A Multiple Sparse Tables Approach for Multi-tenant Data Storage in SaaS

Chen Weiliang, Zhang Shidong[†], Kong Lanju
School of Computer Science and Technology
Shandong University
Jinan, China
chenweiliang999@sina.com, zsd@sdu.edu.cn, klj@sdu.edu.cn

***Abstract*—The sparse table approach is an effective storage solution for multi-tenant data in SaaS. However in the sparse table, because of the shared data storage of different tenants that have different schemas, there are so many nulls which result in a waste of storage space and query efficiency problems. We analyze the difference between the multi-tenant sparse data and the traditional sparse one such as e-commerce data and propose a multiple sparse tables approach, in which the tables have gradient columns. We carry out experiments and the results figure out that the approach can effectively reduce the space waste and improve the query performance.**

Keywords-multi-tenant; sparse table; null

I. INTRODUCTION

In the SaaS business model, the single application instance is leased by multiple tenants. The multi-tenant data belong to the same application, but are accessed by different tenants. The sparse table, as an effective store solution for multi-tenant data, refers that all tenants' data share one big wide table. Since different tenants have different schemas and different columns, the wide table will be always sparse. For examples, tenant1 needs 100 columns for its data while tenant2 only needs 10 columns. When the data of the two tenants are put into one wide table, the remaining 90 columns will be full of nulls for tenant2.

The problems sparse table approach meets here are similar to the ones in traditional sparse data processing such as e-commerce data. For both, the large amounts of nulls waste the space and reduce the query performance. So the methods for the traditional sparse data processing can be chosen in the multi-tenant data storage, such as vertical schema and interpreted attribute storage. However, the sparsity in multi-tenant environment has its own characteristics.

In this paper, we point out the differences between the multi-tenant sparse data and the traditional sparse one after studying the specific features of the former and we also give a multiple sparse tables approach. The core idea of our method is using several sparse tables that have gradient columns to replace the original wide sparse table, and then each tenant chooses the most appropriate one.

[†] Corresponding author: Zhang Shidong, Jinan, China. 250101

The rest of the paper is structured as follows. Section II refers to the related works while section III refers to the similarities and differences. Our approach is presented in section IV and the experiments follow up in Section V.

II. RELATED WORKS

One of the most important features of the multi-tenant data in SaaS is that different tenants' data belong to the same application, but are accessed by different tenants [1, 2]. The specific property comes from the "single application instance multiply tenancy" business model of SaaS [3]. In the sparse table approach, all tenants' data are stored in one big wide table that is always sparse since different tenants have different customizations and consume different columns [4].

The shared data storage architecture of sparse table prevents the number of tables to boom with the increase of the number of the tenants and the tuple reconstruction is also relatively simple [5, 6]. However, the most troublesome problem is the existence of the large number of nulls which waste the space and give a negative impact on the query performance. These problems are similar to what we meet in traditional sparse data processing, and so some methods of dealing with traditional sparse data like e-commerce data can be chosen.

[7] puts forward a strategy of vertical storage. In this method, sparse data are stored as <oid(object identifier), key(attribute name), val(attribute value)>. The vertical storage model is effective in reducing the nulls but in order to query under this model, we should create a logical horizontal view of the vertical representation and transform queries on this view to the vertical tables. For the multi-tenant data we meet, the problem is that tuple reconstruction will be very time-consuming [8].

[9] proposes to extend the RDBMS tuple storage format to allow the representation of sparse attributes as interpreted fields. Storage system will add the attribute and value information to the current tuple record when it finds a non-null value. After the discussions in the next section we will find that we can choose other solutions such as multiple sparse tables to deal with the nulls rather than change the storage format.

[10] indicates that storing the sparse data set in a single table is the right way to proceed because it frees users from a tedious and potentially ineffective schema-design phase. The schema split method which is not advocated in this paper has

similarities with our method. Both of them replace one single table with several tables, however, the single sparse table of multi-tenant data is originally produced by the shared storage of different schemas owned by different tenants, and so the difference is that there is no schema spit in our method.

III. SIMILARITIES AND DIFFERENCES BETWEEN THE MULTI-TENANT SPARSITY AND THE TRADITIONAL SPARSITY

The multi-tenant sparse data and the traditional sparse data have many points in common. First, there are a large number of columns. For multi-tenant data, we can not predict the needs of tenants, and so we have a large number of columns of the data table to satisfy the different tenants. Second, the data tables for both are always sparse although the reasons are different. Third, there is continual schema evolution and we need frequent altering of the table. Fourth, the large numbers of nulls result in the space waste and degrade the query performance.

Although there are some similarities, there are also some significant differences since the backgrounds of both are different.

