ISSN No. 0976-5697

Volume 4, No. 8, May-June 2013

International Journal of Advanced Research in Computer Science

REVIEW ARTICAL

Available Online at www.ijarcs.info

A Review: Vertical Partitioning Algorithms to Handle Dataload on Distributed Databases

Kamaljeet kaur*, Jaspreet kaur
Assist Prof(CSE Deptt)*, Student of M.Tech(CSE)
Sri Guru Granth Sahib World University
Fatehgarh Sahib,Punjab(India)
preetlotey89@gmail.com

Abstract: Relational databases are widely used in many applications to store data. But there are many problems with relational databases like scalability, handling real time data and handling unstructured data like data on web is not properly structured, it is semi structured or unstructured. To overcome these problems non-relational databases come in to existence. Non-relational databases are growing these days.. Non relational databases deals with the concept of partitioning to handle the different data load on distributed machines. Non relational databases deals with vertical partitioning method which is based on hash partitioning, range partitioning, list partitioning to handle the better data load into some extent. This paper deals with vertical partitioning method as vertical partitioning is applied in three contexts: a database stored on devices of a single type, a database stored in different memory levels, and a distributed database. In distributed databases, fragment allocation should maximize the amount of local transaction process. In this paper, we study on distributed databases and summarizes the problems of data fragmentation, allocation and replication in distributed database.

Keywords- Relational databases, vertical partitioning method, distributed database, fragment allocation, dataload.

I. INTRODUCTION

Non-relational databases are very popular and in use these days because of their various advantages over the relational databases like handle various types of data like key-value, column and document, semi structured and structure data. These databases can handle very large amount of data and also provide greater scalability that is why these are very useful to use in distributed environment like in cloud and grid computing applications.[1]

A partition is a division of a logical database or its constituting elements into distinct independent parts. Database partitioning is normally done for manageability, performance or availability reasons. Data Partitioning is also done using vertical partitioning. We can put different columns on different partitions[2].Partitioning of database is done on several ways: vertical, horizontal and mixed(hybrid). Vertical partitioning subdivides attributes into groups and assigns each group to a physical object. Horizontal partitioning subdivides object instances (tuples) into groups, all having the same attributes of the original object. We refer to the physical objects that are a result of vertical or horizontal partitioning as horizontal or vertical fragments. In this paper we study on various partitioning algorithms using vertical partitioning technique to partition the dataload on distributed machines. It overcomes most of the problems which are used in horizontal partitioning.

a. Efficient performance on aggregation queries (like COUNT, SUM, AVG, MIN, MAX)[4]
b. True scalability and fast data loading for Big Data .[5]
c. Provides hard disk access and reduce disk space .[4]
d. Improved Bandwidth Utilization.

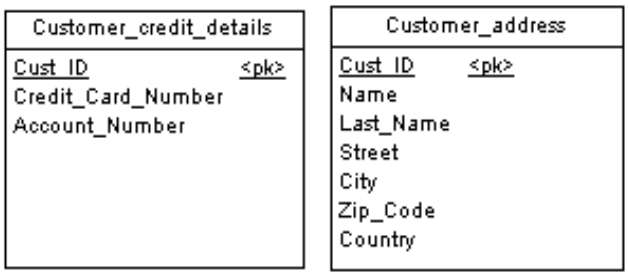

Figure-1 Vertical Partitioning[3]

A. Further vertical partitioning schemes is based on:-

a. Range Partitioning:- It is a partitioning technique where ranges of data is stored separately in different sub-tables. It maps data to partitions based on ranges of values of the partitioning key that is used for each partition. It is the most common type of partitioning and is often used with dates. For a table with a date column as the partitioning key, the janurary 2005 partition would contain rows with partitioning key values from 01-Jan-2005 to 31-Jan-2005. For example, splitting up sales transactions by what year they were created or assigning users to servers based on the first digit of their zip code. The main problem with this approach is that if the value whose range is used for partitioning isn't chosen carefully then the scheme leads to unbalanced servers.[2]

b. Hash Partitioning:- With this approach, each entity has a value that can be used as input into a hash function whose output is used to determine which database server to use. This is typically used where ranges aren't appropriate, i.e. employee number, productID.[1] Hash partitioning maps data to partitions based on a hashing algorithm that applies to the partitioning key that you identify. The hashing

