A Survey on Query Performance Optimization by Index Recommendation

Pratham L. Bajaj
PICT, Pune

ABSTRACT
Query language access data from databases. With exponential growth of data, optimization techniques need to be adopt for better results. Query performance tuning and optimization can be achieved by query reformation and index selection. Searching tuples from millions of results is overhead and it degrades overall system performance. To reduce searching time is goal of index recommendation. Index Selection Problem (ISP) is optimization problem. This is NPH problem and it can be solve by different approaches like greedy approach, dynamic programming, linear programming, branch and bound, genetic algorithm, etc. In general, indexing is done on candidate keys but it will not give assurance of optimal solution. Researchers tried to resemble ISP with knapsack problem and variation of it. Different data structure are used for indexing like tree, hash, bitmap, etc. In composite column indexes, order of columns affects overall performance In-memory databases are fast databases and new data structures to be suggest for indexing. Usually indexing is done on only columns which will yield profit in query execution. Join operations executions are discussed briefly.

General Terms
Database performance optimization and tuning

Keywords
Index selection, knapsack problem, genetic algorithm.

1. INTRODUCTION
Relational Database System is very complex and continuously evolving from last thirty years. Query performance tuning and optimization is area of interest for most of researchers. Query performance optimization can be done by (i) Query reformation (ii) Index Recommendation. Real time applications required fast response so databases are shifted from disk to in-memory. In general, in-memory databases are in de-normalize form means and there are no join operations. Here SQL like query language is considered and there is no nested query. Searching and sorting are two classical problems of computer science. Database engine spend most of time in searching of tuples from millions of tuples. Paper focus on different indexes and indexing techniques.

In In-memory databases spaces are limited and indexes takes some space. To reduce space complexity appropriate selection of indexes is done. In general, database engine suggest candidate key and that can be used as index. Databases deals with range and equal query, tree is used for range query and hash used for equal query. Hash is much faster than tree and provide result in unit amount of time. Time and space are both important parameters, so index selection should done properly without overhead. Query cost estimation plan helps to judge selected index. There are two different errors which can affect system efficiency of system.(1) Error1- Query not utilizing available index (Indexes are not used by predicates in query) (ii)Error2- Indexing not done which satisfy predicates from query. If indexing is not perform properly then whole table scan to be done and it is time consuming job. Index Selection Problem (ISP) is NPH problem. Database administrator cannot performing on all available candidate keys because there is cost associated with index maintenance.

Physical ordering of tuples in tables may affect may affect performance plan, so there is concept of cluster indexes. It force to keep ordering of tuples are same as ordering in index. Researchers suggested different approaches for Index selection like Trial and error approach, some resembles ISP with knapsack problem.

2. QUERY LANGUAGE
Query language need to mention what to do, not necessary how to do and databases system will take all care by database system. Query grammar should be error free so that it parsing result is error free.

It is structural language which allows user to give input as what to do without any procedure. Query execution done in three stages. In first stage, query syntax checking and symbol table is creation is done by query parser. Tokens are separated and compare with standard keywords. Tree like structure is form for query parsing. Operator tree is output from parser. ANTLR grammar is used for query conversion. It is bottom up parser and gives results as early as possible. Once data is parser and validate then it's time for syntactic error checking. Logical errors are removed in this stage and query is forwarded for optimization purpose. In this stage reformation of query is done. Generalized structure of query is shown below:

Select [distinct] <Column_name(s)/*> from <Table_Name> [where <Condition>

Having < Condition>

Order by <Condition>

Group by <Condition>]

Generally operands are at leaf and operator is at parent node, result of it is further computed in bottom up manner. Query reformation is part of query optimization. Nested queries are transform into simple query, it saves recursive scanning time. Clauses deals with conditions so most frequently occurring queries are to be index. Optimize query is represented in intermediate stage.

