IJCSE International Journal of Computer Sciences & Engineering Open Access
Research Paper Volume-3, Issue-7 E-ISSN: 2347-2693

Index Selection for in-Memory Databases

Pratham L. Bajaj[1*] and Archana Ghotkar[2]

[1*,2] *Dept. of Computer Engineering, Pune University, India,*

www.ijcseonline.org

Received: Jun/09/2015 Revised: Jun/28/2015 Accepted: July/18/2015 Published: July/30/ 2015

***Abstract*—**Index recommendation is an active research area in query performance tuning and optimization. Designing efficient indexes is paramount to achieving good database and application performance. In association with database engine, index recommendation technique need to adopt for optimal results. Different searching methods used for Index Selection Problem (ISP) on various databases and resemble knapsack problem and traveling salesman. The query optimizer reliably chooses the most effective indexes in the vast majority of cases. Loss function calculated for every column and probability is assign to column. Experimental results presented to evidence of our contributions.

***Keywords*—** *Query Performance, NPH Analysis, Index Selection Problem*

I. INTRODUCTION

Relational Database System is very complex and it is continuously evolving from last thirty years. To fetch data from millions of dataset is time consuming job and new searching technologies need to develop. Query performance tuning and optimization was an area of interest for many researchers. Query performance can be optimize either by query reformation or index recommendation. In Query reformation, compound queries (nested queries) are converted into simple queries and join operations are restructure to reduce computational cost. Database Engine performs query reformation by converting one operator into other. For example, BETWEEN operator is converted into IN operator, if number of values in IN operator are less. OR operator is try to convert into AND operator because both predicates are evaluated separately that means two times table scan need to do. In general, database engine suggest indexes from candidate keys. But it may not be considered in user query, so complete table scan is perform. And also there is cost associated with index table maintenance so number of indexes and type of data structures are need to choose optimally.

Initially databases are solely stored in disk but due to fast data requirement and by considering limitations of disk, in-memory databases are evolved. Searching specific set of tuples from millions of tuples which satisfies input query is difficult. Automatic Index recommendation in databases will reduce overall query execution cost. CRUD operations may force to change indexing structure so selection of indexes should be done carefully. Also there is maintenance cost associated with indexing. Number of index selection is depend upon application requirement. For experimentation, in-memory databases are used. SQL like query language used from user queries.

II. RELATED WORK

Database tables contains millions of records and indexes helps to reduce overall searching time. Researchers provides rules for index selections, data structures and materialized views. Physical ordering of tuples may vary from index table. In cluster indexing, actual physical design get change. If particular column is hits frequently then clustering index drastically improves performance. In in-memory databases, memory is divided among databases and indexes. Different data structures are used for indexes like hash, tree, bitmap, etc. Collision avoidance should be considered in hash indexes. Indexes allocates some space, so it is important to select indexes which will yield profit and leads to less overhead for index maintenance. In composite indexes, order of columns have major impact on complete system performance. Compound indexes used only when particular set of columns are hits simultaneously in query. For example, supposed column A and column B are considered as compound indexes AB. If queries get fired on column A or on column A and B then compound index AB gives optimal results. If predicate in queries fired on column B then compound index is not used for user query. Tuples retrieval from millions of records is difficult so optimized searching methods are used. Generally indexing is perform on candidate keys and selected columns contributes in query execution [1]. Index Selection Problem is NP-Hard problem. If there are N columns then number of indexes are power set of N i.e. 2^N. Indexes required some maintenance time and allocates memory. If insert and update queries hit frequently then overhead is more. Researchers found ISP resembles Knapsack problem and knapsack. Knapsack allows to select indexes which provides maximum profit to user queries. Genetic algorithm is stochastic search, use when search space is large and space increasing exponentially. It is highly recommended when to deal with non-linear optimization. It follows Darwin's "survival of fittest" model. Following table discussed different searching techniques for ISP.

Select type of index that best fits application specification. Index characteristics include the following:

- Clustered versus nonclustered

© 2015, IJCSE All Rights Reserved

- Unique versus nonunique
- Single column versus multicolumn

Wearable devices and sensors generating data so frequently that fast searching algorithms need to derive. With evolving databases and changing user requirements, new query languages comes like NoSQL and NewSQL were developed.

G. Valentin et. al. [2] considered variant of knapsack problem and suggested best index configuration for query. Smart column enumeration suggests indexes based on user queries. Solution provided is globally optimal. DB2 work as black box index recommendation engine. Various operator combination form within query and remove duplicates. Also formulated number of indexes possible for given databases.