algorithm evenly distributes rows among partitions, giving partitions approximately the same size. For example- if you have ten database servers and your user IDs were a numeric value that was incremented by 1 each time a new user is added. In this example, the hash function is performed on the user ID with the number ten and then pick a database server based on the remainder value. This approach should ensure a uniform allocation of data to each server. The key problem with this approach is that it effectively fixes your number of database servers since adding new servers means changing the hash function.[2] [3]

c. ***List Partitioning:-*** List partitioning enables you to explicitly control how rows map to partitions by specifying a list of discrete values for the partitioning column in the description for each partition. The advantage of list partitioning is that you can group and organize unordered and unrelated sets of data in a convinent way. For a table with a region column as the partitioning key, the North America partition might contain values Canada, USA, and Mexico.[2][3]

II. TYPES OF VERTICAL PARTITIONING ALGORITHMS

Improving the performance of a database system is one of the key research issues now a day. Distributed processing is an effective way to improve reliability and performance of a database system. Distributed and parallel processing on database management systems (DBMS) is an efficient way of improving performance of applications that manipulate large volumes of data[6]. This may be accomplished by removing irrelevant data accessed during the execution of queries and by reducing the data exchange among sites, which are the two main goals of the design of distributed databases. Distribution of data is a collection of fragmentation, allocation and replication processes. There are two aspects of distribution design: *fragmentation* and *allocation.*[6]Fragmentation is a design technique to divide a single relation or class of a database into two or more partitions such that the combination of the partitions provides the original database without any loss of information.

This reduces the amount of irrelevant data accessed by the applications of the database, thus reducing the number of disk accesses. Fragmentation can be horizontal or vertical[7]. Horizontal fragmentation (HF) allows a relation or class to be partitioned into disjoint tuples or instances. Vertical fragmentation (VF) allows a relation or class to be partitioned into disjoint sets of columns or attributes except the primary key. Various partitioning algorithms are discussed to solve the problem of data fragmentation, allocation and cost of transactions in distributed database. The partitioning algorithms use some heuristics to create fragments of a relation.[7]

Input:-The input to most of these algorithms is an Attribute Usage Matrix (AUM). AUM is a matrix, which has attributes as columns, and queries as rows and the accesses frequency of the queries as values in the matrix. Most of data fragmentations algorithms use an Attribute Affinity Matrix (AAM) derived from the AUM provided as input[8]. An AAM is a matrix in which for each pair of attributes, the sum total of frequencies of queries accessing that pair of attributes together is stored. The input to the vertical partitioning algorithm is an attribute usage matrix. The algorithms are discussed below:-

A. Bond energy algorithm:- The Bond Energy Algorithm (BEA) is used to group the attributes of a relation based on the attribute affinity values in AAM. It is considered appropriate for the following reasons:- [8]

a. It is designed specially to determine groups of similar items as opposed to a linear ordering of the items. (ie. It clusters the attributes with larger affinity values together, and the ones with smaller values together). [8]
b. The final groupings are insensitive to the order in which items are presented to the algorithm.
c. The AAM is symmetric, and hence allows a pair wise permutation of rows and columns, which reduces complexity.[9]
d. The computation time of the algorithm is reasonable. $O(n_2)$, where n is the number of Attributes.

This algorithm takes as input the attribute af finity matrix, permutes its rows and columns and generates a clustered affinity matrix (CAM). The permutation is done in such a way to maximize the following globalfinity measure (AM).Generation of the Clustered Affinity Matrix is done in three steps:

a. Initialization: - Place and fix one of the columns of AAM arbitrarily into CAM. [10]

b. Iteration: - Pick each of the remaining n-i columns (where i is the number of columns already placed in CAM) and try to place them in the remaining i+1 positions in the CAM matrix. Choose the placement that makes the greatest contribution to the global affinity measure described above. Continue this until no more columns remain to be placed.[10]

c. Row Ordering: - Once the column ordering is determined, the placement of the rows should also be changed so that their relative positions match the relative positions of the columns. When the CAM is big, usually more than two clusters are formed and there are more than one candidate partitions.[10]

