Multi-tenant Database Clusters for SaaS

Evgeny Boytsov and Valery Sokolov
Department of Computer Science,Yaroslavl State University, Yaroslavl, Russia
{boytsovea, valery-sokolov}@yandex.ru

Keywords: Databases, SaaS, Multi-tenancy, Scalability.

Abstract: SaaS paradigm brings both many benefits to end users and many 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.

1 INTRODUCTION

One of 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.

The main advantage of this development approach 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 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 an 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. The de-facto standard for such data storages are RDBMS. In recent years there was a tendency to move the most of application logic to the database tier expressed in appearing procedural extensions of the SQL language. Modern RDBMS are able to process very large arrays of data, fulfill very complex data selection and data manipulation queries. Most of software development specialists are familiar with the SQL language and principles of data organization in RDBMS.

In traditional on-premice applications data and a database server are hosted by a customer, thus their safety and availability are under responsibility of customer's IT-staff. 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 hundreds of thousands of its clients concurrently. Non-fulfillment of any of these two requirements (speed and availability) would cause penalties to the service provider, and, that is much more important in the cloud industry, would worsen the image of the provider. A typical service level agreement for a cloud service guarantees its availability of 99% (Candan et al., 2009). Thus, maintenance of a cloud application implies large expenditures to organization of the data storage, caused by a need to store data of hundreds of thousands of clients and their backup copies in order to restore in case of failure. These expenditures can drastically limit a barrier of entry to cloud business and decrease provider's profits. 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 an 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 and allows, having sufficient physical (or virtual) computing power, to set up an unlimited amount of application instances to serve clients.

However, these considerations do not apply to database servers which are the first candidates to become a bottleneck as the system grows. The reason for this lies in the fact that, in a 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, often just generating SQL queries and performing simple post-processing of the result. The database server should provide reliable data storage, fast access, transactional integrity and much more. 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. The total amount of data of all provider's customers in cloud solutions and the number of different queries that they have to perform, make traditional database scaling techniques (like vertical scaling or database partitioning) even less ineffective than they were earlier.

2 BACKGROUND: MODERN WAYS OF ORGANIZING A MULTI-TENANT ARCHITECTURE

There has already been some experience in the field of organizing cloud data storages (Chong et al., 2006b). At the moment, there are two main approaches to designing multi-tenant relational database.

- Usage of shared tables for all clients with attachment of 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 its pros and cons.

2.1 Shared Table Approach

This approach is the most radical in answering a question of sharing server resources. Its usage requires adding a special column to every table in a database which stores a tenant identifier to distinguish data of different clients. Every SQL query to the database of such 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 (Schiller et al., 2011), that allow to add such predicates automatically. 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 also has some drawbacks. First of all, it is the enlarging of the size of database tables and their indexes (Jacobs and Aulbach, 2007). This drawback results in a requirement of very high qualification of developer of database queries, since any mistake or inefficient solution can lead to significant degradation of an application performance. In second, usage of this approach implies a need to always add predicate of selection of data of a current tenant. This drawback leads to access errors, when users of one tenant can see data of another tenant in a case of a programmer error. The above mentioned (Schiller et al., 2011) concepts of extensions of the SQL language are possibly able to solve this issue. The third issue of this approach is a complexity of replication and backup copying of data of a separate tenant.

In general, this approach shows good results, when application's data schema does not contain many tables, and a typical query is relatively simple. If the above conditions are met, this approach allows the most effective usage of hardware resources.

2.2 Shared Process Approach

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

- Unification of the code of database queries and ease of writing new ones, because, in contrast to the shared table approach, queries known to operate only the current tenant's data, which usually have relatively small size, and, therefore, do not require a lot of memory and other database server resources for their execution (Jacobs and Aulbach, 2007).
- Relative ease of backup copying and data replication of data of a single tenant.
- Decrease of data security risks since tenant's data are grouped together in the own schema;
- Simplification of system administration.

But there are also some drawbacks of this approach. Its usage makes data dictionary of a database very large and heavyweight, decreasing overall database performance. Because of that a query planner is unable to use its cache effectively, that makes him generate a new plan of execution for almost every incoming query (Schiller et al., 2011). Compared to the shared table approach, a disk space is used less effectively (Schiller et al., 2011). Large amount of database objects results in a very long and hard procedure of a data structure change if it is required.

In general, this approach shows good results, if an application data structure is complex, and a typical query selects data from a large set of tables, makes nested subqueries and other complex data manipulations.

3 MOTIVATION: 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 (Candan et al., 2009). 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 a more significant point is a load balancing for the 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. To provide the required level of service, an application should be able to dynamically adapt 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 general problem assumes the study of the following directions of research:

- development of algorithms of load balancing for multi-tenant cloud database clusters;
- research of developed algorithms for efficiency and safety, including imitation modelling and stress testing;
- development of complex solution for organizing multi-tenant cloud database clusters using ordinary servers.

In the work we will focus our attention on the third goal. The first and the second ones are supposed to be studied in future.

4 SOLUTION REQUIREMENTS

The developed system would be intended for using by small and medium-sized software companies, and, therefore, should be designed to meet their needs and capabilities. The following points are important.

- Reliability and maximum guarantee of data safety. The reputation is extremely important for a cloud service provider, because his clients have to trust him their data.
- Efficient usage of available resources, providing maximum performance of an application.
- Similarity to traditional DBMS with minimal possible corrections for the cloud application specifics.
- Maximum horizontal scalability. The horizontal scalability is preferred to vertical, as it is cheaper and potentially allows infinitely increase the performance of the system.
- Ease and automation of administration as the manual administration of a very large and complex infrastructure could lead to

management chaos and system unmaintainability.

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.

