“Cloud Computing is a computing capability that provides an abstraction between the computing resource and its underlying technical architecture, enabling convenient, on-demand network access to a shared pool of configurable computing resources that can be rapidly provisioned and released with minimal management effort or service provider interaction.”

-Hong Cai, IEEE Senior Member; Ning Wang, Ming Jun Zhou
A non-technical Analogy to the concept of Multi-tenancy

- Effective use of land
- Privacy
- Infrastructure / Maintenance Cost Sharing
- General Maintenance
- Freedom

- Customers have to install & update their own software
- Customers manage their own data
- Every Customer needs his own server for deployment
- Customizations are easy
- High initial costs
Multi-tenancy

- Multi-tenancy is a critical technology to allow one instance of application to serve multiple customers by sharing resources.
  - multi- multiple, independent customers are served
  - tenant is any legal entity responsible for data and is provided on a contractual basis. Tenant is the contract signee

- Applications: IaaS, PaaS, SaaS
Cloud Computing Service Models
What is the most important asset of any business? **Data**

Trust between customer & vendor is a key factor.

Tenant database is the cornerstone of tenant management.

Data architecture for a multi-tenant system should be:
- Robust
- Secure
- Efficient
- Cost-effective

Three approaches to managing multi-tenant data (Database layer):
- Separate Database Process, Shared Machine
- Shared Database Process, Separate Tables
- Shared Table
Each tenant gets their own database process and multiple tenants share the same machine.

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.

- Advantages
- Disadvantages
Each tenant gets their own tables and multiple tenants share the same database process.

It 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.

- Advantages
- Disadvantages

**Shared Database Process, Separate Tables**
It 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.

- Advantages
- Disadvantages

**Shared Table**
Applications optimized for a shared approach tend to require a larger development effort than applications designed using a more isolated approach (because of the relative complexity of developing a shared architecture), resulting in higher initial costs. Because they can support more tenants per server, however, their ongoing operational costs tend to be lower.
### Storage Requirements for Schema instances

<table>
<thead>
<tr>
<th></th>
<th>Memory 1 instance</th>
<th>Memory 10,000 instances</th>
<th>Disk 1 instance</th>
<th>Disk 10,000 Instances</th>
</tr>
</thead>
<tbody>
<tr>
<td>PostgreSql</td>
<td>55</td>
<td>79</td>
<td>4</td>
<td>4,488</td>
</tr>
<tr>
<td>MaxDB</td>
<td>80</td>
<td>80</td>
<td>3</td>
<td>1,168</td>
</tr>
<tr>
<td>Commercial 1</td>
<td>171</td>
<td>616</td>
<td>200</td>
<td>414,210</td>
</tr>
<tr>
<td>Commercial 2</td>
<td>74</td>
<td>2061</td>
<td>3</td>
<td>693</td>
</tr>
<tr>
<td>Commercial 3</td>
<td>273</td>
<td>359</td>
<td>1</td>
<td>13,630</td>
</tr>
</tbody>
</table>
From Multi-tenant database to a Multi-tenant SaaS
What was the SaaS Vision

• What is a Service?
  ○ Intangible & insubstantial commodity, no ownership
  ○ A service is a set of one time consumable and perishable benefits delivered from the accountable service provider

• How does Software classify as a Service? OR What is the difference between Software as a Product and Software as a Service?
  ○ On-demand software

• What are the Advantages of SaaS, in terms of economic impacts?
  ○ On demand licensing
  ○ Low cost user provisioning (setting up a user)
Hosted Services & Multi-Tenancy

- **What is a Hosted Service Model?**
  - In the hosted service a service provider develops an application and operates the system that hosts it.

- **What are its advantages?**
  - 1. reduce the total cost of ownership of an application by aggregating customers together and leveraging economy of scale.
  - 2. This principle applies to both capital expenditures, e.g., for hardware and software, and operational expenditures.

- **How does multi-tenancy help here?**
  - Multi-tenancy is an optimization for hosted services in which multiple customers are consolidated onto the same operational system.

