Degree of Multi-tenancy and its Database for Cloud Computing

[1]Kaushal Jani, [2]Bimal Kumar, [3]Harshal Shah

[1,2]Department of Computer Engineering, Saraswati Institute of Engineering and Management, Kadi, Gujarat, India
[3]Department of Computer Science Engineering, Government Engineering College, Modasa, Gujarat, India
[1]kaushaljani2007@gmail.com, [2]vmlkumar80@gmail.com, [3]erharshal@gmail.com

***Abstract*—Multi-tenancy, which allows a single application to emulate multiple application instances, has been proposed as a solution to this problem. 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. In this paper, we present importance of Multi-tenancy in cloud computing, degree of Multi-tenancy and Multi-tenancy in databases. It will helpful to understand about Multi-tenancy and its benefits in software as a service in cloud computing environment.**

***Keywords*— Cloud computing, Multi-tenancy, Database in Multi-tenancy**

I. INTRODUCTION

In cloud computing, the meaning of multi-tenancy architecture has broadened because of new service models that take advantage of virtualization and remote access. A software-as-a-service (SaaS) provider [2], for example, can run one instance of its application on one instance of a database and provide web access to multiple customers. In such a scenario, each tenant's data is isolated and remains invisible to other tenants. Multi-tenancy is an architecture in which a single instance of a software application serves multiple customers. Each customer is called a tenant. 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. Multi-tenancy can be economical because software development and maintenance costs are shared. It can be contrasted with single-tenancy, an architecture in which each customer has their own software instance and may be given access to code[1]. With a multi-tenancy architecture, the provider only has to make updates once. With single-tenancy architecture, the provider has to touch multiple instances of the software in order to make updates.

II. WHY MULTI-TENANCY MATTERS IN THE CLOUD

There's a debate in the software industry over whether multi-tenancy is a prerequisite for cloud computing. Those considering using cloud apps might question if they should care about this debate. But they should care, and here's why: multi-tenancy is the most direct path to spending less and getting more from a cloud application.

There's a debate in the software industry over whether multi-tenancy is a prerequisite for cloud computing. Those considering using cloud apps might question if they should care about this debate. But they should care, and here's why: multi-tenancy is the most direct path to spending less and getting more from a cloud application. I sit firmly in the multi-tenancy camp. A multitenant architecture is when customers share an app in the cloud, while a single-tenant cloud app is similar, if not identical, to the old hosted model. But compare two subscription-based cloud apps side by side--with the only difference being that one is multitenant and the other is single-tenant--and the multitenant option will lower a customer's costs and offer significantly more value over time. 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.

It's a matter of simple revenue and cost economics of cloud services. Most cloud app providers' revenues come from selling monthly or annual per-seat subscriptions, which bring in just a fraction of the annual revenue that an on-premise software license with comparable functionality would provide a vendor. The challenge for selling software subscriptions, then, is to reduce operating costs in order to manage with less; otherwise the provider may end up doing much more than an on-premise vendor does, such as maintain multiple versions, run multiple infrastructures, maintain customer-specific code, and perform upgrades, but with fewer dollars. Multi-tenancy provides the answer, because it spreads the cost of the infrastructure and labor across the customer base; in fact, customers sharing resources right down to the database schema is ideal for scaling.

The economies of scale get even better as the provider adds customers, and customers benefit from this scaling up. As the cloud app provider's costs decrease, it has more room to innovate and grow, thus delivering more value. Even if customers' costs don't drop, over time they should expect to see more value, such as increased functionality.[2]

So, what's the debate about? Those who say multi-tenancy isn't necessary to making the cloud model work are typically companies that have long made money from on-premise software and don't want to cannibalize their existing revenues. They might offer a subscription for their single-tenant application, but this could simply be the software license, maintenance, and hosting fees divided into monthly payments which almost certainly would be much higher than a comparable multi-tenant application.

What's even more interesting is the "unsure" camp in this debate. These are typically the traditional on-premise vendors that decided to give the cloud a try. They often try to save money by using all or some of their existing on-premise infrastructure and practice for their cloud apps, by avoiding the investment in a new technology infrastructure that supports multi-tenancy. However,

the high cost of replicating and maintaining instances for each single tenant (or customer) eventually catches up with them. They are forced to try approaches where they can share some of the infrastructure, but their fundamental affinity to the old on-premise model usually proves to be a stubborn barrier to changing their software, infrastructure and culture to fully support a shared model. And if they keep on this single-tenant path as they scale up customers, their margins get lower as each new customer sucks up more resources.

