Design and Implementation of a Private Cloud's Database as a Service

Peng Lv [a], Xiuquan Qiao [b]

State Key Laboratory of Networking and Switching, Beijing University of Posts and Telecommunications, Beijing, 100876, China

[a]lvpp1992@163.com, [b]qiaoxq@bupt.edu.cn

Abstract. Cloud computing, which revolutionizes application development model, is an inevitable trend in the development of IT technology. Cloud computing providers sell Server Platform (IaaS), development environment (PaaS) or software application (SaaS) to users. As for PaaS, a separate database as a service (DaaS) is necessary, so that application developers can use as usual as normal database, rather than focus on the specific implementation of the database. In this paper, we present a general solution for a private cloud platform based on OpenStack, using MySQL database as DaaS, which can take advantage of the limited hardware resources. And we will explain how to design and implementation of DaaS, and analysis it's characteristics, such as multi-tenant, scalability, search strategy, high availability and so on.

Keywords: Cloud computing; OpenStack; MySQL; DaaS; Multi-tenant; Scalability.

1. Introduction

With the development of technology, cloud computing has become an inevitable trend. Cloud computing is an integration of a variety of technology development, such as hardware computing, network bandwidth, distributed computing, network storage technology, and virtualization technology. Nowadays, more and more manufacturers are interested in providing cloud computing service, for instance, Amazon Cloud (AWS), Google App Engine (GAE) [3], Alibaba cloud, and Sina APP Engine (SAE). Cloud computing changes traditional IT application development model, so that developers or teams just need to purchase an appropriate service from cloud computing providers, without spending extra effort to manage and maintain their own hardware or software resources. In this model, immediately development can be easy and efficient.

Traditional cloud computing has three types of service [1]:

1) Infrastructure as a Service(IaaS): To provide the basis of computing resources through the Internet, it is simply to provide a virtual machine level service;
2) Platform as a Service (PaaS): To provide development environment through the Interne. For the users, t the virtual machine is transparent, only a development environment they can use to develop;
3) Software as a Service (SaaS): To provide software and applications through the Internet, users can directly use these applications.

Our innovative cloud services platform, which is based on open source projects OpenStack, aims for college students and other research groups, and provide platform as a service (PaaS) and Software as a Service (SaaS). However, SaaS can't exist without database. Although OpenStack owns a database component, which is named Trove and continues to improve. Considering that the version of used OpenStack's Trove module is not perfect, and we can't follow OpenStack's fast updates because of its complex deployment feature [6]. Besides, Trove provide at least an alone database for each user, which can't make full use of hardware resources for our situation. Thus, we propose a new design for multi-tenant situation, and it is compatible with Trove.

2. Database solutions in multi-tenant model

As far as we know, cloud platform is multi-tenant-oriented. So, we need to consider multi-tenant factor when we design the DaaS. There are there common database solutions in multi-tenant model [2]. The first model is independent database [4, 5], in which each user has a separate database instance. It provides strongest independence, tenants' data are physically invisible, backup and recovery is very

flexible, and hardware consume the most. The second model is shared database and independent schema [4, 5], in which each user in the same database instance, but have separate schema. Tenants' data between each other is logically invisible, upper application use database as easy as normal database, but backup and recovery is slightly complex. The last model is shared database, shared schema, and shared data table [4, 5], in which tenants' data sharing in the data table level. The lowest cost, and highest resource utilization leads to various aspects of highly complex. The following figure can vividly display the different storage model of the three solutions (figure 1).

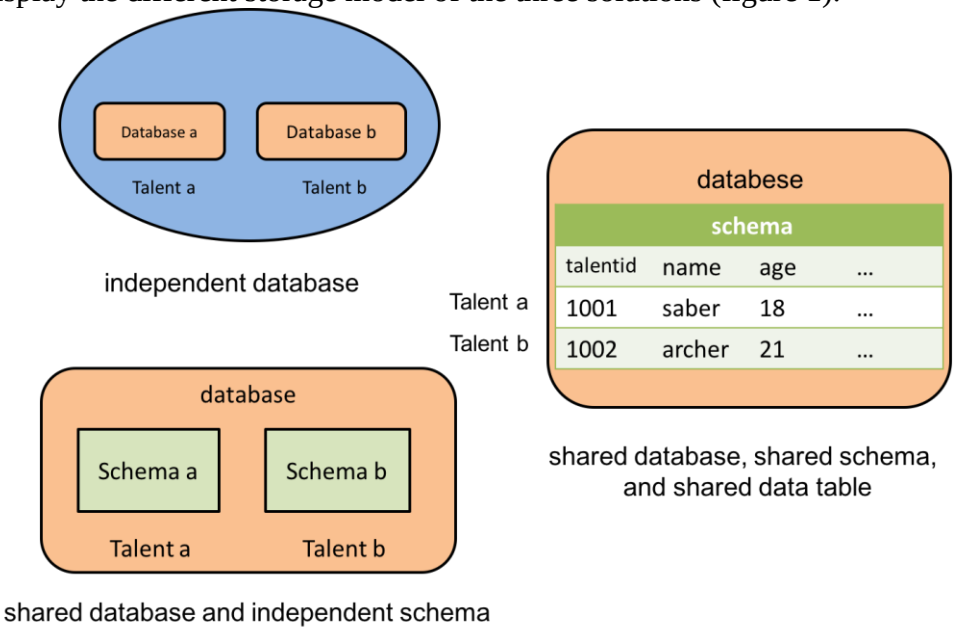

Fig.1. Database Solutions for Multi-Tenant Model

According to actual hardware utilization and technical difficulties, we decided to use the second model to design and implement our DaaS.

