Tenant Oriented Lock Granularity Adjustment Strategy in the Shared Storage Multi-tenant Database

Yongqing Zheng, Chengliang Sang, Xiangxu Meng, Qingzhong Li

Abstract

SaaS is a perfect technology to achieve scale economy. It uses multi-tenant to allow database to share storage but with irregular load, untraceable transaction characteristics and so on. TOL mechanism provides one granularity between the upper coarse-grained granularity and the lower fine-grained granularity for each tenant. But the granularity may not be the most suitable. We define the cluster method and the diffusion method to support the granularity adjustment for a single tenant. Cluster is to make the granularity adjust towards coarser while diffusion gains opposite effect. Because TOL takes advantage of data distribution in multi-tenant database, we combine several aspects when the adjustment strategy is established, including basic table plus extension table and transactions feature. It focuses on locating a more suitable granularity size for one tenant between coarse-grained and fine-grained. In addition, we give experiments indicating that the lock granularity adjustment method is feasible and the adjustment strategy is effective.

Keywords: TOL, Granularity, Multi-tenant Database, Concurrency Control

1. Introduction

The main idea of SaaS (Software as a Service) is to use the software as a service and in the using process, several technologies, such as supporting multi-tenant, on-demand use and personalized customization and so on, are supported. Multi-tenant technology, a technology making integration of multiple applications to one application system, is a good choice for sharing of resources and economies of scale. In achieving sharing of resources for multi-tenant database, [1] D. Jacobs gave three ways, shared machine, shared process and shared table. In the three ways, shared data schema (shared table), the data of each tenant storing in the same schema in one or more database, owes the highest sharing degree. This paper is based on the shared data schema way in the multi-tenant database.

But in the face of multi-tenant database, especially the shared schema way, numerous problems present. Of course, many including security and schema mapping are not our concern. Since a single database can be used by many tenants and their data is mixed, a coarse-grained lock may let one tenant conduct lock on another tenant's data. This is a indirect way to access the other tenants' data and violates tenants' data right. In addition, tenants lock on each other can seriously sour the performance, so it is unendurable. What about fine-grained lock? Obviously, multi-tenant must engender more accesses coupled with so many locks which the lock manager needs to manage. It is no doubt that when the lock number reaches the bottleneck of the lock manager the database will present poor performance. Because of multi-tenant and fine-grained lock, the bottleneck is not hard to gain. At the same time, each tenant's application features and applications size may vary and each tenant's business difference is very large, so multi-tenant database's load shows strong irregularity and unpredictability and the transaction characteristics is of ever-changing. Therefore, it is better to explore a new granularity mechanism which can provide every tenant one granularity and can dynamically change its granularity size according to the tenant's transaction characteristics.

VGL[2] is just a wonderful mechanism. It fixes two boundaries on the upper coarse-grained granularity and the lower fine-grained granularity. According to the storage, it can locate the granularity between the two boundary, so it is convenient to provide special granularity. Realizing that it can provide one granularity for each tenant, we call the mechanism as TOL (tenant oriented lock). For TOL, each tenant's data storage distribution can decide the granularity, but for the tenant there is
one granularity that is more suitable than others, and the TOL may not be the one. We’d like to adjust the granularity towards more suitable for the tenant. We discuss several aspects, storage schema way and transaction characteristics. Two adjusting methods are cluster that makes granularity fine and diffusion that makes granularity coarse.

In this paper, methods for granularity adjustment, cluster and diffusion, are designed. Then combined with shared schema storage way and transaction characteristics, an adjusting strategy is given. Finally, the idea of this paper is simulated through experiments. It can be gained from experiments that the TOL mechanism combined adjustment strategy we proposed can significantly improve the performance of multi-tenant database.

The contributions of this paper:

- Realizing that the TOL mechanism can provide a single granularity for each tenant and the granularity may be not the best one, we define the methods of cluster and diffusion to support reasonable adjustments for the lock granularity of a single tenant. Cluster is for adjustment towards coarser granularity while diffusion is towards the opposite.
- We analyze the features of the basic table plus the extension table, and make use of load and transaction characteristics that may deeply impact the choice of lock granularity size. Then, we give the adjustment strategy for TOL.

