Asian Journal of Mathematics and Computer Research
8(3): 180-206, 2016
ISSN: 2395-4205 (P), ISSN: 2395-4213 (O)

International Knowledge Press
www.ikpress.org

PROPOSED ARCHITECTURE OF MULTI-TENANT'S APPLICATIONS FOR GEOGRAPHICAL DATA CENTERS IN PRIVATE CLOUD SECTOR

MANJEET GUPTA[1*], DEVESH KUMAR SRIVASTAVA[2] AND DURG SINGH CHAUHAN[3]

[1]Uttrakhand Technical University, Dehradun, India.
[2]Manipal University, Jaipur, India.
[3]G.L.A University, Mathura, India.

AUTHORS' CONTRIBUTIONS

This work was carried out in collaboration among all authors. Author MG gathered the initial data and developed - Mutitenancy Model, its proposed architecture, its implementation and wrote VM balance algorithm, implemented the procedure to make Virtual Machine at Red hat Linux platform and developed the private cloud under the supervision of author DKS. Author DKS designed the work and trouble shoot the programing issues to develop the private cloud. Author DSC interpreted the results and conclusion. All authors read and approved the final manuscript.

Received: 4th August 2015
Accepted: 6th September 2015
Published: 12th October 2015

Original Research Article

ABSTRACT

The Multi-tenants software applications work for different groups from a single instance and help to retain development, maintenance, and administration costs. Existing efforts at a structured documentation of the elementary concepts are either technology-specific or confined to certain details. In cloud-based architectures, multi-tenancy means that purchaser, organizations, and consumers are distributing infrastructure and databases throughout the geographical area so as to gain price and performance benefits. These services offer a pay-as-you-go lease style investment with little or no upfront costs versus buying all of the hardware and software outright. Other advantages include the capacity to scale easily and tier more services and functionality on an as required basis. The benefits, are so enthralling that cloud computing is forecasted by someone to be the substitution for traditional means of acquiring these services and business capabilities up to now. In this paper we proposed architecture of Multi-tenant's Applications for geographical data centers which improved performance of Multi-tenant's Applications in private cloud sectors.

Keywords: Cloud computing; virtualization; multi-tenant; cloud sim; software as a service.

1 Introduction

Software as a service provider is the main crux of this chapter. It describes the basic architectures and best practices for implementing SaaS application. Software as a service pronounced as saas is software that is

**Corresponding author: kangmanjeet@gmail.com;*

installed and implemented across the internet and it is deployed to run beyond a firewall in the local area network or on a personal computer [1]. With SaaS, a contributor licenses an application to customers as a service on request, through contribution or a pay-as-you-go model. Occasionally SaaS is also explained as "software on demand." SaaS was firstly widely established for Customer Relationship Management (CRM) and Sales Force Automation. Now, it has become standard place for many business tasks, inclusive of computer generated bill, human resource management, invoicing and service desk management [2]. The prime cloud facility providers comprise the Microsoft, Sales force, Amazon, skytap and Google.

Cloud Computing is just a new term given to the utility computing; which is around in the industry for a long. The evolution of cloud computing is depicted in Fig. 1. The cloud is categorized on the basis of its location and the deployment model.

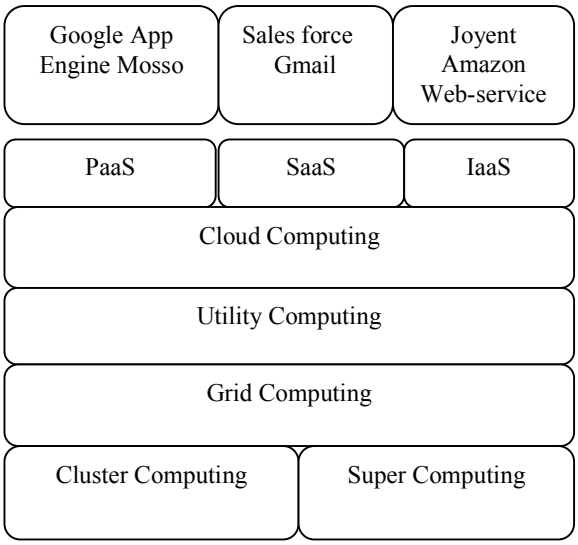


Fig. 1. The evolution of cloud computing [2]

