A new meta-data driven data-sharing storage model for SaaS

Li heng[1], Yang dan[2] and Zhang xiaohong[3]

[1] College of Computer Science, Chongqing University
Chongqing, 401331, China

[2] School of Software Engineering, Chongqing University
Chongqing, 401331, China

[3] School of Software Engineering, Chongqing University
Chongqing, 401331, China

Abstract

A multi-tenant database is the primary characteristic of SaaS, it allows SaaS vendors to run a single instance application which supports multiple tenants on the same hardware and software infrastructure. This application should be highly customizable to meet tenants' expectations and business requirements. This paper examined current solutions on multi-tenancy, and proposed a new meta-data driven data-sharing storage model for multi-tenant applications. Our design enables tenants to extend their own database schema during multi-tenant application run-time execution to satisfy their business needs. Experimental results show that our model made a good balance between efficiency and customized.

Keywords: *Multi-Tenant, Software as service, Schema mapping, meta data.*

1. Introduction

Software as a Service (SaaS) is an emerging software application service and one of the hot topics in the software industry. Expressed most simply, SaaS can be defined as follows: "Software deployed as a hosted service and accessed over the Internet" [1]. Instead of paying for the software license, the end user subscribe for a paid application. In February 2000, SaaS concept started when Salesforce.com launched their web-based service and became the early SaaS adopters. In February 2001 the term Software as a Service or SaaS published for the first time in a white paper called "Software as a Service: Strategic Backgrounder" [2]. SaaS began to flourish in 2005-2006, because the internet speed had significantly increased, had become affordable, and customers had started to be more comfortable to establish business over the internet.

A particularly important challenge in a SaaS application is concerned with enabling multi-tenancy at the data tier [3, 4]. Systems at the data tier of a SaaS application are accessed by the same application for each tenant, who has own unique needs that a rigid, inextensible default data model won't be able to address. Put simply, the challenge is to consolidate multiple tenants onto one data tier resource, e. g. one database server, which can be extended for different versions of the application and dynamically modified while the system is on-line, while at the same time isolating them among one another, as if they were running on physically segregated resources.

This paper researched the related works on multi-tenancy date model, and proposed a new meta-data driven data-sharing storage model. By splitting up the "common tables" shared by each tenant, and mapping the data to "meta data tables" and "data tables", our model enables tenants to extend their own database schema during multi-tenant application run-time execution to satisfy their business needs. Experiments demonstrate that compared with previous techniques, the presented model makes a good balance between efficiency and customized.

2. Related Works

Several works have been presented in [5],[6], [7], [8] on design and implement multi-tenant database schema, such as Private Table, Extension Table, Universal Table and so on, each technique has its' own characteristics and applicable scenarios, This section will explore five techniques of multi-tenant database schema.

2.1 Extension Table

Because multiple tenants may use the same base tables, Aulbach et al. [8], [9] report that the Extension Tables are separated tables joined with the base tables by adding tenant column as well as row column to construct a logical source tables. This approach has its origins in the Decomposed Storage Model [11], where an n-column table

is broken up into n 2-column tables that are joined through surrogate values. Multiple tenants can use the base tables as well as the extension.

2.2 Universal Table

Aulbach et al. [8] refer to Universal Table as a table that contains additional columns of the base application schema columns which enable tenants to store their required columns. A Universal Table is a generic structure with a Tenant column, a Table column, and a large number of generic data columns. The data columns have a flexible type, such as VARCHAR, into which other types can be converted. The n-th column of each logical source table for each tenant is mapped into the n-th data column of the Universal Table. As a result, different tenants can extend the same table in different ways.

2.3 Pivot Table

In a Pivot Table, each row field in a logical source table is given its own row. There are four columns in the Pivot Table including: tenant, table, column, and row that specify which row in the logical source table they represent. In addition, the single data type column that stores the values of the logical source table rows according to their data types in the designated pivot Table. The data column can be given a flexible type, such as VARCHAR, into which other types are converted, in which case the Pivot Table becomes a Universal Table for the Decomposed Storage Model.

2.4 Chunk Folding

Chunk Folding[8] is a technical where the logical source tables are vertically partitioned into chunks that are folded together into different physical multi-tenant tables and joined as needed. Aulbach et al. [8] state that the performance of this technique enhanced by mapping the most used tenants' columns of the logical schema into conventional tables, and the remaining columns in the Chunk Tables which are not used by the majority of tenants.