The main structure of this paper is as follows. The second chapter is the related researches and work. The section 3 describes the architecture of multi-tenant database and introduces the shared data schema storage way. Chapter 4 learns the TOL mechanism. The next chapter defines cluster and diffusion to support a single tenant lock granularity adjustment. The granularity adjustment strategy is also described in this section. Section 6 is the experiment. Finally, it is the conclusions and future work.

2. Related Researches and Work


Gray originally describes the lock manager in 1978 and then Tay[15], Thomasian[16,17], Shasha and Bonnet[18] analyze the performance of lock in several aspects. Gray[19] deeply researches the granularity of locks and gives the hierarchical locks. Then he introduces the intention mode and recognizes five modes of access to a database resource and the compatibilities among them. PostgreSQL [23] is a free object-relational database management system under a flexible BSD-style license issued. It can support a variety of concurrency control techniques, the use of lock technology have two choices, table level and row (record) level. InnoDB [22] is an open source relational database storage engine embeds in MySQL, which supports transaction characteristics, achieves a record level lock. Berkeley DB [21] is an open-source embedded database system, and has a lot of advantages compared to similar well-known database. It blockades at the page level granularity to achieve concurrency control and transaction isolation. Berkeley DB provides reasonable defaults, developers may override them to control system performance. Recently, along with the demand of SaaS and cloud computing, several novel data store systems have been developed. MegaStore [8] provides strong consistency guarantees for the transaction processing. It blocks Entity. G-Store [7] designed a dynamic key Group, and in the stable state, the leader gain ownership of all fellower. It can block the key in the fellower.
3. System Model

3.1. Overall System Architecture

The overall system architecture is shown in Figure 1, where tenant app denotes the lease applications. The applications will have a variety of transaction requests. Routers route the transaction requests to corresponding DBMS nodes according to the routing metadata information. Monitor monitors all requests from all tenants and makes statistical analysis. Controller is response for the distribution of the tenants’ data according to the load on the DBMS nodes, so it may make a little changes on the routing metadata. At the same time, it will receive the monitors' static and execute adjustments. DBMS node stores tenant data. We let the DBMS nodes use RDBMS, so in the following sections, without specific reference, the database is aimed to the RDBMS storage environment. Thinking of reality, we give the DBMS nodes the following assumptions:

\[ K \times \text{Cap}(\text{Ten}(i)) < \text{Cap}(\text{Node}(j)) \quad 0 \leq i \leq n \quad 0 < j < m; \quad (1) \]
\[ \text{Data}(\text{Ten}(i)) \cap \text{Data}(\text{Ten}(j)) = \emptyset \quad 0 \leq i \leq n \quad 0 < j < n \quad i \neq j; \quad (2) \]

Among them, we assume that the system has n tenants, m DBMS nodes. \( \text{Cap}(\text{Ten}(i)) \) denotes the workload and capacity tenant i needs and \( \text{Cap}(\text{Node}(j)) \) stands for node(j)’s total capacity and maximum load. \( K \) represents a specific number, and according to the reality, it changes from several to dozens. The formula (1) shows that a Data Node can accommodate and process multiple tenants' data. That is to say, a single tenant's data capacity and load can not break up the capability of a single DBMS node. Of course, that is not only our reality. In Zephyr[26], they design the live migration for elastic cloud platform and they assume no failures and small tenants are limited to single node in the system. In formula (2), \( \text{Data}(\text{Ten}(i)) \) says the data of the tenant i. The formula (2) indicates that all tenants share data schema, but data between any two tenants disjoints.

<table>
<thead>
<tr>
<th>K * Cap(Ten (i)) &lt;Cap(Node(j))</th>
<th>0 &lt;= i &lt;= n 0 &lt; j &lt;m; (1)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Data(Ten(i))∩Data(Ten(j))=Φ</td>
<td>0 &lt;= i &lt;= n 0 &lt;j&lt;= n i &lt;&gt; j; (2)</td>
</tr>
</tbody>
</table>

Figure 1. Overall system architecture
3.2. Data Storage Model

For the shared schema way, there can be several implementations. We use the basic table plus the extension table [2]. Basic data of the tenants' applications is in the basic table, while extension and customization data is in the extension table. We make a little change from [2]. The data can be joined via the globally unique GlobalID. Of course, this needs the important information in the ColumnMetaData, such as the data type. This model lets tenants determine the extension degree of the data. While maintaining the advantages of shared data schema, it provides considerable flexibility. The basic table plus the extension table mode is shown in Figure 2, the tenant whose TenantID is tenant2 extends out column Gender.