3. Design and Implementation

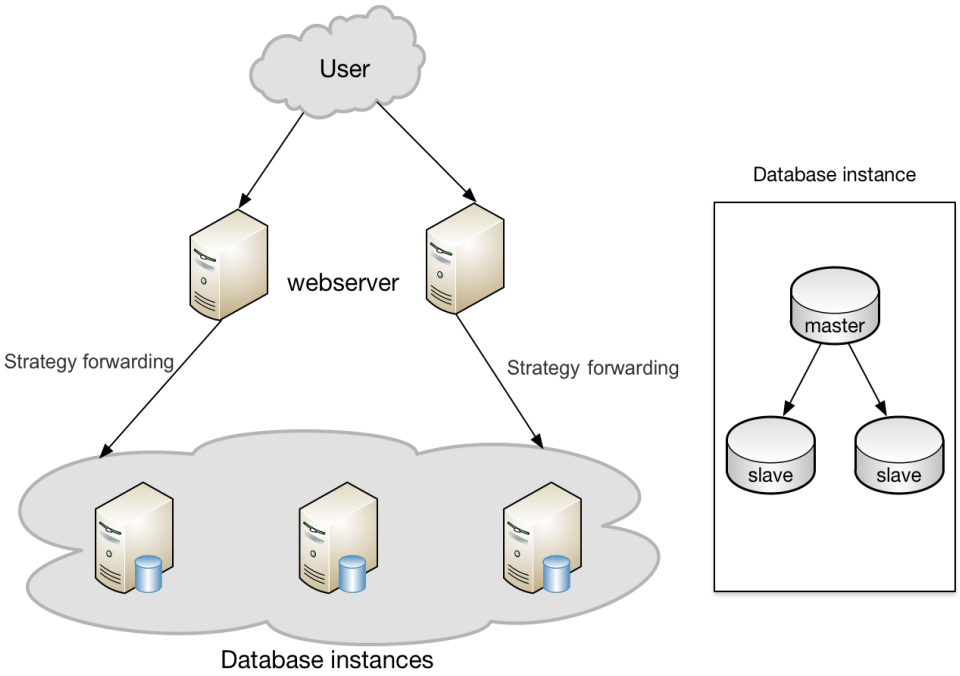

Fig.2. Architecture of the DaaS

MySQL database will be able to meet most needs of users. So, the service is based on MySQL. The figure 2 shows that a database cluster is treated as the basic instance, each of which consists of master and slave MySQL databases.

Scalability: The database system can accommodate natural horizontal expansion of the database. When an instance is added, we just need to update the total amount of current database instance, marked M here, without making any other changes.

Strategy Forwarding: When using the DaaS, a user just need to click the button to initialize his or her database from the website. Then, according to the select strategy, the background will select an instance to create an independent schema and return all the corresponding login information for the user. Finally, the user can normally use the database, without focusing on any other things. In the system, each user of the cloud platform will have a unique ID number N. Meanwhile, each instance has its own cluster number 1, 2, 3 ... M and a flag bit, which is used to mark whether the current instance accommodates the largest number of users. When the flag is true, the instance can't create a new schema, on the contrary, it's feasible. Therefore, the ID number of a selected instance can be calculated as:

$$\text{Number} = \text{N}\%(\text{M} - \text{k}) + \text{k} + 1 \quad (\text{k} = 1,2,3 \ldots \text{M} - 1) \tag{1}$$

The full instances' number increase from 1, 2, 3 to M, and the 'k' means that the first instance number whose flag is false.

Multi-tenant: The system is designed without database middleware. Because that the existing middleware rarely provides multi-tenant characteristic. What's more, those multi-tenant-oriented middleware is too redundant, or does not meet real needs. If we make changes on some lightweight middleware, such as MySQL-Proxy [8]. In this case, inputs and outputs may not perform as expected, and the performance may also decrease (for example, to maintain multiple database connection pool requires high cost) [8]. Ignoring middleware can let the database reach native database's reaction rate, but it will cause a little inconvenience at the same time. If you need to separate read and write or database connection pool, you make it possible from the client.

High Availability: As we know, a database cluster instance is treated as a unit instance. Therefore, when you design the system, you can simply add a standby master and a standby slave node (there are many same existing cluster implementations), to ensure the databases' high availability.

Backup: Backup's core function is based on MySQL database's built-in backup function [7]. Users click on backup button from the cloud platform website, then the system will back up their data. And different users' data are stored in different folders so that data is isolated. Of course, you can also store a user's data into the OpenStack's swift component.