For a customer, there can be trade-offs in sharing an application. Think of it like living in a condo versus a house; one management company serves all the tenants, and you may not be any more special than any other tenant. Everyone gets upgraded with a new version at the same time, for example. But for many types of apps, the cost/value formula of multi-tenancy is the best answer.

III. WHAT IS MULTI-TENANCY

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.

A departmental application that processes sensitive financial data within the private cloud of an enterprise is as much a "tenant" as a global marketing application that publishes product catalogs on a public cloud. They both have the same tenancy requirements, regardless of the fact that one has internal co-tenants and the other has external.[2]

Multi-tenancy is the key common attribute of both public and private clouds, and it applies to all three layers of a cloud: Infrastructure-as-a-Service (IaaS), Platform-as-a-Service (PaaS) and Software-as-a-Service (SaaS).

Most people point to the IaaS layer alone when they talk about clouds. Even so, architecturally, both public and private IaaS go beyond tactical features such as virtualization, and head towards implementing the concept of IT-as-a-Service (ITaaS) through billing -- or chargeback in the case of private clouds -- based on metered usage. An IaaS also features improved accountability using service-level-agreements (SLAs), identity management for secured access, fault tolerance, disaster recovery, dynamic procurement and other key properties.

By incorporating these shared services at the infrastructure layer, all clouds automatically become multi-tenant, to a degree. But multi-tenancy in clouds has to go beyond the IaaS layer, to include the PaaS layer (application servers, Java Virtual Machines, etc.) and ultimately to the SaaS or application layer (database, business logic, work flow and user interface). Only then can tenants can enjoy the full spectrum of common services from a cloud -- starting at the hardware layer and going all the way up to the user-interface layer, depending on the degree of multi-tenancy offered by the cloud.

IV. DEGREE OF MULTI-TENANCY

The exact degree of multi-tenancy, as it's commonly defined, is based on how much of the core application, or SaaS, layer is designed to be shared 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.[3]

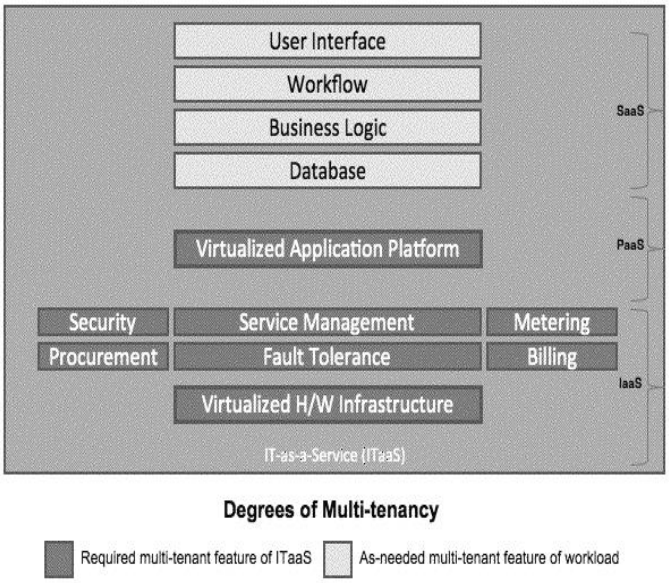

Fig. 1 Degree of Multi-tenancy

In the lowest degree, multi-tenancy is limited to the IaaS and PaaS layers, with dedicated SaaS layers for each tenant and in the middle degree of multi-tenancy are clusters of homogenous tenants that share database schemas (or schemae) and other application layers. In the middle level, each cluster of users has its own version of database schema and the application itself.

We can sum up the discussion on the degree of multi-tenancy as follows:

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

V. Multi-Tenancy and Its Benefits in a SaaS Cloud Computing Environment

In cloud computing, multi-tenancy means that a SaaS (Software as a Service) vendor provides a single version of its software for all its customers. This differs from a single-tenant hosted solution, where the application is housed on a vendor's server but the codebase is unique for each customer.