4. Tenant Oriented Lock Granularity

VGL[2] is a balance between fine and coarse granularity for multi-tenant database under the shared schema. Here, we call VGL as TOL because it better describes its features. In TOL definition, R is the suitable lower granularity boundary in the system, but not smallest. In the relational database, it can be a data record but not always. PG(x) is considered as the suitable upper granularity boundary in the system, but not biggest. It is a collection composed by a number of Rs. Data(Ten(i)) is the data of tenant i.

$$\circ$$ is a mathematic relation defined on a single PG. R1 ⊪ R2 if and only if ($\exists x; R1 \in PG(x); R2 \in Data(Ten(i))) \wedge R \in PG(x), [R] = \{y | y \in PG(x), y \circ R\}$$  (3)

Formula (3) describes the TOL definition. In two extreme cases, if all records within PG(x) belong to a single tenant, the tenant granularity size is equivalent to page granularity; each record in PG(x) belongs to a different tenant, the tenant oriented granularity acts as record granularity. But in most case, it may be in a state between them. Figure 3 gives a vivid explanation of TOL taking page size as the upper boundary and record size as the lower boundary. In the figure, the left page can own five locks, just the same as record lock, while the middle page maximally owns 3 locks. The granularity for tenant 1 is between the page size and record size. The right page, with data in it only for tenant 5, gains a tenant oriented granularity with same granularity as page size. Therefore, we can see that as one situation that TOL provides each tenant a special granularity lock. It is an average granularity for the tenant. Essentially speaking, from the perspective of tenant, the TOL gives every page of the tenant one granularity and in the view of page, it gives every tenant in the page one granularity.
5. Single Tenant Granularity Adjustment

5.1 Cluster and Diffusion

TOL ensures that each tenant uses a granularity to assure transaction isolation. In different circumstances, we should use different lock granularity to provide better performance. Thus, a problem arises naturally: what if the tenant is using a granularity that is not suitable?

Before exploring the problem, we give several denotations about TOL.

- Ten(i): tenant i in multi-tenant database.
- Data(Ten(i)): the data of tenant i in multi-tenant database.
- PG(x): PG is for the upper boundary granularity size, and it is the number x.
- R: the lower boundary granularity size.
- Tran(i): transaction i in multi-tenant database.
- DATA(PG(x)Ten(i)): data in PG(x) belongs to Ten(i).
- DATA(Ten(i)Tran(j)): data is accessed by Ten(i)’s Tran(j).

TOL just makes use of the feature that multi-tenant share data storage in multi-tenant databases to achieve the characteristic of variable tenant oriented granularity. For tenant Ten(i), the distribution of the data Data(Ten(i)), DATA(PG(1)Ten(i)) DATA(PG(2)Ten(i)) to DATA(PG(b)Ten(i)), in the corresponding upper granularity, PG(1) PG(2) to PG(b), determines the lock granularity of the
transaction of this tenant. Therefore, we can manage the distribution of the tenant's data in each page to adjust lock granularity of the tenant transaction. Review the definition of TOL. It uses ◎ to partition PG(x) to partition pieces. The essence of adjustment of the data in a PG is to adjust R number in this PG. However, as is known, the system I/O is the key performance bottleneck in current database system. it is not wise to adjust tenant's records in a page on hard disk which will bring about additional I/O operations. Taking that all data would be cached, we make it a rule that for the adjustment of tenant's granularity, we just adjust PG accessed by transactions and in the cache of the database. In this way, the cost of the adjustment is relatively low. As for adjustment, we introduce two approaches: cluster and diffusion.

Definition of cluster: Suppose that data accessed by transactions Tran(j) of tenants Ten(i) is distributed in m PG(x), DATA(Ten(i)(Tran(j)))= ∪ DATA(PG(x)Ten(i)) (1<=x<=m), let R ∈ DATA(PG(x)Ten(i)) (m>=x>=h h<=m) congregate towards PG(x)Ten(i) (x<=h), and after cluster

DATA(Ten(i)(Tran(j))) = ∪ DATA(PG(x)Ten(i)) (1<=x<=h).