First, in multi-tenant environment, the sparsity is caused by the shared data storage of different schemas owned by different tenants while there is no concept of tenant in traditional sparse data.

Second, in multi-tenant sparse table, there are two types of nulls: ***schema nulls*** and ***value nulls***. The former indicates that the tenant does not customize the columns while the later indicates that the tenant customizes the columns, but values are real nulls. For the multi-tenant sparse table, most of the nulls are schema ones and it means that the data of one tenant is perhaps not sparse once we reduce the schema nulls.

Third, in multi-tenant sparse table, there is one obvious scene that the data are ***left-intensive*** or we can call it ***right-sparse***. The causes of the phenomenon are that most of the nulls are schema nulls and tenants consume the columns of the sparse data table from left to right.

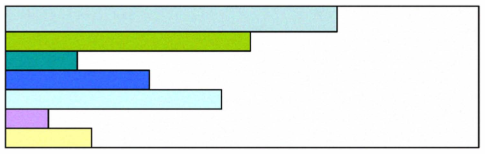
One possible data distribution is shown in Figure1. In (a) the different colorful stripes represent the data that has the same columns. We can see that the sparsity in (a) is caused by the shared storage of different tenants' data that have different schemas and columns and the nulls are mostly schema nulls. Meanwhile the distribution clearly shows the left-intensive feature. It is just these differences that make us use some other methods such as our multiple sparse tables approach to deal with the multi-tenant sparse data.

IV. THE MULTIPLE SPARSE TABLES APPROACH

A. The multiple sparse tables approach

Figure2 (a) shows the single sparse table approach for multi-tenant. In this storage, we use a wide table with a large number of columns such as 500 as the data table [11] and in order to interpret the different schemas in the same data table we have a metadata table that illustrates the real physical positions of the virtual tables and columns accessed by different tenants.

The single sparse table approach can effectively reduce the number of the tables and makes the tuples reconstruction easy as mentioned above. However the approach also brings in the troubles caused by the nulls which give a negative impact on the storage space and the query performance.


(a) multi-tenant sparse table

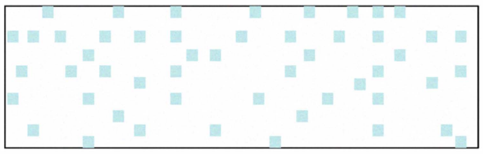
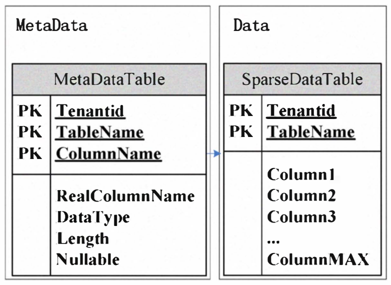
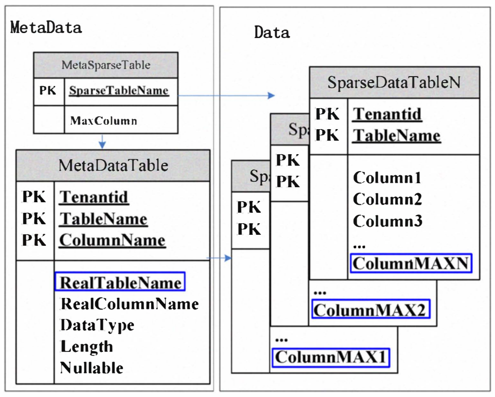

(b) traditional sparse table

Figure 1. Data distribution in both sparse tables.

We improve the method and propose a multiple sparse tables approach as show in Figure2 (b). In our method, we have several sparse tables (SparseDataTable1, SparseDataTable2 to SparseDataTableN) which have gradient columns (ColumnMax1, ColumnMax2 to ColumnMaxN) as the data tables and we add MetaSparseTable as the metadata of these data tables.



(a) single sparse table storage architecture



(b) multiple sparse tables storage architecture

Figure 2. Sparse table storage arichitecture.

When the customizations of one tenant occur, we should determine which data table is appropriate by considering the number of the columns customized (tenantCC) and the metadata of the sparse tables. The strategy is mainly to choose the sparse table with the least columns that meet the needs of the tenant. For instance, there are four sparse tables having 30, 80,200,500 columns. If the tenant has customized 50 columns in total and table2 will be chosen.

When the queries are launched by one tenant, we need to do the query rewriting driven by the metadata in MetaDataTable. The queries on the tenant's virtual private tables are translated to on the real physical sparse tables. Generally, we choose the strategy of *view-replacement* which replaces the virtual tables in tenants' SQLs with the definitions of these views (virtual tables) and of course we need to maintain the SQLs definitions of the views.