2.5 XML Table

The XML database extension technique is a combination of relational database systems and Extensible Markup Language (XML) [6,7]. Aulbach et al. [10] state that the extension of XML can be provided as native XML data type, or by storing the XML document in the database as a Character Large Object (CLOB) or Binary Large Object (BLOB). This technique satisfies tenants' needs, because their data can be handled without changing original database relational schema.

3. A new meta-data driven data-sharing storage model

3.1 Drawbacks

In section 2, we describe five techniques on design and implement multi-tenant database schema, which can extend default data model to address tenant's unique needs. However, there are some drawbacks or limits in these techniques. The "Extension Table" technique is limited to use in small tenants applications, for the number of tables will be increased by increasing the number of tenants and the variety of their business requirements. The "Universal Table" technique enables tenants to extend their tables in different ways according to their needs. However it has the obvious disadvantage that the rows need to be very wide, even for narrow source tables, and the database has to handle many null values. Furthermore, indexes are not supported in universal table columns, as the shared tenant's columns might have different structure and data type. This issue leads to the necessity of adding additional structures to make indexes available in this technique. The "Pivot Tables" technique can eliminating NULL values and selectively read from less number of columns, but it has more columns of meta-data than actual data and reconstructing an n-column logical source table requires (n−1) aligning joins along the Row column. This leads to a much higher runtime overhead for interpreting the meta-data. It seems a good choice to use "Chunk folding", Stefan Aulbach in his paper described the advantage of this technique through many experimental data. However, this technique is in the phase of theoretical research, and lack of an effective vertical partitioning algorithm to get the most appropriate results. The last technique "XML Table" makes the data model arbitrarily extensible while retaining the cost benefits of using a shared database. If the customer requires a considerable degree of flexibility to extend the default data model, I think it is the best approach to take if the ISV wishes to use a shared database. However, this technique is limited to extend fields in a table, sometimes, customers need to define their own objects, for example, in the ERP System.

3.2 A new meta-data driven data-sharing storage model

Based on the previous work, We present a new meta-data driven data-sharing storage model, which can be implemented in the "shared database, shared schema" approach. In this section we will describe the model and implementation details. Comparison of the efficiency will be shown in section 4 by some experiments.

As illustrated in Fig 1, this technique consists of three parts: Common tables, Meta-data tables and Data tables. “Common Tables” are the same as those Non-SaaS applications except there is a column named “tenantid” which is used to isolate each tenant. “Common Tables” are shared by all tenants, it is very easy for indexing, querying and updating record. If there is no need to extend the data model, it will be high-efficiency for the pure association relationship between tables. “Meta-data tables” include “tenants” table, “custom_objects” table, “custom_fields” table. Tenants table describes the information of tenants, including “tenantid” and “tenants_name”. “Custom_objects” table describes the custom objects defined by tenants, including “tenanted”, “objectid”, “object_name” and “object_type”. “Custom_fields” table describe the information about the data field in each data objects. “Data tables” include key-value table and data-value table. Key-value table used to record extend labels and values by key-value pairs when some tenants need to add fields on “Common Tables”. Data-value table used to record field values which defined in the “Fields table” of meta-data.

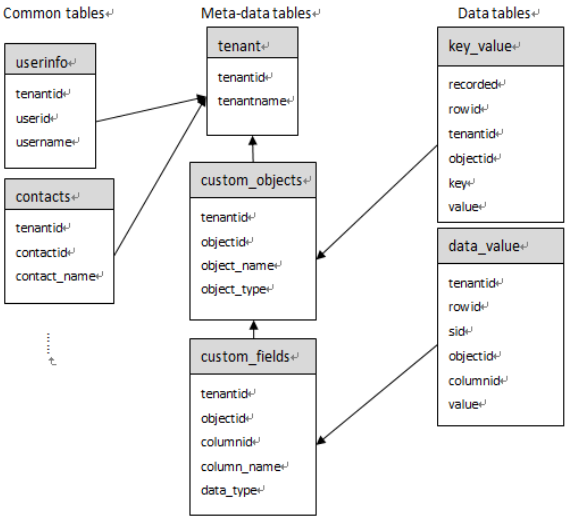

Fig. 1 “meta-data” driven data-sharing storage model.