Definition of diffusion: Suppose that data accessed by transactions Tran(j) of tenants Ten(i) is distributed in m pages, DATA (Ten (i) (Tran (j))) = DATA(PG (x)Ten (i)) (1 <= x <= m), randomly select PG(x) from the cache, PG (x) (x> m, PG (x) Ten (i) = Φ), let R ∈ DATA(PG(x)Ten (i)) (1 <= x <= m) spread to PG (x) (x> m, PG (x) Ten (i) = Φ), and after the diffusion DATA (Ten(i)(Tran (j))) = ∪ DATA(PG (x) Ten (i)) (1 <= x <= h, h> m).

From the definition, it is obvious that cluster transfers data of tenant Ten(i) from more PG to less PG while diffusion gains opposite effect. That is, cluster can make the lock granularity move towards coarser and diffusion fines the lock granularity. Figure 4 is a simple example of cluster and diffusion for the red color tenant 1. From top to bottom is description of the cluster and bottom to top is the Diffusion. A concern is that adjustment for one tenant may cause cascading adjustment for other tenants. However, in multi-tenant database, there are so many pages and a page also contains a lot of data records, part of a tenant data adjustment has little impact on other tenants.

5.2 Adjustment Strategy

Note that in order to ensure a shared data schema model, we use the previously described basic tables plus extension table for data storage. Basic table stores data in original applications' columns, while extension table is for extension data and customization data. For most tenants, the number of transaction requests for basic table is far greater than for extension table. that is, for such model, the basic table is hot spot data. What is more, record in extension table is just equivalent to a column value of one record in the basic table. Accessing the same data in extension table will cover far more number record than in the basic table. And because the extension table relies on the column metadata table to gain metadata, the query efficiency is relatively low. It is obvious that extension table is not suitable for highly concurrent execution transactions. In such a situation, we can use fine-grained lock for basic table and coarse-grained lock for extension table.

Specifically, for the lock granularity adjustment, part of the data R within a single tenant congregates towards fixed PGs. It brings this tenant more coarse-grained granularity. Part of the data R within a single tenant spread to multiple PGs. The granularity of the tenant lock will grow fine-grained. The transaction and the load in the multi-tenant database have obvious irregularities, but in terms of a single tenant, within a period of time, a certain feature of its transactions and load can be gained. Review the system architecture components. The monitor keeps track of all tenants' transaction requests. For a tenant, we can get the tenant's statistics, such as the data amount for single transaction and the number of the request per unit time. Aimed at the features of the tenant's load and transaction, we perform the corresponding adjustment in the follow-up requests. Just as the analysis for TOL, if full of large transactions and low concurrency, grow up the granularity of TOL, while on the other hand, cut down the granularity to support high concurrency. Transactions need to adjust the data it accesses, so the adjustment may be evolved and cost some time. With the ongoing accesses of transactions, the tenant will keep the most frequently accessed data to the ideal granularity. In this case, the granularity of TOL is a balancing point between upper granularity and lower granularity.
Now, we integrate storage properties and transactions characteristic in TOL. A detail adjustment strategy for TOL is given in Table 1. We can conclude that, if the transaction features are inclined to a granularity. We will take this for main factor to adjust the granularity for TOL. If the transaction is not obvious to support a specific granularity, we will consider the basic table plus extension table just as mentioned above.

### 6. Experiments

TOL is designed for multi-tenant database. To achieve this environment, we use the data we got in task 2 in our project. There are several applications including agency management services, payment management services, expert management services and examination management services and so on. They are all deployed to SaaS application by PaaS platform, and leased out to tenants through PaaS platform. For example, examination management services may be leased by the health industry and transportation industry, and different tenants provide services for their users. The number of users under the tenant varies from dozens to hundreds. After each application deployed to PaaS platform, the application data are stored in the basic table plus extension table model mentioned previously. During the deployment, disorder the data in the basic table and order the data in extension able by tenants to ensure the basic table fine-grained and extension table coarse-grained. After repeated testing, we use one DBMS to provide service for 15 tenants. In order to simulate the irregular load and transaction characteristics, we change the number of the user and transaction characteristics. The data is 1.4G for the DBMS node, so each tenant's data is about 100M.