H. Gupta et. al. [3] provide first model for automatic index selection by summary table. They rely on precompute data so that in future queries will give optimal result. Also provide all possible set of solution for view, queries and indexes. Chances of optimal result is very less.

S. Chaudhari et. al [5] depicted autoAdmin model for physical design and provided what-if interface method. Very well explanation of materialize view for index selection. Considered multicolumn indexes with enumeration of single column indexes. Explained challenges in physical design changes, scaling challenges and how it changes overall system maintenance cost. [11] Focused on exploiting feedback from query execution and query progress estimation. Specially focused on merging and reduction in number of calls from optimizer.

P. Kolackow et. al. [6] solve ISP with genetic algorithm by considering Darwin's survival of fittest model. Genetic algorithm (GA) does not guarantee of optimal solution but converges solution space in one phase. Focus on query execution cost by index recommendation. GA refer chromosomes model by adopting mutation and crossover [4].

S. Chaudhuri [8] resembles ISP with variation of knapsack problem and approach lies in Linear Programming. Experiments done by considering materialize view. Highlighted two sub problems: picking right set of cluster indexes and non-clustered indexes and referred as hard to approximate. Explained workload with extensive updates.

S. Chaudhuri et. al. [10][13] number of index set evaluated by considering both query syntax and cost information. Find "goodness" for each index set. Iterative complexity analysis done for multicolumn indexes. Depicted ISP as difficult search problem. Efficiency measure on (1) Number of indexes (2) Enumerated number of index selection configuration (3) Number of optimizer calls for each enumeration configuration and reduce it by atomic configuration plan.

F. Fotouhi et. al. [12] ISP resemble Uncapacitated Facility Location Problem (UELP). Derived cost function by computing ISP by Genetic Algorithm (GA). Objective function explained with time complexity and binary encoding.

One can summarized from above literatures, ISP can be solve either by linear programming or by non-linear i.e. GA. If rate of growth of data is high then it better to go with GA.

III. METHODOLOGY

The paper discussed complete implementation of query performance optimization by index recommendation. Goals and objectives are

- To recommend set of indexes that can minimizes query execution cost.
- To calculate query cost estimation.
- To perform Space analysis.
- To perform Query analysis.
- To calculate probability of every column.
- To validate recommended indexes.

System contains database, system block diagram and detail description of algorithms used in each phase. High dimensional database i.e number of columns more than 7 and having different data types were considered for implementation. Also database contains millions of rows.
Another Tool For Language Recognition (ANTLR) is parser generator that uses LL(*) parsing. LL(1) is top down parser and lookahead one symbol. ANTLR takes an input a grammar that specific language and generate output as source code. It generate lexer, parser, tree parser and combine lexer-parser. Parser then generate Abstract Syntax Tree (AST). Following figure shows syntax tree for grammar.

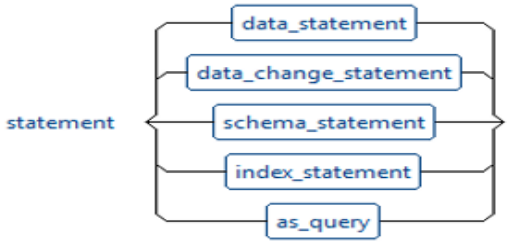

Fig 1. Syntax Tree

The major steps involved in the proposed system are:
1. Database analysis
2. Query analysis
3. Statistical analysis
4. Index recommendation
5. Query execution cost

In proposed work, index recommendation done dynamically by analyzing both user inputs and databases. Generally, indexes are recommended by database engine from candidate keys but chances of utilization are rare. So indexes should be dynamically assign to databases.
In database analysis, cardinality of each column is find out. At this moment, cardinality is find out by perfect matching. It can be further evolve iteratively by analyzing

© 2015, IJCSE All Rights Reserved

what value fire by user in predicate. If number of cardinality greater than threshold then that space is drop. In experiment, threshold value considered 60%. Cardinality distribution is calculated by checking cardinality count between multiple of 0.5 to ratio of total number of rows to cardinality and multiple of 1.5 to ratio of total number of rows to cardinality n query analysis.

Predicates were again separated into column name and operator used on it. For equality and range operator separate count value is maintain. More the count better to select respective data structure i.e Hash or Tree. In special case, if particular column with range operator count more than k times of equality operator then in that case tree data structure preferred over hash.