First of all, it is the huge aggregate size of stored data. As the provider must 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. But the size of data of an average client is small. Since we are talking about multi-tenant solutions, this solution is likely to aim at small and medium-sized companies (so called «Long Tail» (Chong and Carraro, 2006a)), and therefore the number of users and the size of data of an average tenant are not large.

Another common point is the presence of shared data. Usually any cloud application has some set of data, which is shared among all tenants of the provider. The size of that data is usually relatively large, and modifications are very rare.

Since cloud solutions have centralized architecture and a service provider is responsible for the large amount of client's data, it needs good facilities for data backup and 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 data of one or some tenants are replicated. The above mentioned shared data are also replicated.

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

5 THE ARCHITECTURE OF MULTI-TENANT DATABASE CLUSTER MANAGEMENT SYSTEM

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

- Routing of 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 of the 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. 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 will not have in advance the knowledge about a data structure required for an application. 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 application data structure, and thus, its usage is difficult for the generic system. Moreover, the usage of a 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 1.

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

It is important to note that a query routing server has a small choice of executors for each query. If the query implies data modification, there are 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 could be routed to a slave server, but in a general case there would be just one or two slaves for a given master, so even in this case the choice is very limited.

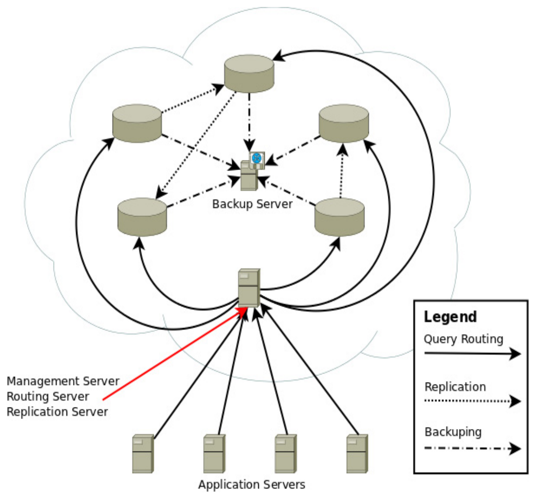

Figure 1: Multi-tenant database cluster architecture.

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 make a decision on 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 title, but it is important to note, that, unlike the traditional databases, in multi-tenant solutions replication is almost always partial, i.e. only a part of data is replicated. For example, data from the first tenant schema could be replicated to one database, from the second tenant schema — to another, and the third tenant schema itself could be a replica of a part of another 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 will be a set of agent-services placed at the same machines as database servers. These small programs-daemons should collect statistics about the load of the server and monitor server state in a case of failure. All the information collected would be sent to a central server for processing and analysing and would be used as an input data for the load balancing algorithm.

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

- initial distribution of tenants data among servers of a 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, 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 has the highest value, since the performance of an application depends on the success of its work. There are several key indicators that can be used to evaluate its effectiveness. First of all, it is the average response time of a service i.e. an average time between the arrival of a request and receiving a response to it. In second, it is an availability of a service, i.e. what percent of requests from the total number has been executed successfully, which failed to meet a time limit or other parameters, and which is not executed at all. Both previous criterions are affected by the average load of database servers. The cluster management system must provide conditions, when servers are relatively equally loaded, and there are no idling servers when others fail to serve all requests.

The core of load balancing system should become an algorithm of cluster load analysis and need for data redistribution. There are several considerations that this algorithm should take into account when making its decision about data redistribution. First of all, it is a performance of cluster servers. If the system is not homogenous, proportions of its parts should be taken into account. In second, it is 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 and the reliability of an application. However, if the number of tenants begins to grow, created redundant copies should be removed. Some data distribution strategies can also take into account the history of the individual tenant activity. If users of tenant A actively use an application, and users of tenant B don't, it makes sense to move the data of tenant B to a busier server and create fewer copies of them, since they unlikely will cause problems for the service. The algorithm should also prevent the creation of hot spots on writing the data in a context of organization of replication. The system should distribute master servers for all tenants in an appropriate way, taking into account the history of tenant activity.

6 CONCLUSIONS AND FUTURE WORK

Thus, in this paper we presented some principles of the architectural design of a multi-tenant database cluster. The implementation of the proposed architectural solutions should provide the framework the usage of which will simplify the development of applications according to the SaaS paradigm. Also, the proposed approach should facilitate the administration and maintenance of the cluster.

The above discussed algorithms for query routing and tenant data distribution 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 algorithms and choose the best in a general case according to the results of imitation modelling and stress testing. It is very likely that such versions will not be found and a final implementation of the system will contain several modifications of them, which showed themselves as the best ones under some external conditions. In this case a choice of an appropriate version of algorithms would become a task of a cluster management system administrator.

There are also some questions for a future work:

- the study of fault-tolerance of clusters in case of failure of one or more servers;
- the study of customization complexity issues on the side of application developers to satisfy specific needs of clients;
- the study of effective strategies of data replication.

REFERENCES

Chong, F., Carraro, G. (2006a). *Architecture Strategies for Catching the Long Tail*. Microsoft Corp. Website.

Chong, F., Carraro, G., Wolter, R. (2006b). *Multi-Tenant Data Architecture*. Microsoft Corp. Website.

Candan, K.S., Li, W., Phan, T., Zhou, M. (2009). *Frontiers in Information and Software as Services*. in Proc. ICDE, pages 1761-1768.

Jacobs, D., Aulbach, S. (2007). *Ruminations on Multi-Tenant Databases*. In Proc. of BTW Conf., pages 514–521.

Schiller, O., Schiller, B., Brodt, A., Mitschang, B. (2011). *Native Support of Multi-tenancy in RDBMS for Software as a Service*. Proceedings of the 14th International Conference on Extending Database Technology EDBT '11.