Elasticity in Multitenant Databases Through Virtual Tenants

[1]Monika Jain, [2]Iti Sharma

Career Point University, Kota, Rajasthan, India

[1]jainmonica1989@gmail.com, [2]itisharma.uce@gmail.com

***Abstract* - Multi-tenancy has shown promising results in achieving high operational cost efficiency by sharing hardware and software resources among multiple customer organizations, called tenants. In the context of cloud computing, this paradigm enables cloud providers to reduce operational costs by dividing resources and to simplify application management and maintenance. These benefits come with associated challenges of isolation, dynamic scaling and elasticity. This paper explores these issues in the context of multi-tenant Database-as-a-Service. In addition, we propose a solution “virtual tenants” to solve the problem of elasticity and isolation in multitenant database applications.**

Index Terms – multitenancy, Cloud computing, database as a service, elasticity.

I. Introduction

Efficient data processing is a fundamental and vital issue for almost every scientific, academic, or business organization. Although it is possible to purchase the necessary hardware, deploy database products, establish network connectivity, and hire the professional people who run the system, as a traditional solution, this solution has been getting increasingly expensive and impractical as the database systems and problems become larger and more complicated. In order to simplify the burden on the users of their cloud offerings, many PaaS vendors have started offering database services on the cloud. All physical database administration tasks, such as backup, recovery, managing the logs, etc., are managed by the cloud provider. The responsibility for logical administration of the database, including table tuning and query optimization, rests on the developer. An organization that provides database service has an opportunity to do these tasks and offer a value proposition provided it is efficient. Such database service providers are called tenants of the cloud provider. Users wishing to access data will now access it using the hardware and software at the tenant (service provider) instead of their own organization’s computing infrastructure, yet the application would not be impacted by outages due to software, hardware and networking changes or failures at the tenant’s site since it is hosted on a cloud. A detailed discussion of database-as-a-service, architecture models has been presented in [1]. Another advantage of the cloud computing is in the elasticity it can provide to applications. In a huge data center, new resources can be allocated and assigned to clients in a short time. The startup company could flexibly allocate new resources when the number of visitors augments and later release these resources when they are not needed anymore. This way it could prevent the deterioration in the quality of service and save money on not paying for non-used infrastructure.

An effective way of utilizing a cloud’s resources is Multi-tenancy. By sharing one application across many tenants, multi-tenancy attempts to replace many small application instances with one or few large instances thus bringing down the overall cost of IT infrastructure. But multitenancy comes with an attaché of effort required to write multitenant applications, designing techniques to handle multiple tenants, and issues like privacy and security. Data isolation and performance isolation top the list of desirable features in a multitenant database service. Another essential feature is elasticity.

Agrawal et al [2] use the concepts of data fission and data fusion, enabling lightweight elasticity using low cost live database migration, and designing intelligent and autonomic controllers for system management without human intervention. But the idea is quite complicated and needs much programming effort.

In this paper we present a model based on virtual tenants, which can be adopted at any data isolation level and multitenancy degree. The aim is to have efficient elastic service along with performance isolation as suggested in [3].

II. Multi-tenancy in clouds

One of the essential attributes of Cloud Computing is nowadays is multi-tenancy. The concept was pioneered by Software as a Service vendors like Salesforce.com and spread across other key segments of the cloud marketplace, taking the form of both Platform as a Service (PaaS) and Infrastructure as a Service (IaaS) systems. Multi–tenancy refers to a principle of sharing a single instance of an application among multiple customers, also known as tenants. Multi-tenancy is a common key attribute for both public and private clouds and though it applies to all three layers of a cloud: IaaS, PaaS and SaaS, it is more often used in the context of SaaS applications. Multi-tenancy enables cloud service subscribers to leverage more cost-effective shared, or “pooled,” resources built on a single code-base. Tenants may be given the ability to customize some parts of the application, such as color of the user interface or business rules, but they cannot customize the application's code.

This also implies that although tenants are using the same building blocks in their configuration, the appearance or workflow of the application may be different for two tenants. Also, the Service Level Agreement (SLA) of each tenant can differ. A tenant is any application -- either inside or outside the enterprise -- that needs its own secure and exclusive virtual computing environment. This environment can encompass all or some select layers of enterprise architecture, from storage to user interface. All interactive applications (or tenants) have to be multi-user in nature. The advantages of a multi-tenancy SaaS over a third-party-hosted, single-tenancy application are discussed in [4].

