be a good solution. But, data retrieval still remains a research problem. Recently used grid databases require a support for dynamic query optimization for efficient and cost effective data retrieval.

During the execution of a query in a database management system (DBMS), the query optimizer creates all possible query evaluation plans. All plans are equivalent in term of their final output but vary in their cost, i.e., the amount of time that they need to run [2]. The query optimizer chooses the best one. As soon as the data is retrieved the query and its plans are deleted from memory to free the space for future usage. For the next query same technique is repeated even if the query is already executed; we propose that the queries and their executed plans should be saved into the query bank and loaded into the buffer while accessing the database. We assign a weight to each query before saving into the query bank and the weight increases by one if the same query is executed again.

In the proposed system at the start of database application, the auto query and execution plan loader, loads queries that have higher weights into buffer from query bank according to the buffer size. The query matcher finds a query with queries that are available in buffer; if query matches then its plan is executed directly and the weight manager increases the weight of the query by one. In case the query is not available in buffer then auto query and execution plan loader loads the next queries and their plans into the buffer from query bank and replaces these queries with queries and plans of higher weight until required match if available in query bank is not found. If query matcher fails to find the query then new query plan is created and processed and the query with its optimized plan is saved in query bank for future usage.

In the second section we discuss system architecture of the proposed system. We review related work in section three, and conclude and present future research direction in fourth section.

K. Irfan, F. S. Khan, T. Zia are with Department of Computer Science and IT, University of Sargodha, Sargodha, Pakistan (e-mail: kashif_irfan31@hotmail.com, fahadji@yahoo.com, tehseen_zia@yahoo.com).

M. A. Anwar are with Department of CS and MIS, University College Yanbu, Saudi Arabia (e-mail: anwarma@yahoo.com).

II. System Architecture Overview

In this section we discuss the proposed system in detail. The architecture of the proposed system is shown in Figure 1. The system components consist of new query handler and optimizer (NQHO), query matcher (QM), buffer, weight manager (WM), query bank (QB) and auto query and execution plan loader (AQEPL).

A. New Query Handler and Optimizer

Every query is passed to the NQHO and it passes that query, as it is to the QM. QM match's the query in buffer. If match is found then query is executed and NQHO job finishes. If match is not found then NQHO creates all possible execution plans for the input query and execute a plan, which is cost effective according to the query optimization technique. After the execution of query the query is passed to the WM, which assign weight one to query and that query is stored into the QB for future use.

B. Query Matcher

This component of system search's the input query from buffer. When a query comes for processing new query handler passes the query to QM for verification either query is available in buffer or not. Here three cases may arise query is available in buffer; query is not available in buffer but is in QB and query is not available in QB.

In the *first case* match is available in the buffer and QM matches the user query with one of the queries available in buffer. If its match is found then, directly, query plan is accessed and executed. After execution WM increase the query weight by one.

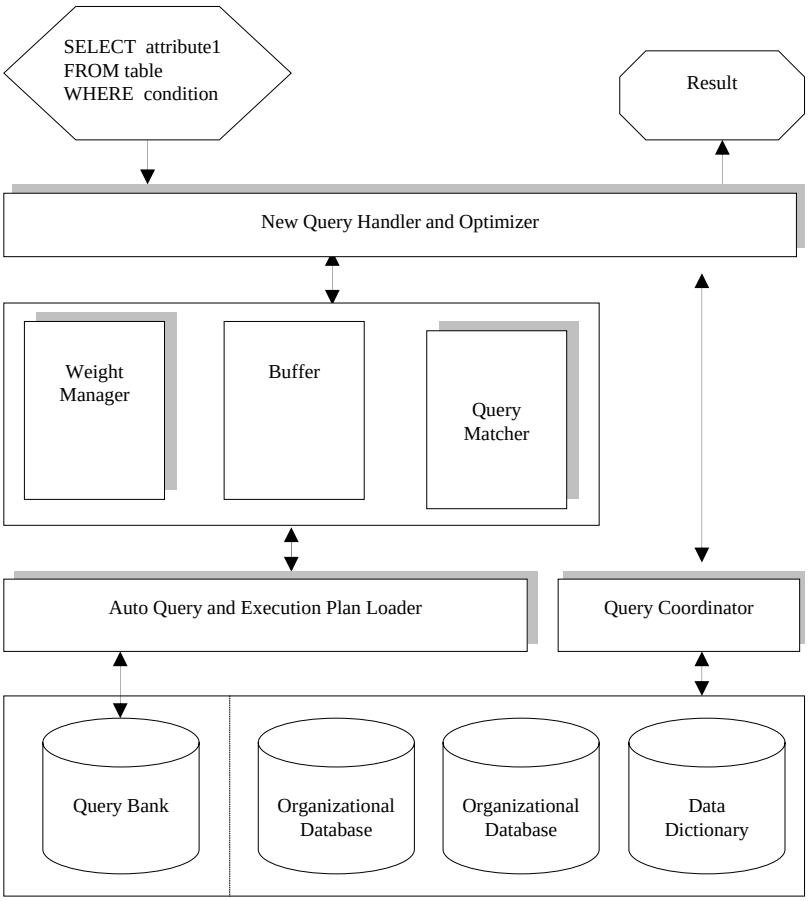

Fig. 1 System Architecture Diagram

In the *second case* match is available in buffer but is in the QB. The AQEPL swaps next queries of higher weight from QB into buffer according to buffer size and takes the already existing queries into the QB. This process is repeated until the QM finds the required query. When query is found its execution plan is accessed and processed. After execution the WM increases the weight of query by one and AQEPL refreshes the buffer, it takes the buffers queries into QB and loads the queries of highest weight from QB into buffer again.