How does multi-tenancy work? Although all users of the software access the same foundational components, the data and configurations that are specific to a customer are stored in a separate and secure container. Users can access all the capabilities of the software, but their data aren't shared.[2]

The advantages of a multi-tenancy SaaS over a third-party-hosted, single-tenancy application include the following:

A. **Lower costs through economies of scale**
With a single-tenancy-hosted solution, SaaS vendors must build out their data center to accommodate new customers. In contrast, in a multi-tenant environment, new users get access to the same basic software, so scaling has far fewer infrastructure implications for vendors.

B. **Shared infrastructure leads to lower costs:**
SaaS allows companies of all sizes to share infrastructure and data center operational costs. Users don't need to add applications and more hardware to their data centers, and some small- to medium-sized businesses don't even need data centers if they utilize SaaS.

C. **Ongoing maintenance and updates**
End users don't need to pay costly maintenance fees in order to keep their software up to date. New features and updates are included with a SaaS subscription and are rolled out by the vendor.

D. **Configuration can be done while leaving the underlying codebase unchanged**
Although on-premises applications and single-tenant-hosted solutions are often customized, this endeavor is costly and requires changes to an application's code. Additionally, this customization can make upgrades time-consuming, because an upgrade might not be compatible with your customization. Most multi-tenant SaaS solutions are designed to be highly configurable so that businesses can make the application perform the way they want without changing the underlying code or data structure. Because the code is unchanged, upgrades can be performed easily.

E. **Vendors have a vested interest in making sure everything runs smoothly**
Multi-tenant SaaS providers have, in a sense, all their eggs in one basket. Although this sounds dangerous, it's a benefit to end users. In a single-tenant environment, if there is a service disruption, it may affect only one customer, meaning that the vendor might be slow to respond and fail to take the necessary steps to ensure the problem doesn't recur.

In contrast, with a multi-tenant solution, a slight glitch could affect all of a vendor's customers. It is, therefore, imperative that SaaS vendors invest significant amounts of money and effort into ensuring uptime, continuity, and performance.

VI. Database of Multi-Tenancy

A. Multi-tenancy Models

Multi-tenancy in databases has prevalent for hosting multiple tenants within a single DBMS while enabling effective resource sharing [2, 12, 9]. Sharing of resources at different levels of abstraction and distinct isolation levels results in various multi-tenancy models. The three models explored in the past [12] consist of: shared machine (also referred to as shared hardware), shared process, and shared table. SaaS providers like Salesforce.com [11] are a common use cases for database multi-tenancy, and traditionally rely on the shared table model. The shared process model has been recently proposed in a number of database systems for the cloud, such as RelationalCloud [7], SQLAzure [4], ElasTraS [8]. Nevertheless, some features of cloud computing increases the relevance of the other models. Soror et al. [10] propose using the shared machine model to improve resource utilization. To improve understanding of multi-tenancy, we use the classification recently proposed by Reinwald [9] which uses a finer sub-division (see Table 1). Though some of these models can collapse to the more traditional models of multi-tenancy. However, the different isolation levels between tenants provided by these models make this classification interesting and helpful for selecting a target classification when building a multitenant database.

#	Sharing Mode	Isolation	IaaS	PaaS	SaaS
1	Shared hardware	VM	√	√	
2	Shared VM	OS User		√	
3	Shared OS	DB Instance		√	
4	Shared instance	Database		√	
5	Shared database	Schema		√	
6	Shared table	Row		√	√

Table 1: Multi-tenant database models, how tenants are isolated, and the corresponding cloud computing paradigms.

Shared Table **-** The shared table model uses a design which allows for extensible data models to be defined by a tenant with the actual data stored in single shared table. The design often utilizes 'pivot tables' to provide rich database functionality such as indexing and joins [2]. While this model offers advantages of maintaining a single database instance, isolating tenants for migration becomes difficult due to shared locking mechanisms. The reliance on consolidated pivot and heap tables could lead to poor performance due to all tenants sharing index structures. Additionally, the shared table model requires that all tenants reside on the same database engine and release (version). This limits specialized database functionality, such spatial or object based, and requires

that all tenants use limited subset of functionality. This model is ideal when tenant data requirements follow similar structures or patterns, such as in the case of Force.com offering customizations on a customer relationship database [11].