- **What are the characteristics of such a multi-tenant system?**
  - Multi-tenancy allows pooling of resources, which improves utilization by eliminating the need to provision each customer for their maximum load.
  - Multi-tenancy can also improve management efficiencies by providing a uniform framework for administering the system.

- Multi-tenancy at different layers of hosted service.
1. Single Application Instance

2. Multiple Application Instances, Shared Address Space.

3. Multiple Application Instances, Separate Address Space

4. Multiple Virtual Images

5. Multiple Instances on Separate Hardware

Five Major Approaches for enabling multi-tenancy [2]
Multi-tenancy & Virtualization
Multi-tenant Virtualization platform

- Define Virtualization
- Hardware Virtualization v/s Software Virtualization
- Multi-tenant virtualization is cost-effective.
- Multi-tenant virtualization provides a high degree of concurrent virtual environments by sharing hardware / software platform securely to provide isolation between instances.
- Requires applications to be modified or developed from scratch to target a specific multi-tenant virtualization platform. (inflexible, ex. Salesforce.com – CRM)
Towards building a complete Multi-tenant cloud

1. Next generation multi-tenant virtualization cloud computing platform [3]


1. Next generation multi-tenant virtualization cloud computing platform[3]

- Multi-tenant virtualization platform – Uranus
- Increased scalability & security maintaining flexibility
Multi-tenant virtualization cloud computing platform

Functional Layers

- Reverse Proxy Layer
- Virtualization Layer
- DNS Layer
Uranus

Reverse Proxy Layer
The users’ HTTP requests first encounter the system here.

This layer consists of one or more reverse proxy servers that route the HTTP call to the LAMP server in the Virtualization layer.

One or many Apache web servers running on standard HTTP port (80) but using a reverse proxy configuration.

Apache caching modules – cache & mem_cache to enhance performance & scalability. This reduces the no. of HTTP requests to the back-end Virtualization layer by caching hot objects.

Apache Configuration for Reverse Proxy

```xml
<VirtualHost *:80>
  ServerName sitel.com
  ServerAlias www.sitel.com

  ProxyPass / http://sitel.com:9101/
  ProxyPassReverse / http://sitel.com:9101/

  <IfModule mod_mem_cache.c>
    CacheEnable mem /
    MCacheSize 4096
    MCacheMaxObjectCount 100
    MCacheMinObjectSize 1
    MCacheMaxObjectSize 2048
  </IfModule>

</VirtualHost>
```
Uranus

Virtualization Layer
Components

- Consists of several LAMP components, each consisting of a separate Apache & Mysql instance. These servers implement the LAMP stack to which individual LAMP applications would be deployed.
- It’s a multi-tenant Virtualization layer.
- Each tenant exists as a separate system user with a dedicated home directory which is shared by the tenants own Apache & Mysql processes.
- Process & Data isolation.
- Apache Virtualization.
- Mysql Virtualization.
Configurations

Apache config for tenants

```
<VirtualHost *:9101>
    ServerAdmin webmaster@sitel.com
    ServerName sitel.com
    ServerAlias www.sitel.com
    DocumentRoot /tcmvm/tenant1/var/www-vhost/sitel.com
    <Directory />
        Options FollowSymLinks
        AllowOverride All
    </Directory>
    <Directory /tcmvm/tenant1/var/www-vhost/sitel.com>
        Options Indexes FollowSymLinks MultiViews
        AllowOverride All
        Order allow,deny
        allow from all
    </Directory>
    ErrorLog "/tcmvm/tenant1/var/log/apache2/sitel.log"
    CustomLog "/tcmvm/tenant1/var/log/apache2/sitel.err"
    ServerSignature On
</VirtualHost>
```

Mysql config setup

```
[mysqld_multi]
mysqld = /usr/bin/mysqld_safe
mysqldadmin = /usr/bin/mysqldadmin
user = multi_admin
password = aim3MeR6

[mysqld1]
datadir = /tcmvm/tenant1/var/lib/mysql
socket = /tcmvm/tenant1/var/lib/mysql/mysqld.sock
pid-file = /tcmvm/tenant1/var/run/mysqld/mysqld.pid
user = mysqldmulti_admin
port = 7101
server-id=1
log-bin=mysql-bin
log-error=/tcmvm/tenant1/var/log/mysqld.log
```
The base OS directory structure was replicated as shown in the snippet on the right.

