Available online at www.sciencedirect.com

SciVerse ScienceDirect

Procedia Engineering 29 (2012) 1107 – 1112

Procedia Engineering

www.elsevier.com/locate/procedia

2012 International Workshop on Information and Electronics Engineering (IWIEE)

Schema Evolution via Multi-Version Metadata in SaaS

Wu Shengqi, Zhang Shidong *, Kong Lanju

School of Computer Science and Technology, Shandong University, Jinan 250101, P.R. China

Abstract

The Basic-Table combined with Extension-Table (BT&ET) layout has become the popular data storage architecture for SaaS currently. For the sake of improving the processing efficiency we improve the BT&ET layout, and migrate some tenants' frequently accessed extension fields into the basic table based on the tenants' constantly need on data access. With the development of cloud computing, Multi-Tenant data need to be stored in multiple data nodes. When tenant's data storage schema evolution happens, all data nodes hava to do data migration simultaneously and tenants may perceive the influence. In order to minimize the costs and negative load of schema evolution we propose the Multi-Version metadata technology. Tenant's data on each data node may contain different version metadata, and schema envolution can run asynchronously. We carry out experiments and the results figure out that the workload decreases significantly.

Keywords: Multi-Version metadata; schema evolution; SAAS

1. Introduction

Multi-tenant data storage has become the key of SaaS. In Microsoft white paper [1] Frederick Chong et al propose a novel schema based on Basic-Table combined with Extension-Table to support Multi-tenants' data storage. Tenants' common fields are stored into Basic-Table, while the extension fields are stored into the Extension-Table.

In [4] we improve the (BT&ET) layout and propose the dynamic self-adaptive algorithm based on the improved data storage architecture. In the novel layout, we can migrate some of the tenants' extension fields which are accessed frequently into the Basic-Table, and maintain the data consistency and

* Corresponding author. Tel.: +86-138-5310-7876; fax: +86-0531-88390081.
E-mail address: zsd@sdu.edu.cn .

sustainable access during the migration. So these extension data can be accessed without tuple reconstruction and the efficiency will be increased. During the execution of data migration, we ensure the data consistency by maintaining two data storage schemas: the original schema (OGS) and target schema (TGS). The update operations have to do on both schemas, and this will generates additional workload.

With the development of cloud computing, Multi-Tenant data will to be stored on multiple data nodes instead of single data node. We need to store tenants' data on different data nodes with multiple copies in cloud to ensure the reliability of data storage. In actual operation we also need to maintain the data consistenacy between different data node copies.

When a tenant's data storage schema evolution happens, all the data nodes which contain the tenant's copies need to do data migration simultaneously and every data node has to maintain two schemas. However, this will expand the additional workload. Worse still, tenants may perceive that the response time becomes slow, and this should be avoided.On the other hand, all the data replicas share the same metadata, only until all the data node complete data migration, can we change the metadata and start to use the target schema. So even some data nodes have finished, they still need to maintian two schemas and wait for other nodes.

In this article, we propose the Multi-Version metadata technology. Every data replica can execute data migration asynchronously and owns its metadata version which consists with the data migration step on this node. As the result, we can avoid the expanded additional workload which generates by simultaneous data migration and the worthless workload during waiting.

The article is structured as follows. Section 2 outlines the Multi-Tenant data storage architecture in cloud. Section 3 describes the Multi-Version metadata architecture and the improved data engine. Section 4 presents the results of our experiments, and it's followed by related work Section 5. At last we discuss the conclusions and future work in Section 6.

2. Multi-Tenant data storage architecture in cloud

2.1. Improved BT&ET multi-tenant data storage mapping mechanism

In the shared database with shared schema data storage architecture, all of the tenants' logical view don't exist, but stored into the same table by some mapping mechanism and identified by some special columns such as tenancyid, tenantid. In our improved BT&ET layout, all of the tenant's common fields and some of their extension fields which have high access frequency are stored into the Basic-Table, while the other extension fields are stored into the Extension-Table.