Multi-tenant application sellers produce a single type of its software for its entire consumer. This differs from a single-tenant hosted solution; where the application is kept on a vendor's server but the code base is distinctive for each purchaser [3]. Many tenants are sharing its resources over multi-cloud. Service suppliers are skillful to make network infrastructures and information architectures that are scalable computationally, systematic and clearly incremented to perform the numerous customers that share them. Multi-tenancy spreads on the layers at that services are provided [4]. In IaaS, the tenants share the infrastructure resources like hardware, servers, and information storage devices. Multi-tenancy spreads on the layers at that services are provided. In IaaS, the tenants share the infrastructure resources like hardware, servers and knowledge storage devices. With SaaS tenants are derivation constant variety of application (e.g., Salesforce.com), which suggests that knowledge of multiple tenants is kept within the identical information and may even use the same Tables [6]. For the security, risks with multi-tenancy must be recognized at all layers. The following few sections explore how this can be used for shared hardware and application infrastructure. Multi-tenant and multi-tenancy are not new; both the term have been very much used to narrate the application architectures which is mainly designed for supporting the multiple users or "tenants" up to many years.

CEO's of the leading SaaS companies' accord that it is not feasible to grow a SaaS trading without multi-tenancy.

The benefits of a multi-tenancy SaaS for a third-party-hosted, single-tenancy requisition comprises the following:

1. Costing less through the economies of scale**:** By using a single-tenancy-hosted solution, SaaS dealers must constructed their data center to entertain new customers. In a multi-tenant environment, every new user get the access to the identical basic software, so the scaling has far fewer infrastructure implications for sellers which mainly depends upon the size of the application and the quantity of infrastructure required which is clearly shown in figure
2 Shared infrastructure leads to lower costs:
3. Ongoing maintenance and updates
4. Arrangement can be done while quitting the underlying codebase uninterrupted.
5. Sellers have a vested fascination in making sure that all runs evenly.

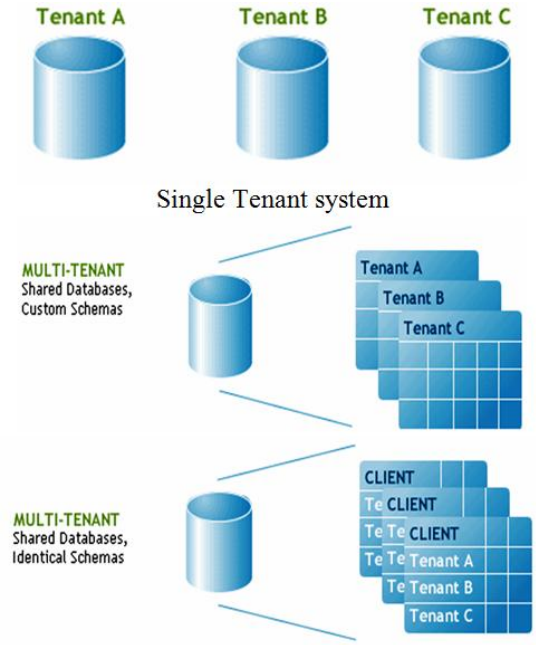


Fig. 2. Multitenant scheme with custom schemas [3,4]

2 Related Works

Cloud computing is an emerging computing paradigm. It aims to share data, calculations, and services transparently among users of a massive grid. Although the industry has started selling cloud-computing products, research challenges in various areas, such as UI design, task decomposition, task distribution, and task coordination, are still unclear [1]. SaaS multi-tenancy models consisting of tenant interceptor, tenant context, tenant map, tenant propagation, remote resources (such as database server, LDAP server, message queue server) in a distributed multi-tier Web system [2]. A tenant is considered the owner or the supplier of a SaaS application. He is responsible of the management, maintenance and update of the application and should ensure its availability and security. The other aspect that provides the advantages and power to multi-tenant architectures is the database model [5]. An appropriate model for the database can take an important increase in the scalability of the system. For achieving that it is possible to find different configurations, each one with its domain of applications, powers and limitations- Separate databases, Shared database, separate schema, Shared database, shared schema. To choose the best option in each case it is necessary to study the case in which the configurations are going to be used and try to make the most of them in terms of efficiency and cost. In this approach a shared database, shared schema is chosen. This allows using only the stored