Security: Since that the database can only be visited from cloud platform's internal access, external network is naturally isolated from database environment. And the database is deployed on a virtual machine created by OpenStack [6], which can make a good use of the security features of OpenStack. Within double protection, the database service will be able to be in a relatively safe environment, without additional security measures. But, if the database is deployed on a physical machine, there may need some special measures to make sure that the database is adequate security.

Resource Limits: As far as we know, cloud platform is adapted for multi-tenant. User resources requires not only isolated from each other, but also be appropriately limited to prevent a malicious user occupy all resources. Database resources limit is absolutely essential. As for the database connection limit and common properties [7], MySQL's build-in function is enough. However, there is no ready-made function for users' data capacity limit. Therefore, we propose a solution for capacity constraints:

1) We monitor each user's data capacity, we named Q here, for real-time, when a user's Q exceeds a preset value (e.g.1G), we will revoke the user's update, create and insert permissions of the database;
2) The user can still use the database, but only have query, delete permissions;
3) When the user deletes the data to a certain threshold (e.g.800M), we will regrant user's insert, update, and create permissions of the database. Finally, the user can normally use the database.

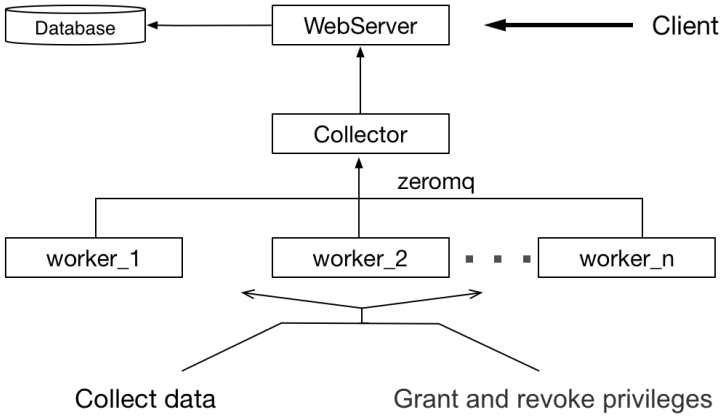

Fig.3. Architecture of the Limit and Monitoring Module

The worker will be running in the master node of each instance, collecting the user's capacity usage. When the capacity exceeds the limit, the worker will revoke users' appropriate permissions. But when the data is deleted, the worker will regrant the privileges. Each worker sends the data to the collector module through zeromq message queue. Then, the collector will store the data into the database, which is used by the website. In the end, the web page can display users' capacity usage dynamically, by getting the data from the database.

The limit and monitoring module not only limits the capacity of resources, but also monitors the users' real-time capacity usage.

Fig.4. Web Implementation

Fig.4 is the demo webpage. When you have clicked the initial button of the database for an application, you can see this page, which shows all the information of the database. In addition, you can directly operate the database after clicking the first button displayed in the Fig.4, because we have integrated the phpMyAdmin.

4. Conclusion

In this paper, a common cloud platform solution of database as a service (DaaS) was proposed, according to differences between general solutions in multi-tenant model. The design and implementation was detailed analyzed, through the scalability, security, backup, and other factors. The solution is especially fit for small-scale private cloud, because it can make a good use of hardware resources. If the cloud platform is established based one OpenStack, the proposed DaaS solution can coexist with OpenStack's Trove components. Besides, if OpenStack's version is old so that Trove's features are imperfect, or hardware limited situation, the solution will be an easy and better choice. However, the solution can't match the need of high performance or functionality, you'd better chose other solutions. Of course, the solution still needs further optimization and we will improve it in the next step.

Acknowledgements

In this paper, the research was sponsored by the Service Identification and Migration mechanism of Future Internet (No. 2012CB315802), National Key Basic Research Program of China (973 Program) and Web-based Dyanmic Service Provisioning Mechanism and Method Research in Information-Centric Networking (No. 61671081), Project of National Natural Science Foundation of China (NSFC).

References

[1]. Intel open source technology center. Design and Implementation of OpenStack [M]. Electronic Industry Press, 2015. (In Chinese)

[2]. Jacobs D, Aulbach S. Ruminations on Multi-Tenant Databases[C]//BTW. 2007, 103: 514-521.

[3]. Lawton G. Developing software online with platform-as-a-service technology [J]. Computer, 2008, 41(6): 13-15.

[4]. Qi S, Lin H. Suitable data migration approach for SaaS multi-tenant model [J]. Jisuanji Gongcheng yu Yingyong (Computer Engineering and Applications), 2011, 47(32): 65-70.

[5]. Multi-Tenant Data Architecture. https://msdn.microsoft.com/en-us/library/aa479086.aspx?cm_mc_uid=50806818414214333444546&cm_mc_sid_50200000=1473045504

[6]. OpenStack official website. http://www.OpenStack.org/

[7]. MySQL5.6 Reference Manual. http://dev.MySQL.com/doc/refman/5.6/en/

[8]. MySQL proxy guide. http://downloads.MySQL.com/docs/MySQL-proxy-en.pdf