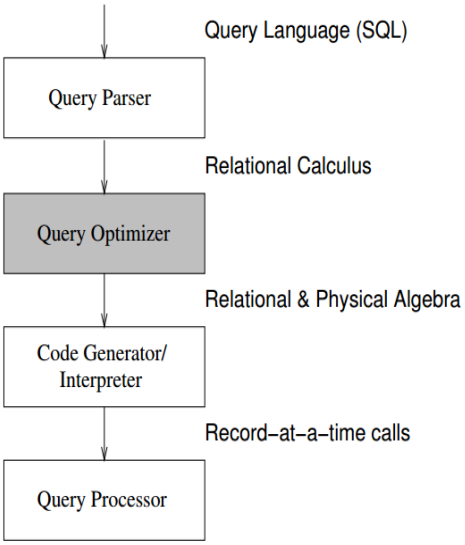

Fig. 1 Query flow of DBMS [2]

Database query performs select, update, insert and delete operation. Index data structure affects insertion, update and delete operation.

In tradition databases, structure data is store in fixed schema but it is not suitable for documents, blob type data. So SQL query modified into NoSQL query. NoSQL stands for not only SQL. NoSQL allows to store data in schema less and it is more flexible. It is not using relational model. By CAP theorem consistency (C), availability (A) and partition tolerance (P) in distributed system it is possible to achieve only two features Consistency and Availability or Consistency and Fault tolerance. There are different types of NoSQL databases. It improves program productivity. It further classified in following:

1) Key-Value- Best for session data, profiles and also allows to store multiple keys. E.g. Riak, Redis, Memcached.

2) Documents- Best for content management, web analytics, e-commerce. E.g. MongoDB, Terrastore, RavenDB.

3) Column family- Best for content management and different row may have different set of columns. Helps in maintain heavy log- Cassandra, Hypertable.

4) Graph- Best for connected data just like social network, routing information.

Join operation is most time consuming operator in overall query execution life time. Left deep and Right deep should be handle carefully so that overall tuples scans can be minimize. Generally, evaluation of join operations are considered as NP Complete problem and there are different strategies are [13]

1) Bottom up optimization- Execution start from based relation and step by step execution is done.
2) Top down optimization- Uses divide and conquer, each part is optimize separately and after aggregation again optimization is perform.
3) Transformation- Transform complex execution plan to other simpler plan.

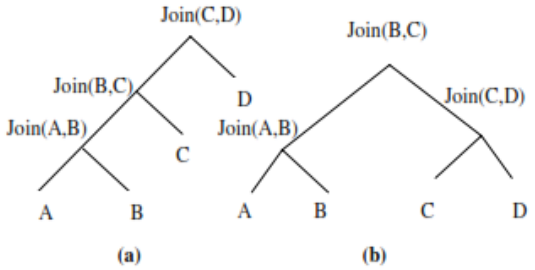

Fig 2 (a) Left deep (b) bushy tree [13]

2.1 Deterministic Algorithms

Deterministic search solutions space and accepts different approaches.

2.1.1 Dynamic Programming

This approach first is suggested by IBM by complete searching of solution space. Generally complexity goes in $O(n^3)$. It grows exponentially in join relations.

i. Selection Projection Heuristic
ii. Cartesian product heuristic

2.1.2 Iterative Dynamic Programming

Combination of classical Dynamic programming and greedy approach.

2.1.3 Minimum Selectivity Heuristic

Heuristic construction of left deep tree.

2.1.4 IK Algorithm

Takes advantage if nested loop cost and find optimal left deep join.

2.1.5 Relational Difference Calculus

Finds most influence relation in join expression. [13]

2.2 Randomized Algorithms

Overcome on classical deterministic algorithms and developed non-deterministic approaches.

2.2.1 Random Walk Algorithm

Quality of algorithm is completely depends upon ratio of good and bad solution in solution space.

2.2.2 Iterative Improvement Algorithm

Simple approach similar to hill climbing just like greedy search strategy. It can apply iteratively.

2.2.3 Simulated Annealing Algorithm

Improvement over above approach. Less chances to trap in poor local minimum.

2.2.4 Two Phase Optimization

It is combination of above two models. Iterative covers large portion of solution space and simulated annealing good to search point of neighborhood. [13]

2.3 Genetic Algorithms

Genetic algorithm is derived from survival of fittest model and are applied to complex problems.