Shared Hardware **-** The models corresponding to rows 1–3 share resources at the level of the same machine with different levels of abstractions, i.e., sharing resources at the machine level using multiple VMs (VM Isolation) or sharing the VM by using different user accounts or different database installations (OS and DB Instance isolation). There is no database resource sharing. Rows 1–3 only share the machine resources and thus correspond to the shared machine model in the traditional classification. While these models offer strong isolation between tenants, these models come with a cost of increased overhead due to redundant components and a lack of coordination using limited machine resources in an unoptimized way. The lack of coordination is prominent in the case of using a virtual machine for each tenant, row 1, where each tenant behaves as if it has exclusive disk access [8].

Shared Process **-** Rows 4–5 involve sharing the database process at various isolation levels—from sharing only the installation binary (database isolation), to sharing the database resources such as the logging infrastructure, the buffer pool, etc. (schema isolation), to sharing the same schema and tables (table row level isolation). How a database instance can be isolated between tenants varies between implementation. For example, with MySQL each tenant can be given their own schema with limited user permissions. Rows 4–5 thus span the traditional classes of shared process (for rows 4 and 5).

With different forms of multi-tenancy, components that constitute a tenant vary. We henceforth use the term cell to represent all information necessary to serve a tenant. A multitenant database instance consists of thousands of cells, and the actual physical interpretation of a cell depends on the multi-tenancy model.

CONCLUSION

The objective of this paper was to present a comprehensive
study about the Multi-tenant mechanisms available today. we also explain database in Multi-tenant. Initially, we proposed a classification for Multi-tenant, based on the main features found in the analyzed commercial and academic solutions. In a second moment, diverse related works were reviewed in order to define the state of the art of Multi-tenant in clouds.

REFERENCES

[1] M. Armbrust, A. Fox, Griffith et al., “Above the clouds: A berkeley view of cloud computing,” EECS Department, University of California, Berkeley, Tech. Rep. UCB/EECS- 2009-28, 2009.
[2] S. Aulbach, T. Grust, D. Jacobs, A. Kemper, and J. Rittinger. Multi-tenant databases for software as a service: schema-mapping techniques. In SIGMOD, pages 1195–1206, 2008.
[3] F. Chong, G. Carraro, and R. Wolter, “Multi-Tenant Data Architecture,” MSDN Library, Microsoft Corporation, 2006.
[4] P. A. Bernstein, I. Cseri, N. Dani, N. Ellis, A. Kalhan, G. Kakivaya, D. B. Lomet, R. Manner, L. Novik, and T. Talius. Adapting Microsoft SQL Server for Cloud Computing. In ICDE, pages 1255–1263, 2011.
[5] D. Jacobs and S. Aulbach, “Ruminations on multi-tenant databases,” BTW Proceedings, 2007.
[6] N. Chohan, C. Bunch, S. Pang, C. Krintz, N. Mostafa, S. Soman, and R. Wolski. AppScale: Scalable and Open AppEngine Application Development and Deployment. In CloudComp, 2009.
[7] C. Curino, E. Jones, R. Popa, N. Malviya, E. Wu, S. Madden, H. Balakrishnan, and N. Zeldovich. Relational Cloud: A Database Service for the Cloud. In CIDR, pages 235–240, 2011.
[8] S. Das, S. Agarwal, D. Agrawal, and A. El Abbadi. ElasTraS: An Elastic, Scalable, and Self Managing Transactional Database for the Cloud. Technical Report 2010-04, CS, UCSB, 2010.
[9] B. Reinwald. Database support for multi-tenant applications. In IEEE Workshop on Information and Software as Services, 2010.
[10] A. A. Soror, U. F. Minhas, A. Aboulnaga, K. Salem, P. Kokosielis, and S. Kamath. Automatic virtual machine configuration for database workloads. In SIGMOD, pages 953–966, 2008.
[11] C. D. Weissman and S. Bobrowski. The design of the force.com multitenant internet application development platform. In SIGMOD, pages 889–896, 2009.
[12] D. Jacobs and S. Aulbach. Ruminations on multi-tenant databases. In BTW, pages 514–521, 2007.