The Problem of Creating Multi-Tenant Database Clusters[*]

Evgeny A. Boytsov
Valery A. Sokolov
The faculty of the computer science,
Yaroslavl State University,
Yaroslavl, Russia
{boytsovea, valery-sokolov}@yandex.ru

Abstract **— SaaS (Software as a Service) paradigm brings both lots of benefits to end users and lots of problems to software developers. One of such problems is an implementation of a data storage which is able to satisfy needs of clients of a service provider, at the same time providing easy application interface for software developers and great opportunities for administration and scaling. This paper provides a brief review of existing problems in the field of organizing cloud data storages that are based on the relational data model and proposes the concept of architecture of RDBMS cluster dedicated to serve multi-tenant cloud applications.**

***Keywords** - databases, SaaS, multi-tenancy, scalability.*

I. Introduction

One of the most notable tendencies in the modern software development industry is the shift to Software as a Service (SaaS) paradigm. The main ideas of this approach are the following:

- An application is developed as a system of distributed services interacting with each other.
- All computing power and infrastructure needed for operating an application is supplied by a service provider.
- A fee for an application is taken on the basis of actual usage.

This approach is one of cloud computing delivery models. The main advantage of its usage for customers is that all expenditures for deploying infrastructure required for correct and stable operation of software suit are taken by a service provider. This fact should eliminate the need for a customer to have his own IT staff and purchase a new computer equipment with every new release of an application. Besides, this approach allows to completely solve the problem of software updating, because now it is done in a centralized manner by the software company itself, that means that all customers always use the most recent (i.e. the most safe and featured) version of the application.

However, the «jump into clouds» brings not only benefits, but also new problems mostly for developers and administrators of such systems.

It is known that most of enterprise-level applications are based on interaction with relational databases. Any customer relationship management system, docflow or enterprise accounting management system needs to store somewhere data used by the application . The de-facto standard for such data storages are RDBMS. In recent years there was a tendency to move most of the application logic to the database tier expressed in appearing procedural extensions of the SQL language, in which developers wrote stored procedures and packages of them. Modern RDBMS are able to process very large arrays of data, fulfill very complex data selection and data manipulation queries. The complexity of most powerful such systems is comparable with the complexity of modern operating systems. Some companies even create a specialized hardware for them. Most software development specialists are familiar with the SQL language and principles of data organization in RDBMS.

A typical scenario of RDBMS usage in traditional on-premise application is the following. A developer describes a required data structure (tables, indexes, views, constraints e.t.c.) and selects a database server which is the most appropriate for a «scale» of the task being solved and capability requirements (the choice here is very wide, from embedded libraries, like SQLite, to powerful enterprise-level distributed database clusters, like Oracle or DB2). After the choice has been done, a developer team defines an application logic based on interaction with the database server, and the application with the database server is installed at the customer's hardware. The application can be tuned in place to satisfy special needs of the customer. The most advanced applications are able to extend their data structure in place and to add an additional logic, usually using one of the script languages for such purposes. System administrators configure the user rights and requisites and upload necessary initial data to the database. After that the application is ready for use. In the context of the discussed problems there are two key moments of this process:

- Data and a database server are hosted by a customer, thus their safety and availability (except the case of data corruption by the application itself) are under responsibility of customer's IT-staff.
- Access rights are granted at the database level by configuring users and departments rights inside a company.

In the case of a cloud application, data are hosted by (and thus are under responsibility of) a service-provider which undertakes to provide instant and fast access to them for tens or

* This work is supported by the RFBR grant # 11-07-00549-a.

hundreds of thousands of its clients concurrently. Violation of any of these requirements (speed and availability) will cause penalties to the service provider and, that is much more important in the cloud industry, will worsen the image of the provider. A typical service level agreement for a cloud service guarantees its availability of 99%. These circumstances lead to the following conclusions.

- Maintenance of a cloud application implies large expenditures to organization of the data storage, caused by the need to store (most likely, in multiple instances to guarantee a required speed of access) data of hundreds of thousands of clients and their backup copies in order to restore in case of failure.
- Maintenance of a cloud application implies large expenditures on support of infrastructure, required to store data. This includes organization of monitoring resources availability, efficiency of their usage, backuping and restoring, organization of application data structure update and much more.

