International Journal of Database Theory and Application
Vol.9, No.3 (2016), pp.107-112
http://dx.doi.org/10.14257/ijdta.2016.9.3.12

Multi-Tenant Data Storage Model and Performance Evaluation

Dun Li*, Zhenfei Wang, Zhiyun Zheng* and Jin Zhao

School of Information Engineering, Zhengzhou University, Zhengzhou, China
llidun@163.com

Abstract

Multi-tenant data storage model has multiple solutions and comparing the different storage solutions can help users improve their work efficiency. This paper proposes a query performance evaluation method based on the relational algebra. First of all, we introduce three wide table models. Secondly, we unite the format of tenant query SQL statement by analyzing structure of storage model, replace the unified format SQL with the relational algebra and evaluate the I/O cost of SQL query using relational algebra. Finally, through theoretical calculations and experimental simulations, we evaluate the performance of multi-tenant storage model according to query performance. The results show our evaluation method based on relational algebra provides new perspective for the study of performance evaluation in multi-tenant data model.

Keywords*: Performance Evaluation; wide table; Relational algebra; View Definition*

1. Introduction

In order to evaluate the multi-tenant storage performance, there are several solutions to match the tenants' customized demands and get efficiency. Space utilization can be compared through data intensity [1], and query efficiency can be compared by query costs obtaining form experiments using data modeling [2]. The comparison results from experiments are usually different because of the factors such as experimental design and the hardware environment. How to exclude these factors and compare merits essentially among different levels of storage solutions in query efficiency is the main emphasis in this paper.

In this paper, a query performance evaluation method based on relational algebra is presented through the study of three wide table storage models. Firstly, we establish memory and buffer of the view definition generation to reduce the consumption of additional data reorganization operations, then use query rewriting method based on the view definition replacement to unify and standardize data query methods and convert the SQL statements into syntax tree of relational algebra. After that, take advantage of relational algebra evaluation algorithm in the I/O cost to estimate the performance of multi-tenant query SQL statements. Finally, the results from qualitative analysis and experimental comparisons of query performance in different storage models could provide some reference value for multi-tenant data performance evaluation.

2. Multi-Tenant Wide Table Data Storage Model

Wide table storage model is one of the effective multi-tenant storage solutions because of the good performances in reducing storing cost. It is the classic representation in sharing-database-sharing-infrastructure storage. Based on the classic wide table model, there are multi improvements, such as multi wide table and multi extension table.

ISSN: 2005-4270 IJDTA

2.1 Wide Table

The wide table came from the relational research of common relation databases by David Maier and Jeffrey Ullman [3]. The storage model of wide table is pure multi-tenants data model, is also the classic representative of sharing database and sharing data storage infrastructure. In the wide table, tenants' data in different patterns are stored in a big table where the type of every column is VARCHAR compatible with all kinds of data type. Since different tenants have different data model, the big table has many columns such as the 500-column table of Salesforce.com [4-5]. The number of the tables doesn't change with the tenants scale for saving database space.

The wide table model includes MetaData table and SparseDataTable. MetaData table stores Meta data, which described the tenants' data in the wide table. The Meta data provides support for query rewriting of tenant data. SparseDataTable is a wide table storing the tenants' entity data. Except the necessary tenant's identifier Tenantid and database identifier TableName, other columns are used to store entity data.

2.2 Multi Wide Table

In the data storage of multi wide table, multi wide tables with different columns replace the single wide table in wide table model to store the tenants' business data, and build up the Meta data to store the tenants' customized information [6]. The multi wide table model includes:

(1) SparseDataTable1 to SparseDataTableN has different columns.

(2) MetaSparseData stores the basic information of these wide tables including the table name, the column number and so on.

(3) MetaData table append SparseTable for mapping the real storage position of tenants data, that is, which wide table do the tenants' data store.

2.3 Multi Extension Table

In the data storage model of multi extension table, multi extension tables with basic table replace the single wide table in wide table model to store the tenants' business data, and build up the Meta data to store the tenants' customized information [7]. The multi extension table model includes:

(1) BasicTable has fixed columns, storing the basic customized information of the tenants to match their customized demand in initial stage.