2.3.1 Coding

i. Simple left deep tree coding
ii. Travelling salesman coding
iii. Bushy Tree coding

2.3.2 Selection

i. Roulette Selection

ii. Rank Selection

iii. Adaptive Selection

2.3.3 Crossover

i. Subsequence exchange
ii. Subset exchange crossover
iii. Order crossover

2.3.4 Mutation

i. Reciprocal exchange
ii. Exhaustive[10]

3. INDEXES

Indexing is depend upon physical organization of tuples. Generally indexing done in following situations

- Frequently access query columns
- Unique key constraints on column

Best index selection can be done by analyzing query execution plan for input query. Limit number of indexes for given relation. Most frequent updating table should not index heavily, it degrades overall performance. Heuristic steps for index selection are shown below:

1) Analyze columns that are used in predicate frequently.

2) Initially go for single column indexes and if it is necessary then go for composite column indexes.

3) Order of indexes in composite indexes are to be chosen carefully.

4) If both range and equality condition are coming then go for tree indexes only instead of maintaining separate hash and tree indexes.

5) Selection of appropriate data structure depends upon predicates, insert, update, delete operations. [12]

3.1 Clustered Index

It force to arrange index order in same way as physical design. There are almost one cluster index per table. It gives good result many tuples are fetch having same type.

Bitmap Index- Indexes are represented by set of bits and if column consider in indexing it bit is 1 otherwise 0. [5]

3.2 Multilevel Index

When number of tuples are more and available space for indexing is less then multilevel index used. It improves maintenance cost of system but update in indexing takes more time than one level indexing. [5]

3.3 Hash

It gives result in unit time. Bucket is basic storage unit.

3.3.1 Static hashing

Key-value may search to one bucket and leads to sequential search. Bucket size and key-value pair should remain in uniform distribution.[5]

3.3.2 Dynamic hashing

Size of bucket is not uniform and it accommodate size that is shrinking and expansion depends on database. Extendable hashing is form of it. [5]

3.4 Tree

B tree is unbiased structure in which date is store at both intermediate and leaf node. Searching is quite difficult than B+ tree. Insertion and deletion are more complicate than B+ tree. Implementation harder than B+ tree.

B+ Tree alternative to indexed-sequential files and it automatically organized itself in small local space. It provides high fanout or low depth. Extra insertion or deletion will lead to overhead. Length of leaf nodes to root are same means all leaf nodes are remain at same level. Leaf node has between [(n-1)/2] to (n-1) values. Leaf nodes have all key elements and sequential scanning possible without moving back to parent node. Searching is more efficient in B+ tree as compare to B tree. Non-leaf nodes has between [n/2] to n children.

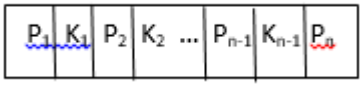

Fig 3 Tree node structure

Here.

P –Pointer to children

K- Key $K_1<K_2<K_3<\ldots<K_{n-1}$

RB tree is also store data in unbiased form so that it is easy for cost estimation. Root is always color black and two blacks nodes can be remain neighbors but not red nodes. It allows better insertion and deletion operation. Tree data structure is used when there is high cardinality means there are different values for particular field. [5]

3.5 Bitmap

It allows fast read by maintain structure in 0 and 1. Structure makes it possible for the system to combine multiple indexes together for fast access to the underlying table.

1) For columns with very few unique values (low cardinality)
2) Tables that have no or little insert/update are good candidates (static data in warehouse)
3) Stream of bits: each bit relates to a column value in a single row of table.

A modification to a bitmap index requires a great deal more work on behalf of the system than a modification to a b-tree index. In addition, the concurrency for modifications on bitmap indexes is dreadful.[5]

For example,
Select * from Table where a>100 AND b<500 AND c=0

In this query, where clause contain three predicate and all are having different arithmetic operator. The order of execution of those predicates affects overall system execution cost. Query will get best result when execution start from columns having high cardinality and equality condition. If c=0 is executed first then next can be go for columns having less number of records.

Initially histogram based approach is used for cost estimation. Range of histogram depends upon number of fields with distribution metrics. Suppose 1, 2, 3, 4, 5, 6, 7, 8 are set of histogram and sets 1 to 4 having values in million and sets 5 to 8 having values in hundred. In this case, data set is more error prone.

4. OPERATOR OPTMIZATION

Query reformation affect overall index selection scheme for query execution.