Fig 2 shows an example that how our model store the multi-tenancy data. When an end user needs to add some fields, he should insert into “custom objects” table a record to define a new object, the “object name” is the same as the table name which needs to be extended and the “object type” is assigned to “field”. When the end user saves a record with a custom field he defined before, 3 things happen. First, the application looks up “custom_objects” table to find whether exist a record which has the same “object_name” with the current object and has a value of “object_type” equals “field”. If true, then, some boxes will be shown in the browser to input keys and values. Finally, values are saved in “key-value” table and the application creates a unique identifier for the record and saves it in the “recorded” field. When an end user needs to define his own object, he defines the object name, each field’s name and type contained in the object through the web browser. The application first creates a meta-data record both in the “custom_objects” and “custom fields” table, then insert the field value into “data-value” table. Especially, the value of “object_type” field should be set to “table” in this situation

Common Table:

tenantid	userid	username
112	201001	poly
135	201203	batty
178	200111	jenny

Meta-data Table:

tenantid	objectid	object_name	object_type
112	1	Userinfo	field
135	1	Project	table

tenantid	objectid	columnid	column_name	data_type
135	1	1	projectid	int
135	1	2	projectname	Varchar

Data Table：

recordid	row	tenantid	objectid	key	value
101	0	112	1	phone	113
102	0	112	1	age	24

tenantid	rowid	sid	objectid	columnid	value
135	1	1	1	1	1
135	1	2	1	2	firstpro
135	2	1	1	1	2
135	2	2	1	2	secondpro

Fig. 2 Sample data.

Compared with other techniques, optimizing queries is now possible on a per-tenant basis as it is now possible to create indexes on the common tenant table. Querying customer extension data will not be hindered by the handling process of NULL values. Indeed, as each tenant is given her own extension table containing her specific data, there is no need to pad the table with NULL values.

4. Experiments

In this section we will test the performance of our new data-sharing storage model, and make a comparison with other techniques in section 2. Since there is no standard data set for this task, we construct a base schema of a particular business domain application from the data schema in TPC-W database[14]. The base schema contains eight tables as depicted in Fig 3. We append a tenantid column so that it can be shared by multiple tenants as “Common tables”. Then we will extend or map the base schema by different techniques above. For example, Fig 4 shows the “chunk folding” as it is described in literature [12]. In order to avoid influence each other, multiple

copies of the base schema are created, each copy contains one model.

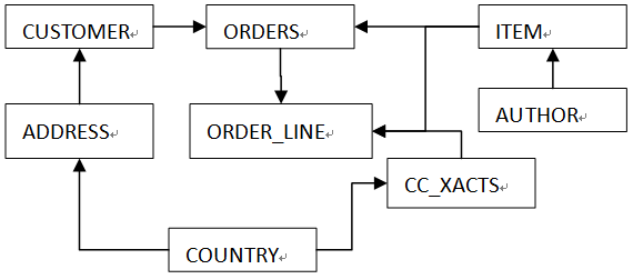

Fig. 3 TPC-W schema.

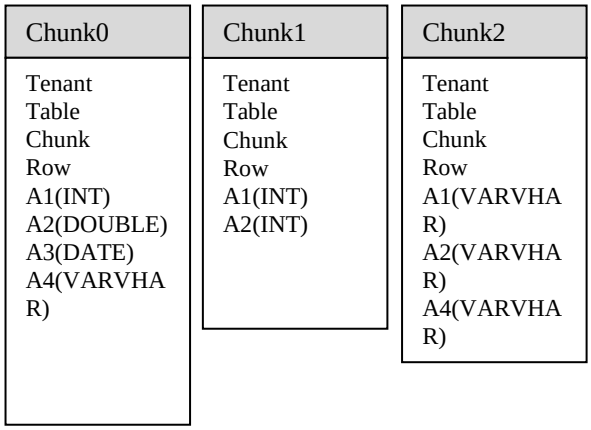

Fig.4 Chunk Folding schema.

In the experiments, we simulate a real multi-tenant scenario in "client/server" model by sending query and update requests from many tenants concurrently, and then evaluate the solutions by analysis the response time and TPS data captured during those experiments. The experiment simulated four kinds of scenarios which were described in table.1. Clients are designed to be able to simulate many tenants, in the experiment we set the number of tenants from 1-100 and every tenant had 50 users in parallel. Every simulated tenant would submit all the four kinds of scenarios mentioned above to the server and records the execution time. We divide the experiments into two groups by the number of simulated tenants. We collect average response time as the indicator of evaluation. The comparisons are shown in Fig 5. The horizontal axis shows the different request classes, and the vertical axis shows the response time in milliseconds. The experiment was run on a sql-server database server with a 3.0 GHz Intel Xeon processor and 1 GB of memory

Table 1: four kinds of scenarios

S1	Select tenant custom attributes of a single entity as if it was being displayed in a detail page in the browser
S2	Select all attributes of 1000 entities as if they were being displayed in a list in the browser
S3	Update custom entity instances as if data were updated by some clients.
S4	Insert one new entity instance as if it was being manually entered into the browser.