These considerations, in turn, lead to the following conclusion: an attempt to solve problems of scalability and management of a storage infrastructure using the existing technologies (which were used for designing and development of traditional «boxed » applications) would require very large expenditures, thus drastically limiting a barrier of entry to cloud business. That is why a common desire of SaaS vendors is to minimize costs of data storing and to find architectural solutions that would lead as much as possible to such a minimization without compromising performance and functionality of the application,.

One of such solutions is a multi-tenant application (thus also database) architecture. The main idea of this approach is to share one instance of the application among many tenants (companies subscribed to the service), thus drastically reducing expenditures to application servers, web-servers and associated infrastructural elements. An application design according to such architectural principles imposes some restrictions to functionality, but it brings unprecedented opportunities to scale the solution. Every instance of an application server lives «from request to request», without saving its state in internal data structures that allows, having sufficient physical (or virtual) computing power, to set up an unlimited amount of application instances to serve clients.

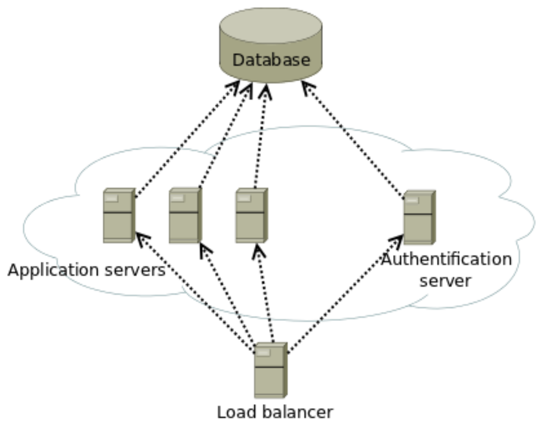

Figure 1. Simplified typical SaaS architecture

However, these considerations do not apply to database servers. Figure 1 shows a simplified architecture of a typical SaaS application, but even in this diagram we can see that a database server is the first candidate to become a bottleneck as the system grows. The reason for this lies in the fact that in contrast to application servers database servers scale poorly. To be more precise, application servers are able to scale well just because they descend most of load to the level of database servers, just often generating SQL queries and performing simple post-processing of the result. The database server should provide a reliable data storage, fast access, transactional integrity and others. A trend in recent years, when the most of the application logic moved to the database level, increased the load on this component of the system even more. In traditional systems the problem of scaling is less crucial, because a typical modern RDBMS in a single instance is able to serve needs of a medium company or even a fairly large one. Even if the problem of scale appears, it can be solved by formation of database clusters which consist of some powerful servers with a huge disk storage and database partitioning. However, this approach is not applicable to cloud solutions, because the total amount of data of all customers and the number of different queries, that they have to perform, are beyond the capabilities of any modern cluster.

II. Modern ways of organizing a multi-tenant architecture in cloud solutions

There is some experience in a field of organizing cloud data storages [1,2,3]. There are both completely new technologies based on absolutely new approaches and architectural solutions that allow to get multi-tenant environment in traditional RDBMSes.

A. *NoSQL databases*

The usage of NoSQL databases is one of the approaches that is widely used in the software development industry to organize a cloud data storage. This term denotes a wide range of technologies and solutions, the main characteristics of which is the emphasis on speed and ease of scalability at the expense of other functionality. The data in such databases are stored in separate records which are not limited by a predefined format, there are no terms like «NoSQL database normalization» or «NoSQL database data structure», in fact, these databases are just key-value storages. Such systems are simpler than traditional relational databases and provide a relatively small amount of data types and abilities to manage them. But the lack of features is compensated by very high performance and unlimited scalability. High performance is achieved by the simple structure and data indexing, scalability — by absence of complex data arrays and relations. Such databases can be used when a complex selection of large data arrays is not required and the application logic is based on the processing of individual objects or documents.