In Fig. 1(a) describes an example of our practical application. Each tenant can only see his own logical view, which formed from the Basic-Table and Extension-Table by metadata.

2.2. Cloud data storage architecture

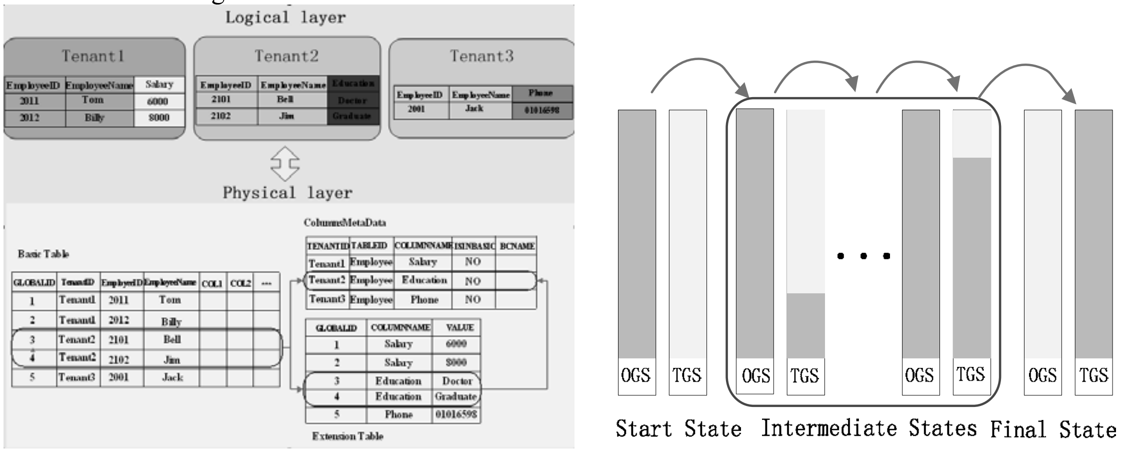
In cloud, the tenants' data are stored on different nodes with multiple replicas to ensure the reliability of data storage. When a tenant queries data, query operation should do on all of his replicas and combine the results as the final result to ensure the accuracy of the result. Also when updates data, every replica updates its data to maintain the consistency of data.

Before we introduce our storage architecture, we first define some term which we will use later.

- **Definition 1:** Data Metadata. Data Metadata describes how the tenant's data stores in the data node, by which we can mapping the physical view to tenants' logical view.
- **Definition 2:** DB Node Metadata. DB Node Metadata describes the tenant's data stores on which data nodes. It tells us the request should be send to which data nodes.

In cloud data storage architecture, all requests are sent to controller server. When controller receives a request, it identifies the tenant by the context and gets the Data Metadata of the tenant from controller server. The data engine rewrites the tenant's logical SQL into physical SQL by the Data Metadata.Then the router sends the physical SQL to the data nodes which are determined by the DB Node Metadata. The controller will wait until every data node finishes the operation and return the combined result to tenant.

In this data storage architecture, when a tenant's data needs schema evolution, all data nodes of the tenant start to migration simultaneously. Each has to maintain two schemas and this will expand the additional workload. Worse still, because of all data nodes share the same metadata, the data nodes have to maintain two schemas until all of them finish migration and modify the metadata, even some of them have finished. So we propose multi-version metadata architecture to minimize the adddtional workload during the migration. Fig. 1(b) illustrates the process of the data migration. The aqua colour stands for the data which need migrate.