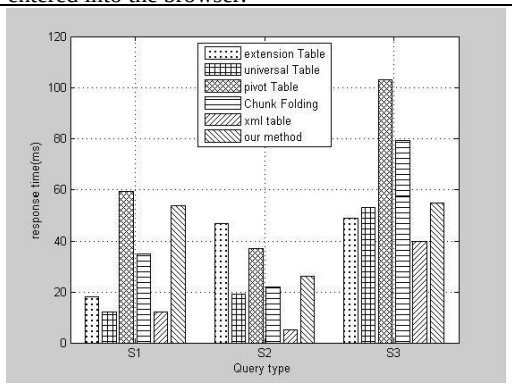

Fig. 5(a) one tenant.

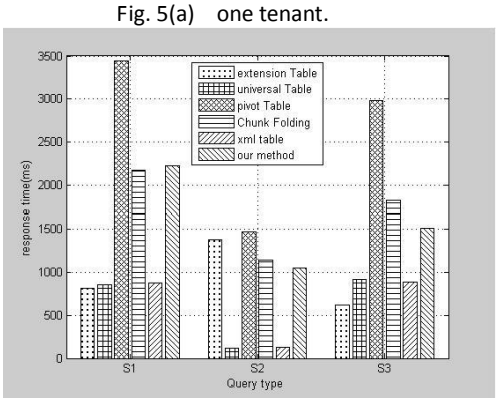

Fig. 5(b) 100 tenants.

Fig. 5 average response time in each scenario.

From Fig 5 we can see that "xml table" has the smallest response time while the "pivot table" is the highest. When there were small tenants, for example, only one tenant with 50 users in Fig 5 (a), our model spent much more time than the other techniques except "pivot table" in scenario "S1", but in scenario "S2" and "S3", our model performs better, and the response time is in a middle level and almost the same as other two techniques. That means in small requests scenario, our method has a better performance in update field and select limit entities. With the increase in the number of tenants, the response time of our model is still in a middle level in each scenario, and it performed much more stable than "chunk folding" when increased the number of tenants.

Fig 6 shows the TPS for tenants. TPS is an important indicator to measure the system capacity, it records the transactions per second of the server, the larger value it is, the better processing capacity of the system is. Fig 6 (a)-Fig 6 (d) show the TPS for tenants from scenario "S1" to

"S4" , from which we could find that our model performed as better as other techniques when doing insert operation, and it is better than "chunk folding" and "pivot table" when doing select and update operations.

In conclusion, though our model does not has the shortest response time and the highest TPS value, it makes a good balance in efficiency and customization. The most efficient technique "xml table" seemed to be a good choice, but as described in section 3.1, it is not suitable for those applications in which customers need to define their own objects. The second efficient technique "universal table" needs to handle too many null values, and it is waste of space. The "Extension Table" is good for small tenants. "Chunk folding", "pivot table" and our model are more complex, and the efficient are lower than other three techniques for additional joins are required, but they are more flexible to extend and customized. If we want to make a balance between efficiency and customized, our model is a good choice.