B. *Multi-tenancy in RDBMS environment*

In the case, when the application logic requires the complex processing of data, transactional integrity and other features that are currently unavailable in modern NoSQL databases, software developers have to return to the usage of traditional RDBMS. At the moment, there are two main approaches to designing multi-tenant relational database.

- Usage of shared tables for all clients with the attachment of a tenant identifier to every record — this is the shared table approach.
- Creation of the own set of tables for every tenant (usually, these tables are united into one database schema) — this is the shared process approach.

Both approaches have their pros and cons.

1) Shared table approach

This approach is the most radical one in answering a question of sharing server resources. The usage of this approach requires adding a special column to every table in the database which stores a tenant identifier for distinguishing data of different clients. Every SQL query to the database of such an architecture should be supplemented with an additional WHERE/HAVING predicate that leaves in the query result only records that belong to a specific client. There are also some projects of SQL extensions [4] that allow adding such predicates automatically, but at the moment these extensions are only concepts under development and research. The advantages of the shared table approach are the following:

- better usage of a disk space;
- small size of a data dictionary;
- better usage of a query planner's cache (i.e. shorter time of query analyzing and generation of its execution plan).

This approach has some disadvantages.

- The enlarging of the size of database tables and their indexes [5]. This drawback results in the requirement of very high qualification of developer of database queries, because any query for which database query planner is not able to generate an effective, indexes-based execution plan, will «hang» a database server for a large period of time. The most unpleasant thing in this situation is that usually the size of data of one tenant is relatively small and the query that operates on a data set with millions of records would return just a pair dozens of rows.
- The need to add the predicate of selection of a current tenant's data. This drawback leads to access errors, when users of one tenant can see data of another tenant. Such errors may be very destructive for the reputation of a service provider .The above [4] concepts of extensions of the SQL language are possibly able to solve this issue.
- The complexity of replication and backup copying of separate tenant's data. Since a tenant's data are «smeared» across the database, it is very hard to develop a generalized mechanism of their extraction and recovery.

In general, this approach shows good results, when application data schema is not large (there are not many tables) and a typical query is relatively simple (it does not contain joins of tens of tables, complex orderings and grouping, nested subqueries). If the above conditions are met, this approach to designing a database allows the most effective usage of a disk space and other hardware resources, storing data of tens of thousands of tenants in a shared database.

2) Shared process approach

This approach occupies an intermediate position in solving a problem of sharing server resources between complete isolation of a tenant's data in a separate database and a shared storage of them in the shared table approach. The separation of the tenant's data is achieved by creating its own set of database objects (tables, views, e.t.c.) for each tenant. The advantages of this approach are the following.

- Unification of the code of database queries and ease of writing new ones. In contrast to the shared table approach, queries are known to operate only the current tenant's data which usually have a relatively small size and therefore do not require a lot of memory and other database server resources for their execution [5].
- Relative ease of backup copying and replication of data of a single tenant. Because the tenant's data are grouped together in their own schema, they can be easily separated from others.
- Decrease of data security risks. It is much harder to make a mistake when writing a SQL query so that users of one tenant could be able to get access to data of another one.
- Simplification of system administration. Since a single tenant's data are a small logically separated database, they are human-readable and can be analyzed and corrected by standard database administration tools, if needed.

But there are also some drawbacks of this approach.

- Usage of this approach makes data dictionary of a database very large and heavyweight. Database metadata tables contain millions of records and are expensive to access. Every new database object makes them even larger.
- Because of a data dictionary enlargement, a query planner is unable to use its cache effectively, that makes him to generate a new plan of execution for almost every incoming query [4].
- A disk space is used less effectively than in the shared table approach [4].
- If a data structure change is required, it will take a very long time to complete, because of the large amount of database objects.

In general, this approach shows good results (to be more precise, there is no real alternative), if an application data structure is complex (contains a lot of objects) and a typical query selects data from a large set of tables, makes nested subqueries and other complex data manipulations.