EmployeeID	EmployeeName	Salary
2011	Tom	6000
2012	Billy	8000

EmployeeID	EmployeeName	Education
2101	Bell	Doctor
2102	Jim	Graduate

EmployeeID	EmployeeName	Phone
2001	Jack	01016598

GLOBALID	TenantID	EmployeeID	EmployeeName	COL1	COL2	---
1	Tenant1	2011	Tom			
2	Tenant1	2012	Billy			
3	Tenant2	2101	Bell			
4	Tenant2	2102	Jim			
5	Tenant3	2001	Jack			

TENANTID	TABLEID	COLUMNNAME	ISINBASIC	BCNAME
Tenant1	Employee	Salary	NO	
Tenant2	Employee	Education	NO	
Tenant3	Employee	Phone	NO	

GLOBALID	COLUMNNAME	VALUE
1	Salary	6000
2	Salary	8000
3	Education	Doctor
4	Education	Graduate
5	Phone	01016598

Fig. 1. (a) Example of BT&ET layout; (b) The process of data migration

3. Multi-Version metadata architecture

In order to minimize the workload during the migration in cloud data storage architecture, we propose the multi-version metadata architecture. Based on this architecture, we improve the data engine to fit the architecture.

3.1. The implementation of Mulit-Version metadata architecture

In the Multi-Version metadata architecture, all data nodes of a tenant don't share the same metadata. Each data node owns its metadata separately. So during the schema evolution, every data node can execute migration asynchronously, when the data node has finished, it can just modify its metadata, needn't to maintain two schemes and wait for others any longer. This will divide and decrease the additional workload which all data nodes generate when migrate simultaneously. Because of the migration on different data nodes is asynchronous, the tenant's data on different data node may have different metedata. That means there may be Multi-Version metadata to the same data on different data nodes during schema evolution. Fig. 2(a) outlines the Muilt-Version metadata architecture.

When controller receives a tenant's request, it gets the DB Node Metadata of the tenant first, and then gets the Data Metadata from every data node. The data engine generates different physical SQLs corresponding with every data node by its metedata. The router sends the physical SQL to its corresponding data node at last. Fig. 2(b) illustrates the improved cloud data storage architecture.

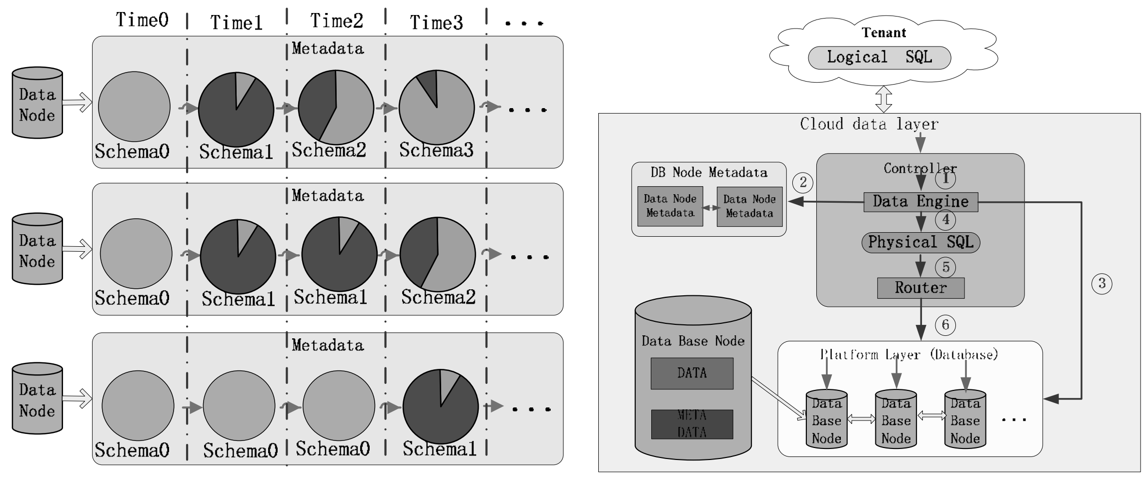


Fig. 2. (a) Muilt-Version metadata architecture; (b) The improved cloud data storage architecture.

3.2. The novel data engine

In our Multi-Version metadata architecture, every data node owns its metadata, so the data engine could support to rewrite a logical SQL into different physical SQLS according to different metadata. During the schema evolution, the metadata of every replica may be different for the migration is asynchronously.