Aspects related to Multitenancy are:

- Security: sharing database naturally raises security concerns. Data isolation is the term used for prevention of leakage of data of one tenant to others who also have their data in the same database. There are 3 key components that define the degree of isolation between multiple tenants in a data center: access policies, application deployment and data access and protection.
- Economy: Multi-tenancy can be economical because software development and maintenance costs are shared between the tenants. It can be contrasted with single-tenancy, an architecture in which each customer has its own software instance and may be given access to code. There is always a trade-off between the economy level a subscriber wishes to achieve and security desired.
- Service assurance and faster updates: With a multi-tenancy architecture, the provider has to make updates only once. With single-tenancy architecture, the provider has to touch multiple instances of the software in order to make updates. This reduces downtime.
- Efficiency and flexibility: A SaaS provider can run one instance of its application on one instance of a database and provide web access to multiple customers. Each tenant's data is isolated and remains invisible to other tenants.

A. *Degrees of Multitenancy*

The degree of multi-tenancy is based on how much of a core application or SaaS, layer is designed to be shares across tenants. The highest degree of multi-tenancy allows the database schema to be shared and supports customization of the business logic, workflow and user-interface layers. In other words, all the sub-layers of SaaS offer multi-tenancy in this degree. The lowest degree corresponds to Multitenancy only at infrastructure and platform level.

- Lowest degree: IaaS and PaaS are multi-tenant, while SaaS is single tenant.
- Middle degree: IaaS and PaaS are multi-tenant. Small SaaS clusters are multi-tenant.
- Highest degree: IaaS and PaaS are multi-tenant. SaaS is fully multi-tenant also.

In fact, the higher the degree of multi-tenancy (meaning the more a cloud provider's infrastructure and resources are shared), the lower the costs for customers.

B. *Multitenancy in Databases*

The principles underlying a multi-tenant data service are:

- Simplicity: Application developers should be able to write applications for a multi-tenant database as for a single database tenant, without bothering about how database is allocated and shared among tenants. Other tenant needs would be completely abstracted from the application.
- Efficiency: Efficiency is maintained by the fact that all data tenants can share the same physical resources (memory, CPU) with other tenants.
- Dynamic scaling: Each database tenant can be distributed and scale dynamically across multiple machines, if necessary, to meet demand. This includes moving data to other machines if necessary. Scaling must be supported seamlessly without any downtime.
- Support for multiple isolation/sharing levels: With a multi-tenant database, the level of sharing/isolation is set on demand, based on the specific requirements. At that level, a user can choose a dedicated tenant, in which case the multi-tenant database would allocate tenants on a set of machines that is dedicated to that specific user. Clearly, the benefit of this is isolation, at the cost of efficiency. Another type of sharing is available to the user at the same time: Choosing to share with any individual or group under the user's control. In other words, rather than dictating either extreme (full isolation or full sharing) for all users, with a multi-tenant database we can dynamically choose the right level that best fits our purpose.

III. Multitenancy in DBaaS : Approaches

The approaches towards building a multitenant database can be broadly categorized as fine-grained and coarse grained depending on the degree of multitenancy involved. The architecture also depends on the data isolation desired. In this section, we'll look at the continuum between isolated data and shared data, and identify three distinct approaches for creating data architectures that fall at different places along the continuum..

A. *Data Isolation*

Fig 1 shows the continuum between isolated data and shared data.

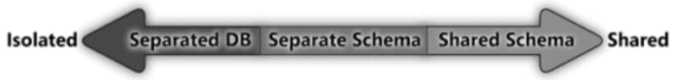

Figure 1 Continuum of Data Isolation

- Separate databases (shared nothing): Storing tenant data in separate databases is the simplest approach to data isolation. Computing resources and application code are generally shared between all the tenants on a server, but each tenant has its own set of data that remains logically isolated from data that belongs to all other tenants.
- Shared databases, separate schemas: Another approach involves housing multiple tenants in the same database, with each tenant having its own set of tables that are grouped into a schema created specifically for the tenant.