B. Binary vertical partitioning algorithm: - The Bond Energy Algorithm determines an ordering of attributes, but it is still left to the subjective judgment of the designer to decide how to clump the attributes together to form fragments. The binary vertical partitioning algorithm uses the clustered affinity matrix to partition an object into two non-overlapping fragments[9]. The approach of this algorithm is splitting rather than grouping with the objective of finding sets of attributes that are accessed mostly by distinct set of applications. Assume that point x is fixed along the main diagonal of the clustered affinity matrix. The point x defines two blocks: upper (U) and lower (L). Each block defines a vertical fragment given by the set of attributes in that block. If At is the set of attributes used by transaction t, then it is possible to compute the following sets:

T= (t|t is a transaction)
LT = (t|A(t) $\subseteq$ L)
UT = (t|A(t) $\subseteq$ U)
IT = T - (LT U UT)

T represents the set of all transactions. LT and UT represent the set of transactions that match the partitioning,

as they can be entirely processed using attributes in the lower or upper block, respectively; IT represents the set of transactions that needs to access both fragments.[9][10]

$CT = t \in T\ q_t$

$CL = \sum t \in LT\ q_t$

$CU = \sum t \in UT\ q_t$

$CI = \sum t \in IT\ q_t$

CT counts the total number of transaction accesses to the considered object. CL and CU count the total number of accesses of transactions that need only one fragment; CI counts the total number of accesses of transactions that need both fragments. Totally n-1 possible locations of point x along the diagonal is considered, where n is the size of the input matrix (ie. the number of attributes). A non-overlapping partition is obtained by selecting the point x along the diagonal such that the following objective function z is maximized: [10][11]

$$\max z = CL*CU-CI_2$$

The partition that corresponds to the maximal value of the z function is accepted if z is positive and rejected otherwise. The above objective function comes from an empirical judgment of what should be considered a "good" partitioning. The function is increasing in CL and CU and decreasing in CI. For a given value of CI, it selects CL and CU in such a way that the product CL* CU is maximized.[10]

This results in selecting values for CL and CU that are as nearly equal as possible. Thus the above function z will produce fragments that are "balanced" with respect to the transaction load. This algorithm has the disadvantage of not being able to partition an object by selecting out an embedded "inner" block.

C. Limitations of the Bond Energy and Binary Vertical Partitioning Algorithms:

All the Algorithms discussed above use affinity matrix as input and because the attribute affinity is a measure of an imaginary bond between a pair of attributes, this measure does not reflect the closeness or affinity when more than two attributes are involved.

a) In the BEA the creation of partitions is left to the subjective evaluation of the designer.[8]
b) There is no common criterion or objective function to compare and evaluate the results of these vertical partitioning algorithms.[8]
c) The above algorithms assumes that there will always be a possibility of an (n-1) partitioning for a relation R, without ignoring that there could be a situation where considering the entire relation R as one fragment could be the optimum solution, i.e. having an (n-0) partition possibilities.[11]

a. Graph-based vertical partitioning:- A new algorithm has been developed which is based on a graphical technique This algorithm starts from the attribute affinity matrix by considering it as a complete graph called the "affinity graph" in which an edge value represents the affinity between the two attributes, and then forms a linearly connected spanning tree. By a "linearly connected tree" we imply a tree that is constructed by including one edge at a time such that only edges at the "first" and the "last" node of the tree would be considered for inclusion. We then form "affinity cycles" in this spanning tree by including the edges of high affinity value around the nodes and "growing" these cycles as large as possible. After the cycles are formed, partitions are easily generated by cutting the cycles apart along "cut-edges". The major feature of this algorithm is that all fragments are generated by one iteration in a time of $O(n_2)$ that is more efficient than the previous approaches.[11]

b. Advantages:- The major advantages of this method over the previous approaches are:-

a) There is no need for iterative binary partitioning. The major weakness of iterative binary partitioning used is that at each step two new problems are generated increasing the complexity; furthermore, termination of the algorithm is dependent on the discriminating power of the objective function.[12]
b) The method requires no complementary algorithms such as the SHIFT algorithm that shifts the rows and columns of the affinity matrix.The comp lexity of this approach is $O(n_2)$ as opposed to $O(n_2 \log(n))$[12]

i. Disadvantage:- This algorithm produces a fixed no of partitions and it is difficult to control over number of partitions to be generated. To overcome from this problem Exhaustive Enumeration Algorithm is developed.