Each Tenant is capable of running multiple LAMP and static websites within their allocated directory.

The snippet depicts a typical tenants sand-boxed file system.

This file is shared by the

**Tenant’s sand-boxed filesystem**
Proxy Server: Should be a powerful one with moderate amount of memory/CPU in order to handle the many thousands of concurrent connections and caching of hot objects.

DNS server has been excluded under the assumption that this feature is already present in the network.

Deployment Diagram
Threshold for Single Apache with no other load
Regardless of whether we are running a single apache/mysql or 40 idle apache/mysql there is hardly a difference.

The performance hit by using their virtualization method does not impact the system performance.

Threshold for Single Apache with 40 idle tenants
Threshold for single Apache with 40 loaded tenants
In this and the previous the system is put under constant load.

They get a similar threshold value whether they are benchmarking a single apache instance being constantly loaded or where there are 40 instances where the load is equally spread.

**Threshold for Single Apache while being constantly loaded**

- Foundations of SaaS multi-tenancy
- Separation of concern of different roles
- Integration of SaaS multi-tenancy with cloud virtualization
The Conceptual Model of SaaS
Multi-tenancy
Web Application is deployed on a Web Application Server.

Database Server is usually installed on a separate machine devoted to database processing.

A remote LDAP server may provide directory access service.

A remote Message Queue server may provide messaging service.

**Topology of a production stage Web application deployment**
Foundations of SaaS Multi-tenancy

Isolation and Customization
An *isolation point* means a specific Web application artifact (class, method, field) etc. that has its tenant specific behavior or value.

The *isolation points* could be identified at the application migration phase & stored in a metadata repository.

SaaS applications can provide customizations & configurations only on the basis of the *isolation points*. 

### Concept of Isolation & Isolation points

<table>
<thead>
<tr>
<th>Type of isolation points</th>
<th>Where it is used</th>
</tr>
</thead>
<tbody>
<tr>
<td>App Scope</td>
<td></td>
</tr>
<tr>
<td>Objects</td>
<td></td>
</tr>
<tr>
<td>EJB</td>
<td></td>
</tr>
<tr>
<td>Authentication &amp;</td>
<td></td>
</tr>
<tr>
<td>Authorization</td>
<td></td>
</tr>
<tr>
<td>Remote Service Call</td>
<td></td>
</tr>
<tr>
<td>File</td>
<td></td>
</tr>
<tr>
<td>Database</td>
<td></td>
</tr>
<tr>
<td>LDAP</td>
<td></td>
</tr>
<tr>
<td>Message</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Type 1: Application level isolation</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Global variables (including static field, global objects, cache, etc.)</td>
<td></td>
</tr>
<tr>
<td>Literals (constant)</td>
<td>SessionBean</td>
</tr>
<tr>
<td>Application cache</td>
<td>EntityBean</td>
</tr>
<tr>
<td>Servlet Context</td>
<td></td>
</tr>
<tr>
<td>Servlet Context Application configuration files</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Type 2: Resource level isolation</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Pure JDBC</td>
<td></td>
</tr>
<tr>
<td>Common Data Access Framework (e.g. JPA)</td>
<td></td>
</tr>
<tr>
<td>EntityBean</td>
<td></td>
</tr>
<tr>
<td>Basic LDAP</td>
<td></td>
</tr>
<tr>
<td>JMS</td>
<td></td>
</tr>
<tr>
<td>MDB</td>
<td></td>
</tr>
</tbody>
</table>
Tooling for the Application Development Team