In statistical analysis, two errors are calculated for optimal utilization of resources. Error1 is columns not indexed but used by query i.e. wastage of time, database engine go for complete table scan. Error2 is columns are indexed but not used by query i.e. wastage of memory. Both error1 and error2 varies from 0 to 1. Depend upon application requirement weightage is assign to error and payoff factor is calculated. Form error1 and error2 payoff factor is calculated by eq. (1) and then columns were arranged.

$$\text{Payoff}=k^* \text{ Error1}-(1-k)^* \text{ Error2} \quad (1)$$

```
Item is:id with value:0.5833334
Item is:FirstName with value:0.6111111
Item is:LastName with value:0.7222222
Item is:phone with value:0.7777778
Item is:city with value:0.7777778
Item is:email with value:0.9166667
Item is:address with value:0.9722222
Item is:dob with value:0.9722222
```

Fig 2. Column with error count

Probability value is set for every column and it is resembles with knapsack problem. Probability of columns can be calculated by

$$PB(C) = \frac{PB(Ci)*P(E)}{\sum_{k=0}^{n} PB(Ck)*P(E)} \quad (2)$$

Attribute	Probability
city	0.13333334
dob	0.016666668
phone	0.13333334
email	0.050000004
id	0.25
FirstName	0.23333335
LastName	0.16666667
address	0.016666668

Fig 3. Column Probability

Depending upon application requirement number of indexes are set by altering physical design. Indexes can be single column or multicolumn. More the number of indexes more will be the maintenance cost. Proposed system recommend indexes with its data structure. For single column indexes, columns with highest probability of occurrences is selected first. The point is what data structure is used if both equality and range operators applied on column. Hash indexes are faster than tree but in insert and update operations tree gives better results. In following figure column names are listed out with occurrences of errors.

IndexColumn	IndexType
FirstName	Range
dob	HASH
phone	Hash
LastName	Range
id	Range

Fig 4. Recommended Indexes

IV. EXPERIMENTAL RESULTS

The result of different modules are shown below with single column and two column indexes. Figure 1 shows query performance optimization by single column indexes. Here we can see difference between query execution time before indexing and after indexing. Number of queries fired were 540 and most of them is selection queries. If the ratio of insert or update to select get change then final results will also change.

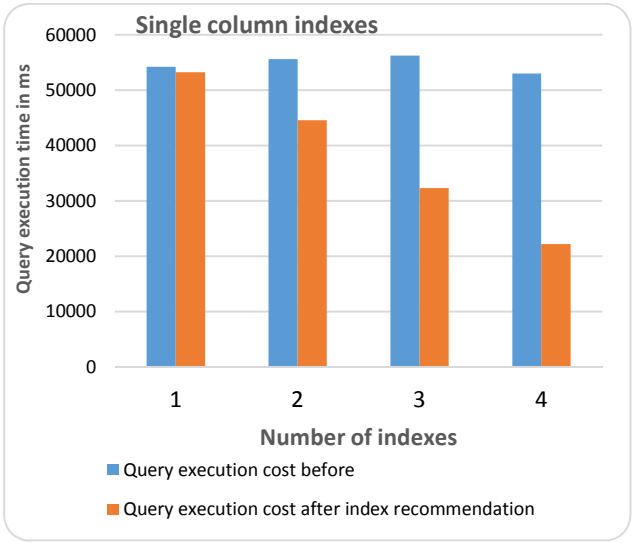

Fig 4 Query execution time with one column indexes

Figure 2 show query performance optimization by two column indexes obtained from by calculating correlational factor between two columns.

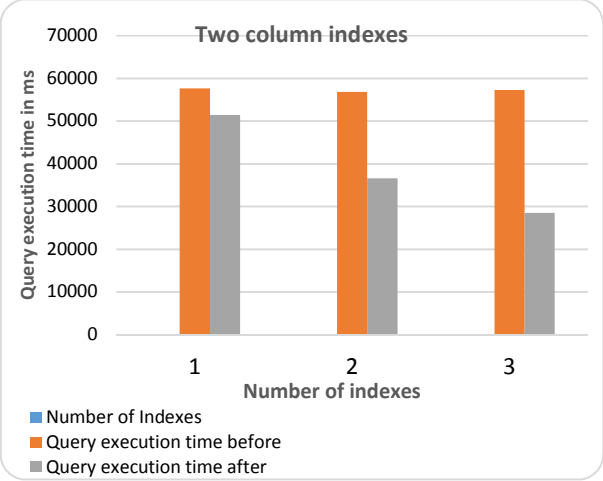

Fig 5 Query execution time with two column indexes

Figure 3 show query execution cost suggesting one and two column indexes.