![Figure 6. Result of Experiment 1](image-url)
storage concept, including table space, section, page and record, etc. This provides convenience for achieving several lock granularity. In the original InnoDB, the basic structure of the page contains File Header, Page Header, Infimum Supremum Records, Page Directory and other information. Page Header mainly contains record information. Page Directory contains a variable number of pointers, and InnoDB assigns each 6 records a pointer, so every time to locate the record position needs local traversal. Note that this paper attempts to explore lock granularity. In order to reduce the unrelated impact factors, in the implementation of the data record scanning, regardless of the scanning result, the entire page of data should be scanned. In order to compare, we set page as the upper granularity size and record in mysql as the lower granularity size. PGL (page granularity lock) is achieved as long as one transaction access a record in the page, a lock will be performed on it. For TOL, you need to add a PAGE_TENANT into PageHeader variables to support the partition pieces in the page. For RGL (record granularity lock), use the original method, but every time scan the entire page, which may bring additional costs, but is not our concern, we aim at ensuring fair comparison of several granularity. In addition, InnoDB achieve a clustering B-tree index, in order not to affect the individual tenants during the lock granularity adjustment, we add a gloablrecordid column to all tables as their main indexes. It is to prevent tenants sign in the main index.

This section conducts two experiments, as Experiment 1 and Experiment 2. These two experiments are designed to assess the granularity adjustment in the TOL for tenants. Experiment 1 simulates the granularity to coarsen. Experiment 2 let the granularity fine. In this paper, although the proposed idea is to adjust the granularity of a single tenant, in the experiment, we directly regulate the entire data node, from a broader perspective to observe the adjustment. 10*TPS is used to measure system throughput. The system throughput is defined as: TPS = (Com_Commit + Com_Rollback)/Seconds. TPS stands for the Transaction Per Second, Com_Commit and Com_Rollback is the MySQL variables, namely the number of transactions submitted and rolled back.

In experiment 3, focusing on coarsening process, we fix the number of users in 800 under 15 tenants, and then adjust the characteristics of all transactions to large transactions (the amount of data accessed by single transaction). Then in five minutes time, begin to adjust the lock granularity of TOL, so the data start clustering. Wait the overall system throughput to step into stability and stop adjustment. The result of Experiment 1 is shown in Figure 6. In Experiment 2, we fix the number of users in 1500 under 15 tenants. Adjust the characteristics of all transactions to small transactions. Also in 5 minutes and then begin to adjust the granularity. The result is shown in Figure 7.

Figure 6 shows that, this type of transaction, just as the analysis in this article, is suitable for coarse-grained lock. During the beginning time, PGL is the best one, and then is TOL, the last is RGL. During the adjustment time, due to the need for clustering data, there is a great performance impact. Over time, performance is becoming better. With the ongoing adjustment, less and less data need to be adjusted, and the performance keeps rising. In the last stable phase, it is very close to the performance of PGL, compared to the beginning performance of itself. The performance improves a lot. Experiment 2 described in Figure 7 is similar to Experiment 1 but in a opposite adjust direction, and in Experiment 2, the fining process and the adjustment phase is longer than the Experiment 3.
7. Conclusion and Future Work

In this paper, based on the shared data schema way in multi-tenant database, we put forward cluster and diffusion method to improve TOL. Then, we establish adjustment strategy that take the way of data storage and characteristics of tenant's transactions into account. Finally, the paper presents an experimental evaluation of the method in the paper.

TOL is more suitable for multi-tenant database, but the adjustment depends on records adjustments. Just as shown in experiments, we just adjust the granularity towards to a direction. A profound analysis for TOL is need and a transaction model in multi-tenant is also need to support TOL. What is more, a intelligent model and algorithm are needed to locate the most suitable granularity for the tenant. In the next study we will explore these issues.

8. Acknowledgement

Research was supported by: (1) the National Key Technologies R & D Program No.2009BAH44B02; (2) the Natural Science Foundation of Shandong Province of China under Grant, No.2009ZR019YT, No.ZR2010FQ026, No.ZR2010FQ010, No.2010GGX10105, No.2010GGX10107 and No.2012GGE27036; (3) Independent Innovation Foundation of Shandong University under Grant No. 2012TS075.

9. References