- SaaS multi-tenancy tooling as an Eclipse based plug-in
- Adds a new isolation point view to the traditional Eclipse IDE
- Helps identify all the candidate isolation points for the application developer.
- Isolation points could be
  - Application level
  - Resource level
<table>
<thead>
<tr>
<th>Description</th>
<th>Resource</th>
<th>Path</th>
<th>Location</th>
<th>Type</th>
<th>ExtendInfo</th>
</tr>
</thead>
<tbody>
<tr>
<td>Field</td>
<td>number</td>
<td>Counter.java /javapetstore/...</td>
<td>Line 10</td>
<td>Field</td>
<td></td>
</tr>
<tr>
<td></td>
<td>tsList</td>
<td>Counter.java /javapetstore/...</td>
<td>Line 11</td>
<td>Field</td>
<td></td>
</tr>
<tr>
<td>Literal</td>
<td></td>
<td>C:/mtConfigDir/data/index/...</td>
<td>Line 7</td>
<td>Literal</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>C:/cleanPetStore/workspace/...</td>
<td>Line 9</td>
<td>Literal</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>Java Pet Store /javapetstore/...</td>
<td>Line 11</td>
<td>Literal</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>/javapetstore/images/banner_logic/...</td>
<td>Line 13</td>
<td>Literal</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>/javapetstore/images/splash.gif/...</td>
<td>Line 15</td>
<td>Literal</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>com.sun.javae.blueprints.petstore/...</td>
<td>Line 17</td>
<td>Literal</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>com.sun.javae.blueprints.petstore/...</td>
<td>Line 19</td>
<td>Literal</td>
<td></td>
</tr>
<tr>
<td>JPA DataSource</td>
<td></td>
<td>java:comp/env/jdbc/PetstoreDB/...</td>
<td>Line 12</td>
<td>jta-data-source</td>
<td></td>
</tr>
</tbody>
</table>
Customization

- Handled during tenant on-boarding phase. Ex: visitor counter on the home page of a Web application.
- When a new tenant subscribes to this application the tenant administrator could customize this counter’s label through setting a tenant specific string.
Separation of Concerns of Different Roles

1. Application Developer
2. Cloud/SaaS Operator
3. Tenant Administrator
4. Tenant User
1. Application Developer

- Web Application Developers are responsible for developing UI, Business logic & Database of the application.
- Developer team is responsible for identifying the isolation points and export those isolation points to the metadata repository.
- This is the linkage between SaaS multi-tenant application & the Cloud Platform.
- These Application Developers are not aware of MT concepts. MT non-awareness developers. [1]
2. Cloud Operator

- Responsible for deploying applications to the Cloud platform
- Registering the application as an offering so that customers could subscribe to it
- Defining the charging policy (matching different payment with different SLA) and policy for allocating remote resource pools (database)
- Responsible for approving the request from a Tenant Administrator
3. Tenant Administrator

- Responsible for subscribing to the SaaS multi-tenant application by agreeing to the terms & conditions of the service.
- Responsible for making tenant specific simple & high level Configurations & Customizations.
- Upload tenant specific data to the SaaS platform
- Create account for the end users of the tenant.

- At the end of TA’s operations a URL to the entry point of the application will be provided
4. Tenant User

- End User of a multi-tenant web application
Integration of SaaS multi-tenancy with cloud virtualization
Create Virtual Images for Multi-tenant Application Server & Database Server

- Setting up multi-tenancy runtime from scratch is time consuming
- For App Server: installation of original app server + multi-tenancy add-on pack + configuration of app server to connect to db
- For db Server: installation of original db server, initialization of multi-tenancy system db
- To streamline the process, create virtual images for app & db servers using VMware, KVM, Xen
Topology model is a bridge between an abstract Application model & Cloud infrastructure services based on Virtual Machines.

Topology model is an aggregation of definition of all nodes that make up the cluster built with Virtual Images.