(2) ExtensionTable1 to ExtensionTableN has different columns

(3) MetaExtensionTable stores the Meta data of these extension tables, including the table name and the column number.

(4) MetaData table append ExtensionTable for indicating the extension table storing tenants' customized information.

3. SQL to Relational Algebra

3.1 View Definition

View definition is SQL definition customized by tenants. Simpler than dynamic SQL, it has better readability and further optimization. However, it will be same to the separate table if every tenant creates views for every table. No defining a real view, view definition stores the SQL statements into databases as the SQL string of tenants' reforming data. It avoids the fault of view definition and meets the demand of reforming logical data.

Let the view SQL be

Select [columnList] from [tableList] where [condition]

Where columnList is the column of logical table, tableList is a link table for reforming the logical table, condition is choosing condition, tenanted and tableid are the identifiers

of tenant and logical table respectively. The spelling string algorithm of view SQL is following:

Query the Meta data to get columnList as follows:

<t1.>column1 as realName1, …, <tN.> columnN as realNameN

columnN is the real column name, realNameN is the column name in logical table, N is the column number in logical table. tN is the alias name of logical table, <> is optional according to different models.

Determine the link table's tableList as follows:

t<leftouterjoin t1 on t.tenantid=t1.tenantid and t.tableid=t1.tableid and t1.column=' realName1'…leftouterjoin tN on …>

Leftouterjoin times are different, range from 0 to N, and the link times is lower to 3 in horizontal storage model.

Determine the tenants' information as follows:

t.tenantid=' tenantid' and t.tableid='tableid'

3.2 The Basic SQL in Relational Algebra

In the relational model, relational algebra is the classic processing query language. The basic SQL in relational algebra are following:

(1) Select: represented by σ.

(2) Project: represented by π.

(3) Join: a dual operation represented by ⋈. The join operation of R and S is marked as R⋈S.

Creating the relational algebra syntax tree of view SQL is the first step in tenants' query; the next step is to obtain the query data tenants needed from the view definition. The corresponding relational algebra syntax trees are following in Figure 1 and 2.

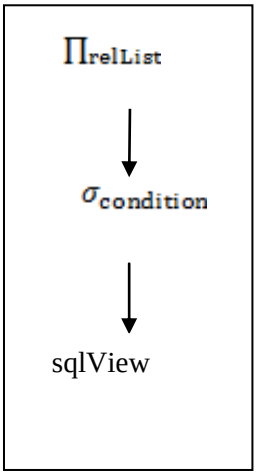

Figure 1. Relational Algebra Syntax

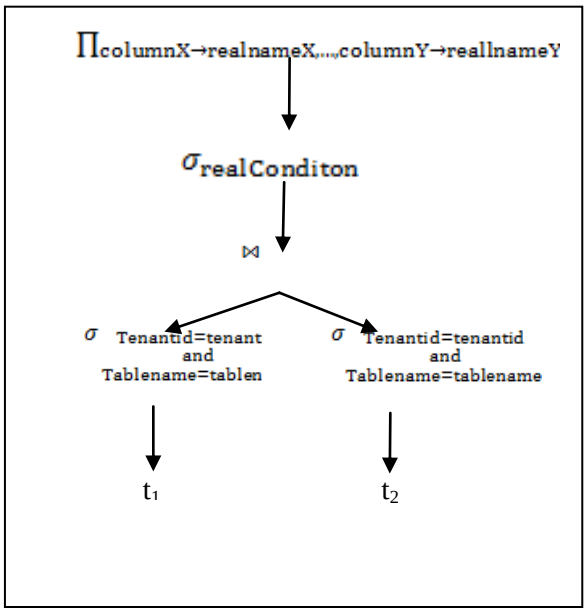

Figure 2. Optimized Relational Algebra Syntax Tree

4. Relational Algebra Evaluation on Disk I/O Cost

Since the time cost on disk access is much more than on the data, in other words, the time cost on data access from disk in much more than from memory, the block access (disk I/O access) is the approximative value of time cost which algorithm needed. At the same time, the cost of storing results relies on the results' scale unrelated to the operation. In most cases, the results print or display in the browsers and the output cost is close to 0 or relies on the application. In the paper, we suppose that all relation operations are on the disk and the result operations are in the memory.