If tenant accesses data during migration, the logical SQL should be rewrite into physical SQL by every replica's metadata. When schema evolution has finished, that means all replicas have finished the migration, the metadata of all replicas are the same. We just rewrite once and send it to all replicas instead of rewriting for every replica. So we modify the data engine in the following table.

Table 1. The implementation of data engine

```
N: a set of metadata of all data nodes of the tenant
S: a set of physical SQLs
T: a set of tables which the logical SQL refers to
PROCEDURE: getPhysicalSQLs(Logical_SQL，N){
   T ← getTablesByLogicalSQL(Logical_SQL);
   if(executingMigration(T)){
      for each n in N{
         s ← getPhysicalSQL(Logical_SQL, n);
         S=S + {s};}}
   else{
      n ← getMetadata(N);
      s ← getPhysicalSQL(Logical_SQL, n);
      S=S + {s};}
   return S;}
PROCEDURE: getTablesByLogicalSQL(Logical_SQL){
    return the tables which the logical SQL refers to; }
PROCEDURE: getPhysicalSQL(Logical_SQL, n){
    analyze the Logical_SQL;
    rewrite the Logical_SQL by metadata n into physical SQL s;
    return s;}
```

4. Experiment

In this section, we empirically evaluate the performance optimization in our multi-version metadata architecture and compare the performance with the original architecture. Our simulate cloud environment has 20 servers. 4 are controllers and the rest 16 are data nodes. Every tenant's data is stored on 3 data nodes, that means there 3 replicas for every tenant. We test the response time with the same requests during the schema evolution in two architectures. The flowing figures show the comparison of the response time when we send the same select, insert, delete and update requests.

The results of the experiments show the response time of the multi-version metadata architecture during schema evolution becomes faster. So we can guarantee the tenants' requirements on performance during the shema evolution better.

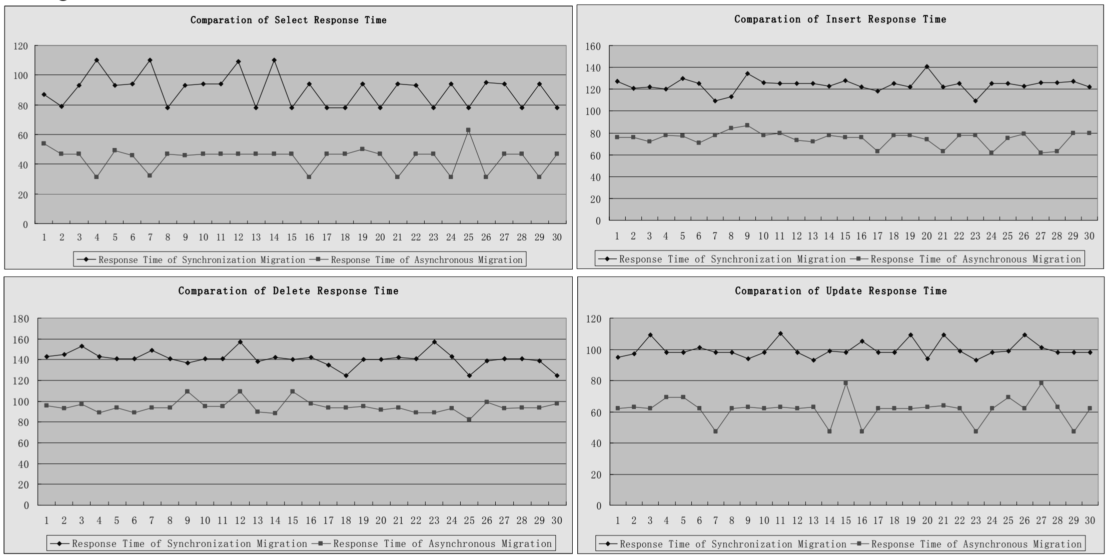


Fig. 3. The comparison of select, insert, delete, update response time.