Application model to Topology model
Example of Topology model for multi-tenancy enabled cloud

"topology_vms": [
{
"id": "vm_App",
"image": {
"location": "http://imagestore/.../images/AppImage/",
"type": "m1.small",
"parts": [
{"part": "http://imagestore/.../AppNode.zip"},
{"part": "http://imagestore/.../App2DBLink.zip"},
{"part": "http://imagestore/.../AppMulti-tenancy.zip"}
]
},
{
"id": "vm_DB",
"image": {
"location": "http://imagestore/.../images/DBImage/",
"type": "m1.small",
"parts": [
{"part": "http://imagestore/.../DBNode.zip"},
{"part": "http://imagestore/.../DBMulti-tenancy.zip"}
]
}]}
Instantiate the Virtual Images on the cloud platform

Parse the topology

Invoke the IaaS layer API’s and initialize the Virtual Images and start the base image

Assign dynamic IP to the Virtual Image

Start the Virtual Image into a Virtual Machine based on the “type” attribute specified in the topology file

Additional activation codes need to be executed as system services of the base OS

Instantiate the Virtual Images on the cloud platform
Offering contains an SLA and a matching price

Application, SLA, price makes up a SaaS billing policy

Different types of SLA’s

- Economic (shared table)
- Intermediate (separate schema)
- Advanced (separate database)
- Deluxe (separate database instance)

(Subscriber) Making the New Multi-tenant Application into an Offering
Tenant Administrator

Subscribing to a Multi-tenant Offering

Customizing the Multi-tenant Application for the Tenant
SaaS Operator

Utilization of CPU for the tenants

CPU usage

Time (minutes)

Tenant 5
Tenant 4
Tenant 3
Tenant 2
Tenant 1
Conclusion

- This paper describes the core SaaS multi-tenancy models consisting of tenant interceptor, tenant context, tenant map, tenant propagation, remote resources.
- This paper also introduces the end to end process of making an existing Web application to be multi-tenancy enabled, and separating concerns of different roles involved.
- This paper puts the SaaS multi-tenancy operation in the complete life cycle of Cloud platform services.
## Main requirements

| Multi-tenancy                                                                 | Deal with monitoring information belonging to all tenants of one data center  
|                                                                              | Isolation based on data privacy laws                                        
|                                                                              | Some info however needs to be propagated to all tenants – a problem with a physical server that affects all the Virtual Machines |
| Scalability                                                                  | Scale to large numbers of monitoring agents, event notifications, tenants, resources |
| Dynamism                                                                     | Inherent in multi-tenant data centers                                        
|                                                                              | Quick & frequent addition/removal of tenants to/from data center             
|                                                                              | Assignment of resources to tenants                                           |
| Simplicity                                                                   | Interface to the monitoring system should be easy to understand, use, code against |
|                                                                              | System must be easy to install and maintain for DC operator and tenant.       |
| Comprehensiveness                                                            | One single monitoring system should be usable for all kinds of monitoring information |
Monitoring System Functional Architecture
4. Two Tier Multi-Tenancy Scaling and Load Balancing[5]

A two-tier SaaS scaling and scheduling architecture at both service and application level

A Resource allocation algorithm that selects suitable server nodes to run application/service duplicates

Two duplication time strategies, lazy and pro-active are provided to be chosen according to application Requirements
Duplication Strategies

Application Request (R)

Component Throughput (thr)

Overloaded Component

Under loaded Component
Duplication Strategies

- Duplication of Application
  - Assuming that the application is built using SOA, the application instance with all of its service instances is duplicated.
  - It’s the coarsest granularity duplication. Used by GAE.

- Duplication of Services
  - New service instances will be created and deployed to servers if all existing instances of this service are overloaded.