- Shared Database, Shared Schema (shared everything) approach involves using the same database and the same set of tables to host multiple tenants' data. A given table can include records from multiple tenants stored in any order; a Tenant ID column associates every record with the appropriate tenant.

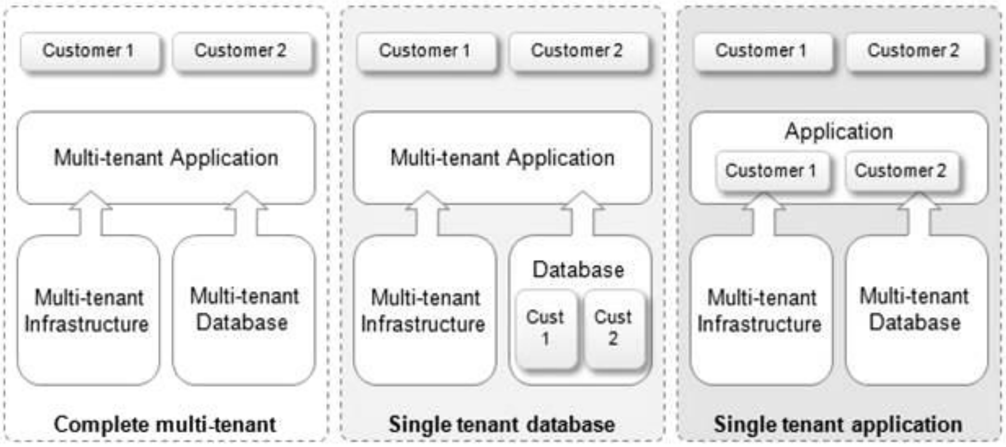

Figure 2 Models for Multitenancy at Database

B. Models for Multitenant Databases

Based on approach used for data isolation and other parameters related to tenants, we can have one of the following three basic models [5], shown in Fig 2:

1) Complete Multitenant: This has highest degree of Multitenancy by employing shared-everything approach. The inherent risks/complexities of this model include requirement of a very complex architecture, high business risk as the data is shared between all tenants, providing customized data backup/restore services is not easy and equal distribution of load cannot be ensured.
2) Single Tenant Database: This has a moderate degree of Multitenancy where application layer is commonly shared among all the tenants but not the databases. The level of separation is determined by the tenant architecture.
   a. Different schemas within the same database – where each schema can contain the tables for the respective tenant.
   b. Different Databases within the same machine – Each tenant gets a different database. This gives the flexibility to provide specific services like customized back-up and restore services.
   c. Different Databases residing on different machines – Independent machines are allocated for each tenant. This provides the highest level of security from a data isolation/storage perspective.
3) Single Tenant Application – This is the inverse of the previous model. Here the database layer is kept common across all the tenants but the application layer is isolated. This model is employed where certain operations like customization of interface (in the form of change in business logic, rules, display, etc) is to be performed over the application layer.

Similar models of multi tenancy relevant in the context of the different cloud paradigms as described in [6] are:

- Virtual Machine Based
- Shared Cluster
- Shared Installation
- Shared Database

C. Issues in Multitenancy at Database Level

A common challenge in almost every SaaS application is multi-tenancy at the data layer. To put it simply, the challenge is how to share data resources (database in most cases) among multiple users, while at the same time ensuring data and performance isolation between those users, as if they are running on completely physically separated servers. Efficient multi tenancy, elastic scalability and database privacy are the three important issues which have to be addressed by DBaaS provider [7].

The challenge with the fine-grained model is that it is fairly complex to maintain and implement. If you have an existing application, it requires a complete rewrite and also forces fairly significant changes in your existing data model. The challenge with the cross-grained model is that cost margins per user are still rather high. The other challenge that neither approaches addresses today is dynamic elasticity. What happens when a customer grows beyond their allocated shared capacity? Both approaches seem to rely on certain capacity assumptions, and can scale by splitting multiple users between their shared resources. However, none of the approaches seem to handle a situation where a particular user's capacity grows beyond the allocated resources.

Ideally a multi-tenant cloud storage system serves requests of multiple customers (tenants) in such a way that (1) computing and storage resources are shared among such customers and (2) this sharing of resources does not weaken system security. In practice, multi-tenancy is a trade-off between security and costs: the wider the subset of resources shared (e.g., same physical machine vs. same OS), the more the cloud system can amortize costs and increase utilization. However, this sharing leads to weaker isolation and consequently higher security risks.