The disk I/O number, as the measure criteria for relational algebra operation, is the cost of every operation and then we compare them on time cost. The parameters are: B(R) is the block number of tuples in relation R; T(R) is the tuple number in R; V(R, a) is the number of different values corresponding to attribute a in R. The disk I/O costs are:

(1) Project: the cost is executing table scanning or index scanning once if R is on the disk. If R is agglomerative, the cost is B; otherwise is T.

(2) Select: select operation is intended to decrease the number of tuples. The simplest select is T(R)/V(R, a), if an attribute is equal to a constant and we know or could estimate the attribute value different values. If select is multiple equivalent AND, the result would be concatenation connection of multiple connections that every connection inspect one condition.

(3) Join: is complicated and we analyze it as follows:

Suppose the conventional join of two relations involved the two attributes, R(X, Y) join to S(X,Y), where Y is a single attribute, X and Z are any attribute set. Thinking the relation between R and Y in S:

(a) If two relations have disjoint Y-value set, join is a null set and T(R⋈S)=0.

(b) If Y is the primary key in S and the external key in R, every tuple in R join to one tuple in S and T(R⋈S)=T(R).

(c) If almost all tuples in R and Shave same Y-value, T (R⋈S)=T(R)T(S).

Aim at the most common situation, we make two supposes:

Including the value set. If Y is one attribute in multi relations, every relation selects the values from the fixed list y1, y2, … and obtain all values in front part. Therefore, every Y

in R would be Y-value in S if two relations, R and S, have Y-attribute and V(R, Y) <V(S, Y).

Preserving the value set. If R joins to another relation, A which is not the join attribute could not lost value in its potential value set. If A is one attribute in R and not in S, V (R⋈S, A) =V(R, A) would be true.

Suppose r is one tuple in R, s is one tuple in S and V(R, Y)>V(S, Y), Y-value in s would be in R and the same Y-value probability in r and s is 1/V(R, Y). Otherwise, if V(R, Y) <V(S, Y) Y-value in r would be in S and the same Y-value probability in r and s is 1/V(S, Y). Therefore, T (R⋈S) =T(R) T(S)/max (V(R, Y), V (S, Y)).

If Y includes more than one attribute in join R(X, Y) ⋈S(Y, Z), R⋈S equals to multiple T(R) to T(S). For the public attribute y in every R and S, it is come from dividing the larger value in V(R, y) and V(S, y).

5. Evaluation and Experiment Analysis

In order to verify the justification and rationality of three wide table models in the paper, we do the qualitative analysis and compare the experiment results to evaluate query performance.

5.1 Qualitative Analysis

In order to evaluate for different view model, we set some parameters: R is the relation in wide table, S is the relation in multi wide table, U is the relation in multi extended table, W is the relation of extended tables in multi extended table, T(R) is the tuple number in R, V(R,a) is the number of different values to attribute a in R. P'' is the view query performance in wide table model, P' is in multi wide table model, P is in multi extended table model. We define three query performances as follows:

$$P'' = T(R)/(V(R, \text{tenantid}) * V(R, \text{tablename})) \quad (1)$$

$$P' = T(S)/(V(S, \text{tenantid}) * V(S, \text{tablename})) \quad (2)$$

$$P = \frac{T(U)T(W)}{\text{Max}(V(U,\text{tenantid}),V(W,\text{tenantid})) * \text{Max}(V(U,\text{tablename}),V(W,\text{tablename})) * \text{Max}(V(U,\text{rowid}),V(W,\text{rowid}))} \quad (3)$$

T(R) is the sum of all tuples in S, tenantid and tablename are the sum of all tenants and tables in S respectively, and it is obtained that P'>P''.