- Duplication at a Mixture Application/Service
  - Duplication can happen at both application and service levels.
  - Each load balancer then might have fewer instances to manage, thus the balancing workload is further distributed on the application level.
Two Tier SaaS Scaling & Scheduling Architecture
Two Tier SaaS Scaling and Scheduling Architecture

Application/Service Container:
Re-deployable Service Package:
Service Replica/Instance:
Monitoring Service:
Service Load Balancer:
Tenant Configuration Files:
Algorithm 1: Resource Allocation (RA)
Input: l level cloud clusters, given resource \( r_x \).
Output: An optimal allocation \( T_{opt} \).
Signature: \( T \) allocate(int \( r_x \))

for(int \( i = 0; i \leq l; i++ \)) // initialization
  \( t_{2i} = 0; \)
  int \( j = l + 1; \)
// \( j \) is the last level adding one more server node of which
// will cause \( r_{total} > r_x \)
for(int \( i = l; i \geq 0; i-- \)) {
  while (\( |C_{2i}| != 0 \&\& r_{total} + r_{2i} < r_x \)) {
    \( t_{2i}++; \)
    \( |C_{2i}|--; \)
    if (\( |C_{2i}| == 0 \)) \( j = i; \)
  }
if(\( r_{total} == r_x \)) return \( T; \)
for(int \( i = 0; i < j; i++ \)) {
  \( t_{2j}--; \)
  \( |C_{2j}|++; \)
if (\( j == l + 1 \)) return null;
else {
  \( t_{2j}++; \)
  \( |C_{2j}|--; \)
  int \( r_{min} = r_{total}; \)
  release(\( T; \)); // release server nodes in current allocation
  return allocate(\( r_{min}; \));
}
Security Issues in a Native Multi-tenant System[1]

- **Security Isolation**
  - Authentication Isolation
  - Access Control Isolation
  - Information Protection Isolation

- **Performance Isolation:**
  - prevent the (potentially bad) behaviors of one tenant from adversely affecting the usage performance of other tenants in an unpredictable manner.
  - Secondly, avoid the unfairness among tenants in terms of usage performance
  - Resource Allocation for better System Performance:
    - By Tenant Resource Reservation
    - By Tenant Resource Admission Control
    - Tenant Oriented Resource Partition

- **Information Protection Isolation:**
  - protect the integrity and confidentiality of each tenant’s critical information, prevent the critical information of one tenant from being read or modified by other unauthorized tenants and users via hacking attempts
Fault (Availability Isolation) [1]

- In traditional single tenant system, the availability is usually measured by following formula:
\[
ST\text{-}Availability = \frac{MTTF}{MTTF + MTTR}
\]

- The availability of the multitenant system can be defined as follows:
\[
MT\text{-}Availability = 1 - \frac{MTTR}{MTTF + MTTR} \times \frac{X}{N}
\]

- **Fault Detection & Diagnosis:**
  - quickly identify the currently infected tenant
  - each tenant should have the ability to monitor the states of its own running instance, and report to the service platform in a timely manner via the mechanisms like heart-beating and periodical simulations.

- **Fault Propagation Prevention:**

- **On-Line Repair:**
Conclusion

- Multi-tenant databases
- Migration of SaaS multi-tenant application to a cloud
- Multi-tenant cloud monitoring system
- Multi-tenancy 2 tier scaling and load balancing
- Security Issues in a Native Multi-tenant System
1. A Framework for Native Multi-Tenancy Application Development and Management, Chang Jie Guo1, Wei Sun1, Ying Huang2, Zhi Hu Wang1, Bo Gao1 IBM China Research Laboratory, Beijing, 2IBM T.J. Watson Research Center, New York
http://ieeexplore.ieee.org/gate.lib.buffalo.edu/stamp/stamp.jsp?tp=&arnumber=4285271


3. Next generation multi-tenant virtualization cloud computing platform,
http://ieeexplore.ieee.org/gate.lib.buffalo.edu/stamp/stamp.jsp?tp=&arnumber=5745841

5. Two-Tier Multi-Tenancy Scaling and Load Balancing, Wei-Tek Tsai*, Xin Sun, Qihong Shao, Guanqiu Qi http://ieeexplore.ieee.org/gate.lib.buffalo.edu/stamp/stamp.jsp?tp=&arnumber=5704303

7. Ruminations on Multi-Tenant Databases, Dean Jacobs, Stefan Aulbach Technische Universität München Institut für Informatik - Lehrstuhl III (I3) Boltzmannstr. 3 D-85748 Garching bei München,
Questions
The End

! THANK YOU !