III. Limitations of existing approaches and goals of the research

Despite the fact that they are not directly supported by most of database engines, both approaches are successfully used by the software development industry. However, generated databases are very large and complex and therefore they are hard to manage. But every cloud application that aims to have a large user base has to operate on dozens of databases of such a complex structure. It is physically impossible to place all clients into one database. The highest level of database resource consolidation known today is about 20 000 tenants in one database with a relatively simple data structure. A simple calculation shows that even with such a high degree of resource consolidation, a company would require 50 database servers to serve 500 000 tenants, storing one backup copy of data for each of them for load balancing and data protection against failures and errors. In reality, such system would require much more database servers.

But the quantity of database servers is not the only problem in organization of a cloud cluster. Even more significant point is the load balancing for optimal usage of computing power and disk space at the entire cluster level. The nature of a cloud application is that the load on it is unpredictable, it may rise and fall like an avalanche and "burst" of activity can occur from a variety of tenants. The load may account for both the CPU and the network adapter of database servers, which occurs when the activity of tenant users increases, and on its disk drives, which will inevitably occur with the data accumulation. To provide the required level of service, an application should be able to dynamically adapt itself to changing conditions by automatically redistributing available resources. At the moment, there are no software systems that are able to solve this problem, as there are no clear requirements and approved algorithms for them. This research has the following goals:

- Development of algorithms of load balancing for multi-tenant cloud database clusters.
- Research of developed algorithms for correctness and safety, including imitation modelling and stress testing.
- Development of complex solution for organizing multi-tenant cloud database clusters using ordinary servers.

IV. Solution requirements

The developed system will be intended for using by small and medium-sized software companies, and therefore it should be designed to meet their needs and capabilities. The following points can be noted:

- Reliability and maximum guarantee of data safety. The reputation is extremely important for a cloud service provider, because his clients trust him their data and for many of them this is a rather difficult decision. This decision will become much more difficult to make, if there are clients' data corruption or loss.
- Efficiency. A multi-tenant cloud cluster management system should use available resources in the most efficient way, providing maximum performance of an application.
- The similarity to traditional DBMS with minimal possible corrections for the cloud application specifics. It is known that most software developers have some experience with traditional RDBMS and the SQL language and know main principles of the relational data model. This is the significant benefit that should be used.
- Horizontal scalability. Horizontal scalability is the ability of a system to increase its performance by adding new servers to existing ones. This differs the horizontal scalability from the vertical scalability, when performance of the system is increased by upgrading the existing servers. The horizontal scalability is preferred to vertical, because it is cheaper and potentially allows to infinitely increase the performance of the system.
- Ease of administration. If a service provider aims to serve lots of clients, it has to deal with a very large and complex infrastructure and its manual administration can lead to management chaos and system unmaintainability. To avoid this, the cluster management system should provide maximum automatization and tools for real-time system monitoring.

Let us list the main characteristics of multi-tenant databases for cloud applications, which should be taken into account, when designing a cluster management system:

- The huge aggregate size of stored data. As the provider is to serve dozens and hundreds of thousands of clients, the total amount of data which it is responsible for is huge and constantly growing with an unpredictable speed.
- The small size of a single client data. Since we consider multi-tenant solutions, this solution is likely to aim at dealing with small and medium-sized companies (so called «Long Tail» [1]), and therefore the number of users and the size of data of an average tenant is not large.
- The presence of shared data. Usually any cloud application has some set of data which is shared among all tenants of the provider. Such data include various directories and classifiers. Accessing them is almost always read-only. Such data should be stored in a single copy, providing their replication to all database servers in a cluster.
- The need for data backup.
- The need for data replication. Like in traditional DBMS, data replication is used to balance the load of database servers. The distinction is that often the replication in cloud solutions is partial, i.e. only the data of one or some tenants are replicated.

Based on these requirements and features, we present the proposed project of the cloud cluster management system.