Tenants in multi extended table model is less than in extended table model, corresponding private tables is less that tenants in extended table model, rowed is the maximal identifier of tuple in relations that is the corresponding tuple number to the largest-record table in relation. Suppose record number in W in multi extended table model is same to in S in multi wide table model, which is larger in P and P' depend on the formula (4):

$$I = T(U)/V(W, \text{rowid}) \quad (4)$$

If I>1 then P'>P, it is shown that the query performance of multi wide table model is better than that of multi extended table model.

If I<1 then P>P', it is shown that the query performance of multi extended table model is better than that of multi wide table model.

If I=1 then P=P', it is shown that they are same.

5.2 Quantitative Evaluation

We generated business data according to TPC-W standard of simulation data generation, experiment data with 100 columns and 30000 records are stored in three different storage data models respectively. We do some experiments to analysis the query performance of their view definition and compare their evaluation values and real query cost. Figure 3 is the performance evaluation of three multi-tenant data storage model. It is shown that the query performance of wide table model is worst and evaluation value is highest, the query performance of multi wide table model is best and evaluation value is

lowest, the query performance and evaluation value of multi extended table model are both middle.

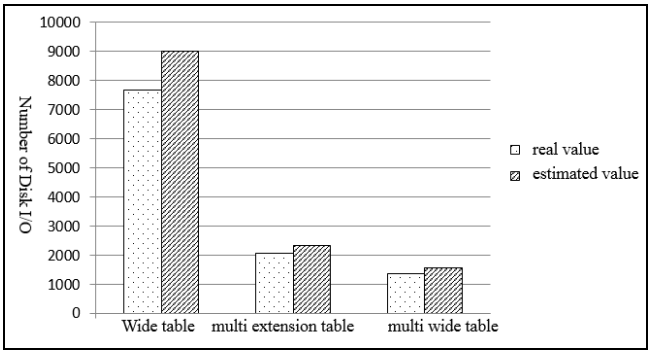

Figure 3. The Value of Performance Evaluation vs Real Value

Through the experiments, it is obvious that evaluation values are higher than real values because the database chooses the lowest-cost plan from multi execution plans while evaluation values in the experiments aren't optimized. However, the query of evaluation values and real values are synchronous, it is shown that the evaluation values can't evaluate the query performance of one tenant exactly but has some reference value for performance comparison and analysis of multi-tenant data model.

6. Summary

Multi-tenant data storage model has many solutions and the comparison of different solutions could help users increase of their efficiency. The paper proposed a new method of query performance evaluation based on relational algebra. Firstly, we introduced three wide table data storage models. Secondly, we analyzed the structure of storage model, replaced the unified SQL with relational algebra to evaluate the I/O cost of SQL query statements. At last, we did theoretical calculation and experiment simulation, and the results showed that the evaluation based on relational algebra provided a new way for performance evaluation of multi-tenant data storage model.

References

[1] S. Berchtold, D. A. Keim and H. P. Kriegel, "The X-tree: An index structure for high-dimensional data", Readings in multimedia computing and networking, vol. 451, **(2001)**.
[2] S. Wouw, J. Viña and A. Iosup, "An Empirical Performance Evaluation of Distributed SQL Query Engines", Proceedings of the 6th ACM/SPEC International Conference on Performance Engineering. ACM, Austin, USA, **(2015)**.
[3] D. Maier and J. D. Ullman, "Maximal objects and the semantics of universal relation databases", ACM Transactions on Database Systems, ACM, vol. 1, **(1983)**.
[4] C. D. Weissman and S. Bobrowski, "The design of the force.com multitenant internet application development platform", SIGMOD Conference, Rhode Island, USA, **(2009)**.
[5] E. Chu, J. Beckmann and J. Naughton, "The case for a wide-table approach to manage sparse relational data sets", Proceedings of the 2007 ACM SIGMOD international conference on Management of data, Beijing, China, **(2007)**.
[6] W. L. Chen, S. D. Zhang and L. N. Kong, "A multiple sparse tables approach for multi-tenant data storage in SaaS", The International Conference on Industrial and Information Systems, Karnataka, India, **(2010)**.
[7] H. Koziolek, "The SPOSAD architectural style for multi-tenant software applications", 2011 9th Working IEEE/IFIP Conference on Software Architecture. (2011) June 20–24; Boulder, USA.