In the *third case* match is not available in the QB. The NQHO creates all possible execution plans and execute the best one which is cost effective. After the execution of best plan the WM assigns weight one to the query and query with its executed plan is saved into QB and AQEPL once again refreshes the buffer, it takes the buffers queries into QB and loads the queries of highest weight from QB into buffer again.

C. Buffer

It temporarily holds the processed queries and their best plans of highest weight according to its size. The size of buffer is limited so limited number of queries is loaded in the buffer. When database application is closed all queries and their plans are moved to QB.

D. Weight Manager

The WM is responsible for assigns weight to processed queries. It increases the weight of query by one after processing the query and assigns weight one to query, which is executed first time.

E. Auto Query and Execution Plan Loader

It loads the queries with highest weights and their plans from query database into buffer according to buffer size. When QM searches query in buffer and fails to find the query then it loads the next queries and their plans of highest weight among the QB. When AQEPL is idle then it calculates the number of queries and their plans in advance according to buffer size.

F. Communication Manager

For the creation of new execution plans CM provides database statistics from data dictionary and for the processing of query it provides data from database also.

G. Query Bank

All executed queries and their best plans are stored in QB in descending order according to weight assigned by WM.

Example

Consider a bank database application with two users A and B has their accounts in the bank. They deposit and draw cash from bank and a query is executed for the updation of their account balance. User A deposits and/or draws cash monthly where as user B is a frequent bank user who deposits and/or draws cash daily. For the first month user A's query is executed twice in a month for the updation of his account and WM assigns weight after every execution which will be two after a month. In the case of user B, the query is executed twice in a day and sixty times in a month; hence, the WM assigns weight sixty to his query. The weight assigned to B's query is greater than that of A's query, therefore the availability of user B's query along with its execution plan in buffer is higher than user A. Whenever user B's query is executed his match will be in the buffer and their will be no need to create execution plans for such a query. Only the available matched plan will be executed with modification in the debit or credit amount. For user A, the match was not in the buffer because of low weight, hence in such case AQEPL will load the query after few swaps into the buffer. After the first time execution of queries for users A and B, there will be no need for the creation of execution plans. Once execution plan is created then, whenever that query come for execution its execution plan is directly accessed and executed. This will save time of execution plan creation and will be faster than the traditional ones.

III. Related Work

Georgia Kouttrika and Yannis Ionnidis [3] discussed query optimization by using rule base technique. In this technique, for each user profile is created and stored along with the weight assigned queries. It's a good approach however, the domain is limited to digital library and two users may have the same profiles. This will cause a redundant storage of user profiles.

Hristidis et al. [4] developed a prototype system called PREFER for efficient execution of multi-parametric ranked queries. Their focus is at query level for a single relation where user preference is assigned to each attribute of the query. Query returns result according to the weighted preference function. This approach also creates execution plans every time for a preference query. These preference queries and their optimized execution plans are not stored. Their prototype system may benefit from our proposed system if these preference queries and relevant execution plans are stored in the database and loaded into the buffer when database application starts first time. It will further speed up the query execution.

Our proposed system is independent of any specific database application, query, relation; we assign weight to the queries without confining it to any user, which will facilitate all the users using the application.

The proposed system efficiently stores the executed queries and their plans in QB for future use and these are loaded into the buffer with the start of the database application. We hope a stage will come when almost all queries will have their matches in the QB and then there will be no need for creation of new execution plan. The only fear we have is that if a query comes for execution to find its match we have to replace buffer many times with queries from QB, and may be the match is not in the QB. It will happen during early usage of the database application and will be gradually eliminated as the queries in the QB increases. The price of memory is decreasing drastically that will allow us to increase the size of buffer to accommodate the more queries and their plans. Hence the chances of match for required query will be increased.

IV. Conclusion

The query optimization is the need of every database. System configuration and resource availability may change during the long evaluation period of execution plans. As a result, queries are often evaluated with sub-optimal plan configurations. To remedy this situation, we have proposed a buffer based query optimization technique. In this technique once optimal execution plan is selected for a query then there is no need to create any execution plan for similar query in future. In future our focus will be the implementation of the system on the GRID technology, where data retrieval is a challenging issue.

Acknowledgment

Authors wish to appreciate the University for providing research opportunity.

References

[1] K. Shim, T. Sellis and D. Nau, Improvements on a heuristic algorithm for multiple-query optimization, Data and Knowledge Engineering, 12, 1994.

[2] Yannis E. Ioannidis, Query Optimization. ACM Computer Surv. 28(1): 121-123 (1996).

[3] Georgia Kouttrika and Yannis Ionnidis, Ruled-based query personalization in digital libraries, International journal digital library, 4: 60-63 (2004).

[4] Hristidis V, Koudas N, Papakonstantinou Y, Prefer: a system for the efficient execution of multiparametric ranked. Proceedings of the ACM SIGMOD international conference on management of data, Santa Barbara, CA, 21-24 May 2001. pp 259-270.