1) OR operator executes either side as separate query independent query and n times of single predicate, where n is number of predicates having OR operator. It is optimize by IN operator.
2) LIKE operator is most difficult to optimize because of unexpected input from user. % is used for unknown set of characters.
3) BETWEEN operator can be optimize by combination of arithmetic operator and AND operator.[12] For example,

Select * from tab_name where col_name BETWEEN expr1 AND expr2
Optimize to Select * from tab_name where col_name>expr1 AND col_name<expr2

5. RELATED WORK

Ioannidis [2] considered optimization problem as combinatorial optimization problem. Discussed Simulated Annealing (SA) and Iterative Improvement in replacement with exhaustive search. This is two phase optimization and discuss under randomized algorithm. Iterative improvement achieves local optimization which is downhill movement. SA accepts some uphill movement by considering probability factor. In two phase optimization, local optimization run for some time and then uphill to find local minimum. Join operations are discussed with left deep and right deep. Cost function is based on following assumption (i) no pipelining i.e. intermediate result, (ii) minimum buffering (iii) no duplication elimination (iv) on-the-fly execution of projection.

Abdekadar et al. [3] discussed query optimization methods for uniprocessor relational databases to data grid system through parallel, distributed systems. Compare optimization methods achieved by (i) Size of search space (ii) Static or dynamic methods (iii) Re-optimization and re-scheduling execution plans (iv) Intra-operator and inter-operator level of modification (v) centralized or decentralized control.

Molina [4] discussed in-memory databases with data representation, query processing, access method, performance, application programming interface and protection, data clustering and migration.

Gupta at. al. [5] suggest limitations of trial and error approach, it divide space between tables and indexes. Discussed space and time complexity for maintenance of indexes. Greedy approach used for subcube selection and explain all possible indexes with m attribute.

$$\sum_{r=0}^{m} \binom{\mathrm{m}}{r} r!$$

With n-dimensional data cubes associated with

(1)2n view

(ii) 3n slice queries

(iii) About 3n! possible indexes and 2n! are fat indexes

Materialize view versus views in associated indexes.

S. Chaudhuri et. al [6] discussed cost driven index selection tool and provided special attention to handle multi-column index complexity. Index goodness evaluated on basis of query syntax and index cost statistics. Follows iterative model in first iteration consider one column index, second iteration two column and so on. Efficiency measures on (i) Number of indexes considered (ii) Number of reformation of indexes i.e. enumeration. Calls between optimizer, enumeration and server is overhead so proposes atomic configuration approach. From set of M configuration set M' sets are chooses by greedy approach. Cost (Q, C) estimation done on

Javier et. al [1] an evolutionary algorithm applied on physical database design for ISP. Discussed problems (i) updates in physical design (workloads) by using logging capabilities. (ii) Resolve set of candidate indexes. Highlights ISP in dynamic environment like materialize views. ISP is a optimization problem and its goal is minimization of cost. Genetic algorithm (GA) evolved from the theory of evolution and it is stochastic search. At initial set of bit sequences are considered as chromosomes and population is all possible set of indexes. Reproduction of indexes are done with genetic operator, crossover and mutation. Reasons to go with GA are (i) To deal with large space (ii) Non-linear optimization problem. Success of GA is totally depends upon fitness function. M is index configuration and number of indexes are

$$\mathrm{M}=\sum_{i-1}^{nCols} (\mathrm{C}\ (\mathrm{i}, \mathrm{nCols})\ * \mathrm{nIndexType})$$

Where,

$$\mathrm{C(I,\ nCols)}=\frac{nCols\,!}{(n-p)!*p!}$$

It provides experimental results with genetic algorithm approach.

Papadomanolakis et. al. [7] Heuristic approach are well suited for large data space but they are hard to analyses and compute. Linear programming guarantee for optimal solution with combinatorial analysis. Linear programming are used for index selection and later branch and bound are used for quality checking if index. Index selection is definition is just like atomic configuration [6]

Gupta et. al [8] it gives algorithm to automate selection of summary table and indexes. It provides first Index selection Plan based on Trial and Error approach to find optimal plan. Combinatorial analysis of views, queries and indexes. Provides 1 step index selection. By aggressive pruning may remove optimal solution. Complex system due to consideration of pre-compute data.

Calle et. al [1] Not only consider gain of index configuration but also generate best configuration for query. Consider as variant of knapsack problem. Not limited to locally optimal solution Recommendation of indexes based on query. Smart column enumeration for index scan. Reducing optimizer call by placing enumeration algorithm inside optimizer. No concept of partitioning in parallel databases. To guarantee of quality work No analysis of hardness study. Indexing on materialized view in optimizer. No mass query optimization.