V. The architecture of multi-tenant database cluster management system

When designing the cluster management system, it is important to think about the comfort of its users in advance. There are two categories of users of the system: application developers and system administrators. The first category deals with the system API to create their applications based on multi-tenant cluster technologies. The ideal interface for this category of users should ease the interaction with the system and hide cluster implementation details behind an additional layer of abstraction. From the point of view of a software developer, the cluster should be considered as a set of independent databases for each tenant with a single entry point (i.e. database connection string). A desired application interface should be the following:

```
Connect( params );

Execute( ClientID, «SELECT * FROM T1» );

Disconnect();
```

As for the second category of users (system administrators), the most important characteristics of the system are the ease of configuration and monitoring. Ideally, the configuration of the system should be as simple as possible and should be limited to specifying the physical structure of a cluster. Monitoring tools should provide reports that show the current state of the system in maximum detail.

The main idea of the proposed solution is to add a new layer of abstraction between application and database servers, functions of which are:

- The routing queries from an application server to an appropriate database server by the tenant identifier.
- Management of tenant data distribution among database servers, dynamic data redistribution according to an average load of servers and characteristics of different tenants activity in time.
- Management of data replication between database servers in the cluster.
- Management of data backuping.
- Providing a fault tolerance in the case of failure of one or some cluster databases.
- Analysis of resource usage and system diagnostics.

The system should be implemented as a set of interconnected services using its own database to support a map of the cluster and collect statistics on the system usage by tenants and characteristics of the load. The shared process approach is going to be used for tenants data separation at the level of a single database. The choice of this approach is explained by the fact that the system must be sufficiently general and have no knowledge about the data structure required for an application beforehand. Because one of the requirements of the shared table approach is to add a service column to every table in the database, it assumes much closer familiarity with the application data structure and thus its usage is difficult for the generic system. Moreover, the usage of the shared table approach requires very good query optimization skills and thus does not hide the underlying structure of the cluster from the developer. The general architecture of the proposed system is shown in Figure 2.

We proceed to a more detailed consideration of the above-mentioned functions of the system. The proposed solution assumes the appearance of a new element in a chain of interaction between application and database servers. This new

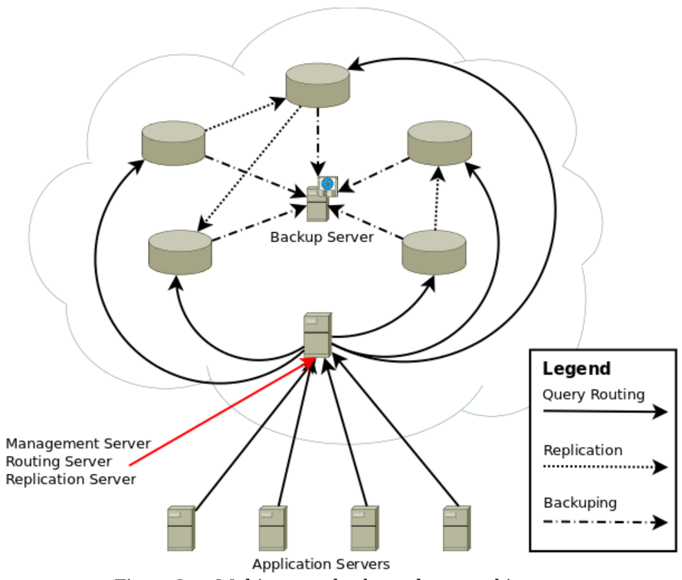

Figure 2. Multi-tenant database cluster architecture

element is a dedicated server which transparently for application servers routes their queries to an appropriate database server, basing on a tenant identifier, provided with a query, characteristics of the query and statistics of the current system load. This is the component application developers will deal with. In fact, this component of the system is just a kind of a proxy server which hides the details of the cluster structure and whose main purpose is to find as fast as possible an executor for a query and route the query to him. It makes a decision basing on the cluster map.

It is important to note that the query routing server has a small choice of executors for each query. If the query implies data modification, there is no alternative than to route it to the master database for the tenant, because only there data modification is permitted. If the query is read-only, it also can be routed to a slave server, but in the general case there would be just one or two slaves for a given master, so even in this case the choice is very limited.

Besides, it is important to mention that the discussed component of the system can not use expensive load balancing algorithms, because it must operate in real-time. All it can use is its own statistics on the number of queries sent to a specific database server of a cluster, the execution of which has not yet been completed. Basing on this runtime data, it must decide where to send the next query.