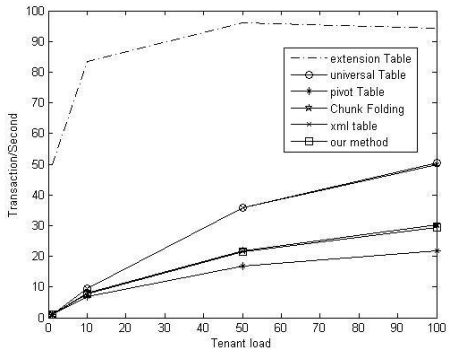

Fig 6 (a) S1 scenario

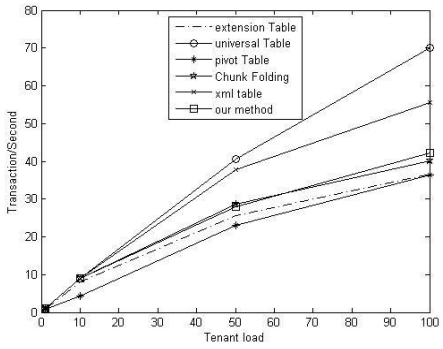

Fig 6 (b) S2 scenario

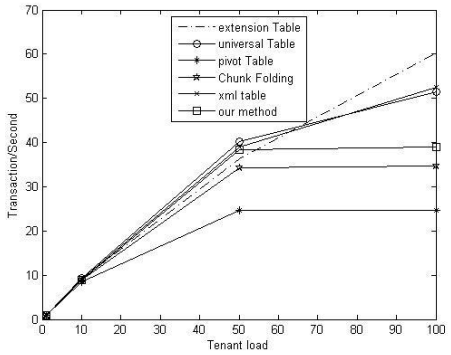

Fig 6 (c) S3 scenario

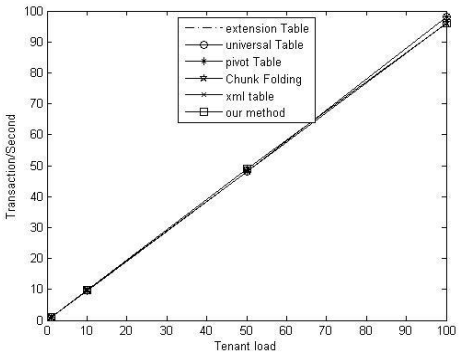

Fig 6 (d) S4 scenario

Fig. 6 TPS comparison.

5. Conclusions

In this paper, first we introduced related techniques, then we presented a new meta-data driven data-sharing storage model which can be used to implement multitenant applications on top of a standard relational database. Our approach works by splitting up the "common tables" shared by each tenant, and mapping the data to "meta data tables" and "data tables". Finally, We studied the performance of standard relational databases on OLTP queries formulated over our model, and presented the results of several experiments designed to measure the efficacy of our model and made a comparison to previous techniques.

The conclusion we draw from this paper is that our meta-data driven data-sharing storage model make a good balance between efficiency and customized. It is a flexible way of constructing tenant database schemas that provide high extensibility for multi-tenant database, enables tenants to have their own fields or tables, and improves database performance by eliminating NULL values. Our on-going work is to improve our model by assigning primary keys to unique columns, providing indexes to table columns, and

creating database relationship between virtual and common tables.

Acknowledgments

This work is supported by State Natural Sciences Foundation Projects of China under Grant (91118005), Natural Science Foundation Project of Chongqing under Grant (CSTC 2011BA2022).

References

[1] F. Burno. "Exeuting an IP Protection Strategy in a SaaS Environment",http://www.slideshare.net/Rinky25/saas-environment, Jul. 22, 2011.
[2] Nitue, "Configurability in SaaS (software as a service) applications", ISEC , 2009, pp. 19-26.
[3] F. Chong and G. Carraro. Architecture Strategies for Catching the Long Tail. Microsoft Corp. Webs ite, 2006
[4] G. C . Freder ick Chong and R . Wolter. M ulti-Tenant Data Architecture. Microsoft Corp . Website, 2006.
[5] D. J acobs and S . Aulbach. Ruminations on Multi-Tenant Databases . In Proc. of BTW Conf., pages 514–521, 2007.
[6] F. S. Foping, I. M. Dokas, J. Feehan, and S. Imran, "A new hybrid schema-sharing technique for multitenant applications", ICDIM , 2009, pp. 1-6.
[7] D. Jia, W. Hao-yu, and Y. Zhao-jun, "Research on data layer structure of multi-tenant e-commerce system", IE&EM, 2010, pp. 362 – 365.
[8] S. Aulbach, T. Grust, D. Jacobs, A. Kemper, and J. Rittinger, "Multitenant databases for software as a service: Schema mapping techniques", SIGMOD , 2008, pp. 1195-1206.
[9] S. Aulbach, T. Grust, D. Jacobs, A. Kemper, and M. Seibold, "A Comparison of Flexible Schemas for Software as a Service", SIGMOD , 2009, pp. 881-888.
[10] G. P. Copeland and S. N. Khoshafian. A decomposition storage model. InSIGMOD '85:Proceedings of the 1985 ACM SIGMOD international conference on Management of data, pages 268–279, New York, NY, USA, 1985. ACM
[11] R.Mietzner, T.Unger, R.Titze, and F.Leymann, "Combining Different Multi-tenancy Patterns in Service-Oriented Applications", EDOC, 2009, pp. 131 -140.
[12] Yao jinCheng, "Mutil-Tenant Database Memory Management Mechanism Based on Chunk Folding", Chinese journal of computers , 2011, pp. 8-9

Li Heng is a lecture in Chongqing University. Currently, he is a PhD student in College of Computer Science of Chongqing University. His interests are in cloud computing, data mining & machine learning .

Yang Dan is a professor of Chongqing University. Current research interests: data mining, computer vision, machine learning, enterprise informatization .

Zhang Xiao Hong is a professor of Chongqing University.