Chaudhuri et. al. [11] use of heuristic approach to find optimal configuration of indexes by:

(1) Removing spurious indexes by considering query and cost information. (2)Optimization of index set (goodness) (3) Iterative approach to manage multi column indexes. Generation of multi column indexes from single column. SQL statement is metric of goodness. Explain measures of efficiency of index selection tool. Not consider materialize view in optimizer operation. Number of optimizer invocation reduce by consider small knapsack with single column.

Kolackow et. al. [10] Converge global solution in one phase. Focus on minimizing cost of query execution plan and optimum utilization of indexes. Genetic algorithms are used. Reduce space for candidate solution. No guarantee of optimal solution. Experiments done without considering materialized view.

6. CONCLUSION

Query optimization by considering suitable physical design (selecting optimal set of indexes) for in-memory databases. One can solve index selection problem by utilization of genetic algorithm for in-memory databases. This heuristic approach does not guarantee of optimal solution but gives best results near to optimal solution. Researchers developed different searching and selecting strategies which can help to maximized objective function. Linear programming assures optimal solution by following branch and bound. Best index selection will gives fast results and also helps for future coming query.

Future scope of this project is to find cost execution plan of particular query based on the number of tuples satisfy given predicate. And solve predicates based on minimum tuples return by it. Also Index recommendation for LIKE operator is not cover in this paper.

7. ACKNOWLEDGMENTS

I would like to express my gratitude to Firat Kart for the useful comments, remarks and engagement through the learning process of this topic. Furthermore, I would like to thank Nikhil Tamhankar and Abhay Chavan for introducing me to the topic as well for the support on the way.

8. REFERENCES

[1] J. Calle, Y. Sáez and D. Cuadra,"An Evolutionary Approach to the Index Selection Problem," Nature and Biologically Inspired Computing (NaBIC) IEEE 2011, pp. 485 - 490.

[2] Y. E. Ioannidis," Randomized Algorithms for optimizing large Join Queries," in SIGMOD '90 Proceedings of the 1990 ACM SIGMOD international conference on Management of data, pp 312-321.

[3] A. Hameurlain and F. Morvan, "Evolution of Query Optimization Methods", Springer Trans. on Large-Scale Data & Knowledge, pp. 211–242, 2009.

[4] H. Molina,"Main Memory Database Systems: An Overview," IEEE Transaction on Knowledge and data engineering, vol 4 No.6, December 1992.

[5] Oracle "Performance Tuning Guide 11g Release 2".

[6] S. Chaudhuri and V. Narasayya, "An Efficient, Cost-Driven Index Selection Tool for Microsoft SQL Server", Proceeding of the 23rd VLDB Conference, 1997, pp 146-155.

[7] S. Papadomanolakis and A, Ailamaki in "An Integer Linear Programming Approach to Database Design" in International Conference on Data Engineering Workshop, 2007, pp 442-449.

[8] H. Gupta, V. Harinarayan and A. Rajaraman, "Index Selection for OLAP", in International Conference on Data Engineering, 1997, pp 208-219.

[9] S. Chaudhuri, V. Narasayya, "Self-Tuning Database Systems: A Decade of Progress" in VLDB Endowment, ACM 978-1-59593-649 2009.

[10] P. Kolaczkowski, and H. Rybinski, "Automatic Index Selection in RDBMS by Exploring Query Execution Plan Space," Springer SCI 223, 2009, pp.3-24.

[11] S. Chaudhuri, "Index Selection for Databases: A Hardness Study and a Principled Heuristic Solution" in IEEE transactions on knowledge and data engineering, vol. 16, no. 11, 2004.

[12] G. Valentin, M. Zuliani, D. C. Zilio, A. Skelley and G. Lohman, "DB2 Advisor: An Optimizer Smart Enough to Recommend Its Own Indexes".

[13] S. Vellev "Review of Algorithms for the Join Ordering Problem in Databases Query Optimization".

[14] S. Chaudhuri "An Overview of Query Optimization in Relational Systems", in ACM SIGACT-SIGMOD-SIGART symposium on Principles of database systems, 1998, pp 34-43.