B. Determination of the number of sparse tables and their respective columns

The determination of the number of sparse tables and their respective columns is actually the speculation on the tenants' demands. Because in reality the demands can not be accurately forecasted, it is often unable to get the optimal results. However, there are some measures and principles we can consider to get a relative superiority.

First, we can get close to the needs of the tenants by analyzing the business requirements of the application. Second, the statistical analysis on the existing customization information will be very helpful. Third, we find that the preferred split strategy is to give tiny sections on the small number of the columns while coarse sections on the big ones such as our four sparse tables with 30, 80, 200, 500 columns. The reason is that the improvement will be more obvious if we project 10 columns from 500 ones than project 300 columns from 500 ones. Fourth, the number of the sparse tables should not be too large either too small. If there are too many sparse tables, it may lead to the frequent migration between tables because the schema evolution may be continuing.

Though we can use the methods mentioned above, what we would like to reiterate is that it is hard to get the optimal results because the demands of different tenants are always unknown or uncertain and the statistical analysis on the existing customization information will be very favorable.

V. USE CASES AND EXPERIMENTS

We choose DWPaaS that is a SaaS application platform developed by Research Center of Software and Data Engineering of Shandong University as our use case. We choose a sparse table with 500 columns from the data center of DWPaaS and we have 4 sparse tables with 30, 80,200,500 columns in our multiple sparse tables approach and each column referred here is varchar2 (100).

The data density in the original single sparse table is $\rho' = 0.11$ while for the multiple sparse tables the density is $\rho = 0.59$ and for each sparse table: $\rho_1 = 0.55$; $\rho_2 = 0.68$; $\rho_3 = 0.53$ (There is no data in SparseTable4). The results of the experiments show that the multiple sparse tables approach can effectively reduce the nulls. Apart from this, the approach can improve the query performance by narrowing down the scanning range.

In order to facilitate our experiments and show the results clearly, we only choose schema5 (10 columns), schema26 (50 columns), schema43 (100 columns) as our query test data. For the single sparse table approach, all the data will be put into the sparse table with 500 columns. For multiple sparse tables approach, data of schema5 will be put into SparseTable1 (30 columns); data of schema26 will be put into SparseTable2 (80 columns) and data of schema43 will be put into SparseTable3 (200 columns). Assume that they belong to the same tenant (tenanted='1') and they have same numbers of rows. We have tableJ with 50 columns (columnName1 to columnName50) and 5000 rows for the join test. The SQL definitions of the virtual table5 accessed by tenant1 are shown in tableI (table26 and table43 are similar).

TABLE I. SQL DEFINITIONS OF THE VIRTUAL TABLE5

Virtual table	SQL Definition in single sparse table approach	SQL Definition in multiple sparse tables approach
table5	slect column1 as columnName1,column2 as columnName2…column10 as columnName10 from ***sparseDataTable*** where tenantid='1' and tablename='table5'	slect column1 as columnName1,column2 as columnName2…column10 as columnName10 from ***sparseDataTable1*** where tenantid='1' and tablename='table5'

Table II shows our test SQLs which will be executed in both the single sparse table and multiple sparse tables. We have mainly focused on the joins.

TABLE II. TEST SQLS

NO	Test SQLs
1	Select * from table5 a, tableJ b where a.columnName1=b.columnName1 and a.columnName2=b.columnName2 and… and a.columnName5=b.columnName5
2	Select * from table26 a, tableJ b where a.columnName1=b.columnName1 and a.columnName2=b.columnName2 and… and a.columnName5=b.columnName5
3	Select * from table43 a, tableJ b where a.columnName1=b.columnName1 and a.columnName2=b.columnName2 and… and a.columnName5=b.columnName5

The results of the experiments are shown in Figure3 to Figure5. We can find that the multiple sparse tables can effectively reduce the time consumed by the joins. The DBMS here is Oracle10g and the server has four Intel(R) Xeon(R) cpus @ 2.0GHZ and 4 GB memory.

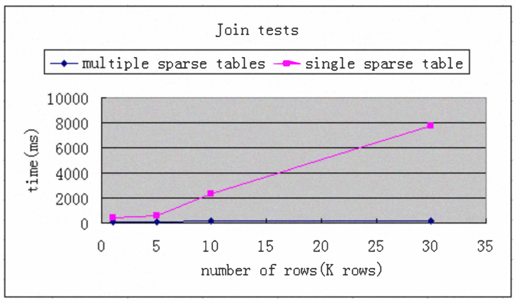


Figure 3. Join tests for schema5(10 columns).