d. Exhaustive Enumeration Algorithm:- This algorithm exhaustively enumerates all possible combinations of the attributes. Hence we can easily choose the number of partitions in the partition scheme. However we are limited by the number of attributes. We have run this algorithm for attribute usage matrices with upto ten attributes. This algorithm is to be modifed to incorporate heuristics to reduce the search space. Then it is possible to work with attribute usage matrix with increased number of attributes.[11]

III. CONCLUSION AND FUTURE WORK

In this paper we conclude the comparison of various vertical partitioning algorithms to solve the vertical partitioning problem. we address the problem of n-ary vertical partitioning problem. The objective function derived in this paper is being used for developing heuristic algorithms that satisfy the objective function. we first derive an objective function that is suited to distributed transaction processing and solves the problem of data fragmentation, allocation and replications in relational database. From this, we conclude that Exahaustive algorithm gives promising results as compared to other vertical partitioning algorithms. Further work can be done to derive the objective function which generalizes and subsumes earlier work on vertical partitioning in distributed database with non relational databases. Work can also be done to develop a partitioning algorithm which can follow the different mapping and load sharing techniques to have dataload with better performance using non relational databases.

IV. REFERENCES

[1]. Vatika Sharma, Meenu Dave, "Comparison of SQL and NoSQL Databases, "International Journal of Advanced

Research in Computer Science and Software Engineering", Volume 2, Issue 8, August 2012.

[2]. Partitioning [online.] Available : http://en.wikipedia.org/wiki/Partition(database)

[3]. Oracle® Database Vldb And Partitioning Guide [online] : Available: http://docs.oracle.com/cd/B28359_01/server.111/b32024/.pdf.

[4]. Building Scalable Databases Pros And Cons Of Various Database partitioning Schemes[online] Available:http://www.25hoursaday.com/weblog/2009/01/16/BuildingScalableDatabasesProsAndConsOfVariousDatabaseShardingSchemes.aspx

[5]. Rick Cattell, "Scalable SQL and NoSQL Data Stores", Communications of the ACM, December 2011.

[6]. Shahidul islam khan, Dr. A. S. M. Latiful Hoque, "A New Technique for Database Fragmentation in Distributed Systems", International Journal of Computer Applications Volume 5-No.9, August 2010.

[7]. Ms.P. R. Bhuyar , Dr. A.D.Gawande , Prof. A.B.Deshmukh, "Horizontal Fragmentation Technique in distributed Databases", International Journal of Scientific and Research Publications, Volume 2, Issue 5, May 2012

[8]. D. W. Cornell and P. S. Yu, "A vertical partitioning algorithm for relational databases", Proceedings of the Third International Conference on Data Eng, February 3-5, pp.30-35, 1987.

[9]. Shamkant Navathe, Stefeno Ceri, Gio Wiederhold, Jinglie Dou, "Vertical Partitioning Algorithms for Database Design", ACM Transactions on Database Systems, Vol. 9, No. 4, December 1989.

[10]. Jeyakumar Muthuraj, " A Formal Approach To The Vertical Partitioning Problem In Distributed Database Design" Proc. of PDIS-2,San Diego, Jan 1993.

[11]. S. Chakravarthy, J. Muthuraj, R. Varadarajan, and S. B. Navathe, "An objective function for vertically partitioning relations in distributed databases and its analysis", Distributed and Parallel Databases, Springer, Vol. 2, No. 2, pp. 183–207, 1994.

[12]. S. Navathe, and M. Ra., "Vertical Partitioning for Database Design: A Graphical Algorithm", Proceedings of the ACM SIGMOD international conference on Management of data, Volume 18 ,Issue 2, pp. 440-450, june 2000.

[13]. Adrian Runceanu, Marian Popescu, "An algorithm for replication in distributed databases", In proceeding of International Joint Conferences on Computer, Information and Systems Sciences, and Engineering, Volume 12,2011.