5. Related work

In order to manage multi-tenant's data in SaaS, [1] proposes three solutions: the independent database, the shared database with independent schema and the shared database with shared schema to build a multi-tenant database. The currently popular data storage layouts for SaaS are described in [2, 3].

With the growing emphasis on cloud comping, some cloud data management systems have been propose, such as Google File System [5] and Hadoop [6]. The former is for web search and the latter is a distributed file system. Microsoft proposed Database-as-a-Service (DaaS) SQL Azure [7] in 2010, which is a cloud relational database based on SQL Server. Xeround Technical White Paper [8] outlines the SQL cloud database Xeround which is based on Mysql. It contains Access Node and Data Node. The Data Node is used for data storage while the Access Node is for receiving request, communicating with Data Node, computing and transferring request results. It can support the ACID absolutely. However, they all don't involve in schema evolution too much.

Sudipto Das et al propose ElasTraS in [9], which supports two common cloud databases: (i) large databases partitioned across a set of nodes, (ii) a large number of small and independent databases common in multi-tenant databases. Based on ElasTraS, they propose live data migration stategy with minimal downtime and impact on performace named Iterative Copy in [10].

6. Conclusions and future work

In this paper, we propose the Multi-Version metadata architecture. In the architecture, data migration can execute asynchronous between data nodes during schema evolution. So the additional workload will be divided and decreased and we can guarantee the tenants' requirements on performance during the shema evolution better.

Our shcema evolution means data migration between Basic-Table and Extension-Table merely. But during application upgrade, more complex schema evolutions will be involved, such as table combination, table division and so on. In our future, we will devote on how the Multi-Version metadata used in these complex schema evolutions.

Acknowledgements

This work is supported by the National Key Technologies R&D Program No.2009BAH44B02, the National Natural Science Foundation of China under Grant No.90818001, the Natural Science Foundation of Shandong Province of China under Grant No.2009ZRB019YT, No.ZR2010FQ026, No.ZR2010FQ010, No.2010GGX10105 and No.2010GGX10107.

References

[1] Mei Hui, Dawei Jiang , Guoliang Li, Yuan Zhou. *Supporting Database Applications as a Service*. IEEE 2009.

[2] Stefan Aulbach, Torsten Grust, Dean Jacobs, Alfons Kemper, Jan Rittinger. *Multi-Tenant Databases for Software as a Service: Schema-Mapping Techniques*. SIGMOD'08, June 9–12, 2008, Vancouver, BC, Canada.

[3] Stefan Aulbach, Dean Jacobs, Alfons Kemper, Michael Seibold. *A Comparison of Flexible Schemas for Software as a Service*. SIGMOD'09, June 29 – July 2, 2009, Providence, Rhode Island, USA.

[4] Wu Shengqi, Zhang Shidong, Kong Lanju. *A Dynamic Data Storage Architecture for SaaS*. MINES2010, NanJing, China; 2010.

[5] S. Ghemawat, H. Gobioff, and S.-T. Leung. *The Google file system*. SIGOPS Oper. Syst. Rev., vol. 37, pp.29-43, 2003.

[6] Hadoop. http://hadoop.apache.org/

[7] Inside SQL Azure,2010. http://social.technet.microsoft.com/wiki/contents/articles/inside-sql-azure.aspx.

[8] Database Scalability and Availability in the Cloud, 2010. http://xeround.com/main/wp-content/uploads/2010/11/Xeround-MySQL-Cloud-Database-Technical-White-Paper

[9] Sudipto Das， Shashank Agarwal， Divyakant Agrawal， Amr El Abbadi. *ElasTraS: An Elastic, Scalable, and Self Managing Transactional Database for the Cloud*. UCSB Computer Science Technical Report 2010-04.

[10] Sudipto Das, Shoji Nishimura, Divyakant Agrawal, Amr El Abbadi. *Live Database Migration for Elasticity in a Multitenant Database for Cloud Platforms*. UCSB Computer Science Technical Report 2010-09.