The implementation of this component should give a significant benefit in performance and ease of cluster administration.

The second component of the system is the replication and backup management server. Its functions are clear from the name but it is important to note that, unlike the traditional databases, in multi-tenant solutions the replication is almost always partial, i.e. only some part of data is replicated. For example, the data from the first tenant schema can be replicated to one database, from the second tenant schema — to another and the third tenant schema itself can be a replica of a part of the second database from a cluster. Once again we recall that the data change request can only be executed by the master database of the tenant and this consideration should be taken into account during the distribution of tenant data among servers to avoid hot spots.

The third component of the system, its peculiar "circulatory system", is a set of agent-services placed at the same machines as database servers. These small programs-daemons are to collect statistics about the load of the server (usage of CPU, RAM, network adapters and disk space) and monitor server state in the case of failure. All the information collected is sent to a central server for processing and analysing and will be used as an input data for the load balancing algorithm.

The last and the most important and complicated component of the system is the data distribution and the load balancing server. Its main functions are:

- initial distribution of tenants data among servers of the cluster during the system deployment or addition of new servers or tenants;
- collecting the statistics about the system usage by different tenants and their users;
- analyzing the load on the cluster, the generation of management reports;
- management of tenant data distribution based on the collected statistics, including the creation of additional data copies and moving data to other server;
- diagnosis of the system for the need of adding new computing nodes and storage devices;
- managing the replication server.

This component of the system is of the highest value, since the performance of an application depends on the success of its work. The key indicators that can be used to evaluate its effectiveness are:

- the average response time of a service (an average time between the arrival of a request and receiving a response to it);
- availability of a service (what percent of requests from the total number was successfully executed, what percent failed to meet a time limit or other parameters and what percent is not executed at all);
- the average load of database servers (whether servers are equally loaded, whether there are idling servers when others fail to serve all requests).

Obviously, the last criterion affects the previous two.

An algorithm of cluster load analysis and need for data redistribution should become the core of the load balancing system. This algorithm should make its decision about data redistribution, taking into account the following considerations.

- Performance of cluster servers. If the system is not homogenous, the proportions of its parts should be taken into account.
- Free resources available. If the system has free resources in its disposal, it makes sense to use them by creating additional copies of tenant data to increase the performance of the application. However, if the number of tenants begins to grow, the created redundant copies should be removed.
- The history of the individual tenant activity. If users of the tenant A actively use the application and users of the tenant B do not, it makes sense to move the data of the tenant B to a more busy server and create less copies of them, since they are unlikely to cause problems for the service.
- Preventing the creation of hot spots on writing the data in a context of replication organization. The system should distribute master servers for all tenants in an appropriate way, taking into account the history of the tenant activity.

Such an algorithm can be based on a variety of strategies, and currently it is not clear, which of them should be preferred. From this it follows that the most reasonable solution would be to implement several variants of the algorithm as a pluggable modules and choose the best one according to the results of imitation modelling and stress testing. It is very likely that such a version will not be found and the final implementation of the system will contain several versions of the algorithm which showed themselves as the best ones under some external conditions. In this case the choice of an appropriate version of the algorithm will be the task of a cluster management system administrator.

[1] F. Chong, G. Carraro, “Architecture Strategies for Catching the Long Tail“, Microsoft Corp. Website, 2006.

[2] F. Chong, G. Carraro, R Wolter, “Multi-Tenant Data Architecture“, Microsoft Corp. Website, 2006.

[3] K.S. Candan, W. Li, T. Phan, M. Zhou, "Frontiers in Information and Software as Services", in Proc. ICDE, 2009, pages 1761-1768.

[4] Oliver Schiller, Benjamin Schiller, Andreas Brodt, Bernhard Mitschang, “Native Support of Multi-tenancy in RDBMS for Software as a Service“, Proceedings of the 14th International Conference on Extending Database Technology EDBT '11 2011

[5] D. Jacobs, S. Aulbach, “Ruminations on Multi-Tenant Databases“, In Proc. of BTW Conf., pages 514–521, 2007.