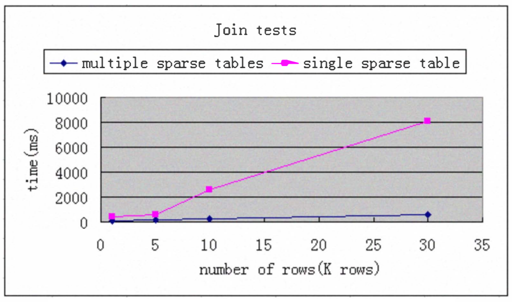


Figure 4. Join tests for schema26 (50 columns).

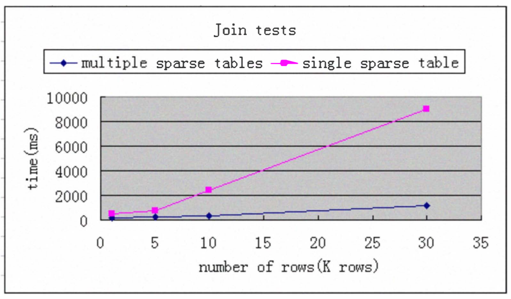


Figure 5. Join tests for schema43 (100 columns).

VI. Conclusion and Future Works

The sparse table approach is an effective solution for multi-tenant data storage in SaaS. However there are some problems including the space waste and the negative impacts on the query performance just like what we meet in traditional sparse data processing.

We propose the multiple sparse tables approach after studying the similarities and differences between the multi-tenant sparsity and the traditional sparsity. In our approach, the sparse tables have gradient columns to meet different demands of the tenants. The results of the experiments show that the approach can effectively reduce the space waste and improve the query performance. We also discuss the determination of the numbers of sparse tables and their respective columns. Some methods and principles are proposed as well.

One of the potential problems caused by this method is the data migration between the sparse tables which will be discussed in our future work and more scientific detailed approaches of the determination are also needed.

Acknowledgment

This work is supported by the National Natural Science Foundation of China under Grant No.90818001, the Natural Science Foundation of Shandong Province of China under Grant No.Y2007G24; No.Y2007G38; No.2009ZRB019YT; No.2009ZRB019RW, Shandong Distinguished Middle-aged and Young Scientist Encouragement and Reward Foundation under Grant No. BS2009DX040, Key Technology R&D Program of Shandong Province under Grant No.2007GG10001009; No.2008GG30001005; No.2009GG10001002, Science and Technology Development Program of Shandong Province under Grant No.2006GG2201052; No.2007GG1QX01036, the Special Fund for Postdoctoral Innovative Project of Shandong Province under Grant No.200703084, the Specialized Research Fund for the Doctoral Program of Higher Education under Grant No.200804221031, China Postdoctoral Science Foundation under Grant No.200904501193, and Independent Innovation Foundation of Shandong University under Grant No.2009TS030.

References

[1] Martin Grund, Matthieu Schapranow, Jens Krueger, Jan Schaffner, Anja Bog, Shared Table Access Pattern Analysis for Multi-Tenant Applications, 2008.

[2] M.T. Hoogvliet, SaaS Interface Design., Communication and Multimedia Design, 2008.

[3] Vidyanand Choudhary, "Software as a Service: Implications for Investment in Software Development", the 40th Annual Hawaii International Conference on System Sciences (HICSS'07)

[4] Wei Sun, Xin Zhang, Chang Jie Guo, Pei Sun, Hui Su,"Software as a Service: Configuration and Customization Perspectives",2008 IEEE Congress on Services Part II

[5] S. Aulbach, T. Grust, D. Jacobs, A. Kemper, and J. Rittinger, Multi-tenant databases for software as a service: Schema mapping techniques, SIGMOD, 2008.

[6] S. Aulbach, T. Grust, D. Jacobs, A. Kemper, and M. Seibold, A Comparison of Flexible Schemas for Software as a Service, SIGMOD, 2009.

[7] R. Agrawal, A. Somani, and Y. Xu, Storage and Querying of E-Commerce Data. In VLDB, pages 149–158, 2001.

[8] M. Stonebraker et al. C-Store: a Column-Oriented DBMS. In VLDB 2005.

[9] J. L. Beckmann, A. Halverson, R. Krishnamurthy, and J. F. Naughton. Extending RDBMSs To Support Sparse Datasets Using an Interpreted Attribute Storage Format. In ICDE, page 58, 2006.

[10] E. Chu, J. Beckmann, and J. Naughton. The Case for a Wide-Table Approach to Manage Sparse Relational Data Sets. In SIGMOD, pages 821–832, 2007.

[11] Craig D Weissman, Steve Bobrowski, The Design of the Force.com Multitenant Internet Application Development Platform, SIGMOD,2009.