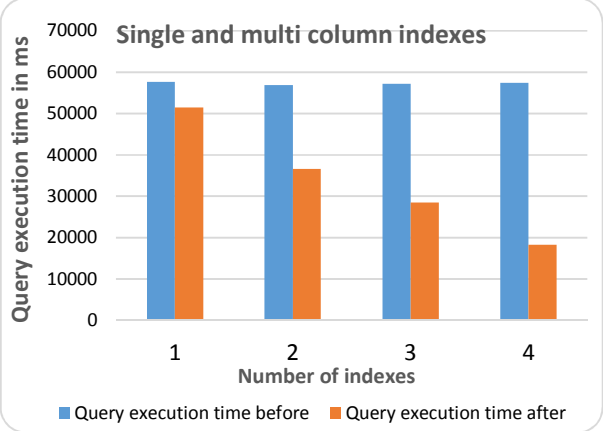

Fig 6 Query execution time with both single and multicolumn indexes

V. CONCLUSION

Query performance optimization by Index Recommendation is proposed for in memory database. Index recommendation problem solved by resembles knapsack problem. Proposed algorithm considered both user query and database for index recommendation and in results it was found that appropriate indexes reduces overall query execution time. In experimental results, comparison done between single column and multicolumn indexes and verified query execution time. The same index recommendation tool can be used with other databases and enhances overall system performance.

ACKNOWLEDGMENET

I would like to take this opportunity to thank Nikhil Tamhankar, Abhay Chavan and Ramaswami Raju for always there for guidance and support.

REFERENCES

[1] Oracle "Performance Tuning Guide 11g Release 2".
[2] G. Valentin, M. Zuliani, D. C. Zilio, A. Skelley and G. Lohman, "DB2 Advisor: An Optimizer Smart Enough to Recommend Its Own Indexes", ICDE, **2000.**
[3] H. Gupta, V. Harinarayan and A Rajaraman, "Index Selection for OLAP," in IEEE, **1997**, pp **208-219**.
[4] J. Calle, Y. Saez and D. Cuadra, "An Evolutionary Approach to the Index Selection Problem," in *IEEE*, pp **485-490**, **2011**.
[5] S. Chaudhuri and V. Narasayya, "Self-Tuning Database Systems: A Decade of Progress," *in VLDB Endowment,* pp **3-14**, **2007**.
[6] P. Kolaczkowski and Henryk Rybinski, "Automatic Index Selection in RDBMS by Exploring Query Execution Plan Space," in IEEE, pp **131-137**, **2005**.
[7] P. Papadomanolakis and S. Ailamaki, "An integer linear programming approach to database design," in *Workshop on Self-Managing Database Systems,* pp **442-449**, **2007**.
[8] S. Chaudhuri and V. Narasayya, "An Efficient, Cost-Driven Index Selection Tool for Microsoft SQL Server," *Proc. 23rd Int'l Conf. Very Large Databases (VLDB)*, pp. **146-155, 1997**.
[9] C. S. Blanken and H.M.Chang, "Index selection in relational databases," in International Conference on Computing and Information, pp. **491–496**, **1993**.
[10] S. Chaudhuri, M. Datar and V. Narasayya "Index Selection for Databases: A Hardness Study and a Principled Heuristic Solution" in IEEE transactions on knowledge and data engineering, Vol. 16, pp **1313-1323**, **2004**.
[11] S. Chaudhuri and V. Narasayya, "AutoAdmin 'What-If' Index Analysis Utility," in *Proc. ACM SIGMOD*, pp **367-378**, **1998**.
[12] F. Fotouhi and C. Galarce, "Genetic Algorithms and the Search for Optimal Database Index Selection" *Springer,* pp **249-255**, **1991**.
[13] S. Agrawal, S. Chaudhuri and S. Narasayya "Automated selection of materialized view and indexes for SQL databases" in *Proc 26th VLDB*, **pp 496-505**, **2000**.
[14] S. Agrawal, S. Chaudhuri, L. Kollar, A. Marathe, V. R..Narasayya, and M. Syamala,, "Database Tuning Advisor for Microsoft SQL Server 2005," In Procs. 30th VLDB Conference, pp. **1110-1121**, **2004**.

© 2015, IJCSE All Rights Reserved