Another major concern arising from multi-tenancy is the lack of performance isolation. When workloads from multiple tenants contend for shared resources, one tenant's performance may be affected by the workload of another tenant.

Databases can be scaled up (by moving to a larger server that uses more powerful processors, more memory, and quicker disk drives) and scaled out (by partitioning a database onto multiple servers). Different strategies are appropriate when scaling a shared database versus scaling dedicated databases, like tenant-based horizontal partitioning or single-tenant scale-out based on the model used.

D. Elasticity

Variation in load (user queries) is a major problem since such load cannot be estimated in advance. A possible solution to the problem is to ensure elasticity. An elastic system dynamically reacts to actual load by adding new or removing some virtual resources. When the load on a component increases over a given limit, new instance of the component is added to accommodate the growth. If on the other hand the load decreases, unnecessary instance can be removed. In cloud computing, elasticity is defined as the degree to which a system is able to adapt to workload changes by provisioning and deprovisioning resources in an autonomic manner, such that at each point in time the available resources match the current demand as closely as possible [8]. Elasticity aims at matching the amount of resources allocated to a service with the amount of resources it actually requires, avoiding over- or under-provisioning. Over-provisioning, i.e., allocating more resources than required, should be avoided as the service provider often has to pay for the resources that are allocated to the service. Under-provisioning, i.e., allocating fewer resources than required, must be avoided, otherwise the service cannot serve its users with a good service.
One potential problem is that elasticity takes time. A cloud VM can be acquired at any time by the user, however it may take up to several minutes for the acquired VM to be ready to use.

IV. PROPOSED SOLUTION: VIRTUAL TENANTS

To achieve Multitenancy with scaling and isolation at infrastructure level, Clouds have been designed for this purpose from beginning. At platform level, Multitenancy is dealt through virtualization. In fact, once multi-tenancy was dealt with properly at the operating system level, writing a multi-tenant application became significantly simpler and therefore widely adopted. Thus came software applications with support for multiple users sharing single instance across the cloud. For databases the challenge is open yet. Existing models cannot adapt well towards elasticity and performance isolation. In this section we propose a model to solve this problem.

A. Approach

At the time of set-up we create a number of virtual tenants with their separate pool of resources (memory, virtual machine). All the virtual tenants share the database. Each tenant is associated with a set of virtual tenants as per its requirements. A tenant may be associated with one or more virtual tenants, but a virtual tenant is associated to at most one tenant. Hence, some virtual tenants may not be associated with any tenant and no virtual tenant is shared among the tenants. As shown in Fig 3, we create a layer between actual database and its application. None of the queries from users hit the database directly. Instead, a mapping layer which uses a mapping table (associating a tenant with virtual tenants) generates a unique customerID, derived as a function of UserId(within a tenant), TenantID and VirtualTenantID. This serves as a primary key into the shared database. The UserID not only manages the identity of the user, but also the privileges granted to a particular user. The query is packed into a query which uses the primary key customerID to select only that part of database which can actually be accessed by the particular user. The translated query then hits the database.

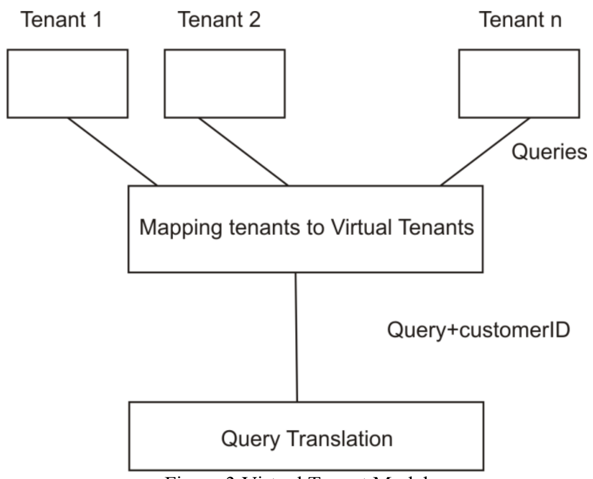

Figure 3 Virtual Tenant Model

B. Features

The features of the proposed model are:

1) Data isolation and privacy – the model can be used with all levels of data isolation. Even with shared schema data privacy is assured by making the user identity (customerID) as primary key and translating all queries such that query accesses only appropriate portions of database.
2) Performance isolation - For performance isolation, each virtual tenant is allotted its set of resources. When demand from a particular tenant grows, we increase the number of virtual tenants associated with it, thus increasing the resources. Among all allotted virtual tenants, the physical resources are shared on a simple round robin basis. Due to varying number of virtual tenants dedicated to a particular group of users or an enterprise, it in effect becomes a weighted round robin approach, serving more where demand is high, yet the level of performance at every virtual tenant is same. This in practice gives every end-user equal performance (or the performance as per their SLA).
3) Elasticity – The association of tenants and the virtual tenants is stored in a mapping table, which can be changed dynamically. If demand from a tenant grows, it can be associated with an extra virtual tenant. When demand shrinks the virtual tenants associated to the tenant decrease. Thus, elasticity can be managed.
4) Simple Management - The main advantage of this model is the simplicity of application management at operating system and infrastructure level. Maximum number of virtual tenants to be created is bound by the virtualization limits of underlying infrastructure and middleware. Since all the virtual tenants have been created with their respective resource pool, its management can be done prior to service launch. At the time of subscription, a tenant is allotted a set of virtual tenants as per its SLA and average load. Also, the set of virtual tenant required at peak load is also decided as per SLA. The virtual tenants actually mapped onto the tenant are always maintained as closest to current load requirements. Whenever demand rises due to query bursts, virtual tenants from allotted set are mapped to the tenant accordingly. These are removed from mapping table as soon as user request traffic drops down. Virtual tenants in the allotted set of a tenant follow the tenant's database schema; hence this model can be used with all approaches ranging from shared-nothing to shared-everything. While in shared-nothing approach, the virtual tenant access the database instance of the tenant to which they are mapped; in shared-everything approach all database instances accessed by virtual tenants are same. A virtual tenant may have its own virtual machine at infrastructure level, thus ensuring the isolation and security of a certain level if this is so demanded in SLA for a particular tenant.

V. Conclusion

Many SaaS applications are in fact Database-as-a-Service and rely on shared databases for fulfilling the promise of cloud computing – high operational cost efficiency. Multitenancy is an essential feature of such applications. The existing approaches for Multitenancy at data layer do not ensure data privacy due to improper data isolation. Also performance isolation is not guaranteed. Another major issue is of elasticity. We propose a model using virtual tenants to ensure data isolation, performance isolation and elasticity with very simple management technique.

Practical implementation and security issues remain an open problem. Revenue calculations can be done along the lines of elastic reservations discussed in [3].

References

[1] P. Malliga. "*Database Services for Cloud Computing – An Overview.*" International Journal of Computers & Technology, Vol 2 No. 3, June, 2012.
[2] D Agrawal, A Abbadi, S das and A J Elmore. "*Database Scalability, Elasticity, and Autonomy in the Cloud*". Proceedings of the 16th International Conference on Database Systems for Advanced Applications, Vol I pp 2-15.2013.
[3] S Das, V R Narasayya. F Li and M Syamala. "*CPU Sharing Techniques for Performance Isolation in Multitenant Relational Database-as-a-Service*". Proceedings of the VLDB Endowment, Sep 2013.
[4] K Jani, B Kumar and H Shah. "*Degree of Multi-tenancy and its Database for Cloud Computing.*" International Journal of Engineering Development and Research, Vol 3, 2013, pg 168-171.
[5] http://erpcloudnews.com/2010/06/multi-tenant-versus-single-tenant-erp-a-comparison
[6] Oracle White Paper on Enterprise Architecture, 2011 Database as a Service: Reference Architecture – An Overview.
[7] D Agrawal, El Abbadi, S Antony S.Das. "*Data Management Challenges in Cloud computing Infrastructures*", In DNIS, pp 1-10, 2010.
[8] N R Herbst, S Kounev and R Reussner. "*Elasticity in Cloud Computing: What It Is, and What It Is Not.*" 10th International Conference on Autonomic Computing (ICAC '13).