space that is necessary in each case and avoiding the problems explained before with lower costs [6]. Here, metadata table stores important information about every custom field defined by every tenant, including the field´s name (label) and data type .Firstly, the particular ID data of the associated record in the primary data table is vital for the approach. This approach allows each tenant to create as many custom fields as necessary to meet its business needs [13]. The technical design of Force.com, the world's first PaaS, which delivers turnkey multitenancy for Internet-scale applications [8]. The research details force.com's patented metadata-driven architecture components to provide an understanding of the features used to deliver reliable, secure, and scalable multitenant applications. Only one set of hardware resources is necessary to meet the needs of all users, a relatively small, experienced administrative staff can efficiently manage only one stack of software and hardware, and developers can build and support a single code base on just one platform (operating system, database, etc.) rather than many. The economics afforded by multitenancy allow the application provider to, in turn; offer the service at a lower cost to customers. Everyone involved wins. Some interesting side benefits of multitenancy are improved quality, user satisfaction, and customer retention. Unlike single-tenant applications, which are isolated silos deployed outside the reach of the application provider, a multitenant application is one large community that is hosted by the provider itself. This design shift lets the provider gather operational information from the collective user population [7] (which queries respond slowly, what errors happen, etc.) and make frequent, incremental improvements to the service that benefit the entire user community at once. Two additional benefits of a multitenant platform-based approach are collaboration and integration. Because all users run all applications in one space, it is easy to allow any user of any application varied access to specific sets of data. This capability greatly simplifies the effort necessary to integrate related applications and the data they manage. Raw computing clouds are machine-centric services that provide on-demand infrastructure as a service (IaaS) for the deployment of applications [12]. Such clouds provide little more than the computing power and storage capacity needed to execute virtual servers that comprise an application. Some SaaS vendors looking for a quick go-to-market strategy avoid the challenges of developing a true multitenant solution and choose to deliver single-tenant instances via IaaS. Multitenancy is practical only when it can support applications that are reliable, customizable, upgradeable, secure, and fast [10].

3 Multi-Tenant Approach

Multi-tenancy is that type of case where a lot of users, typically distinct, use the shared resource. Discussing with our past analogy, we can very easily think about the multi-tenancy by only taking the relationship into an apartment building and its tenants. In the apartment building, each tenant can personalize their own apartment to a definite degree and feel assured behind their door. The reason is that the building owner can divide the costs of property conservation among its tenants; the owner can keep rents which is very reasonable for the tenants while still making a gain [4].

Many up to date, business, on-demand applications are multi-tenant applications. Multi-tenant, web-based applications fulfill the requirement of many application users. At home, we access the Google Apps using any web-browser for our word processing, spreadsheet, and presentation software needs. At that time also, many different people across the world are also utilizing Google Apps to do their work, all without downloading, installing, and maintaining applications and the data [5].

In the enterprise, we may use the web-based Sales force CRM application for CRM requirements. At the same instant of time, many other companies are also utilizing the same Sales force CRM application instance for trading purposes, without maintaining applications and data for themselves.

From the application provider's points of view, the advantages of multi-tenancy mainly relate to operational and cost efficiencies. It's because of the reason that one instance of the application can very easily meet the needs of many tenants/sellers, thus granting consolidation of system administration tasks like as monitoring, performance tuning, software maintenance, and data backups. The main drawback of multi-tenancy is that application providers ought to learn and implement new type of patterns that are very tough. This process

can detain the timeliness of software releases and instigate problems that affect not only one, but all tenants using an on-demand service [7,8].

- The application scales and maintains to perform well as the user population increases in size.
- The work of one tenant does not give very much effect on the work of other tenants.
- Each and every tenant can utilize the application to meet their own needs.
- The application sustains the safety and privacy of each and every tenant's data, even though multiple tenants use one or more system resources such as database, application, and web servers.

4 Extreme Multi-Tenant Designs

Two disparate extremes that are appraised as multi-tenant are the Virtual Multi-Tenancy and the Organic Multi-Tenancy.

4.1 Virtual Multi-Tenancy

Virtual multi-tenancy is the first extreme multi-tenant design. This design occurs whenever virtualization is at the base of the system architecture. In this context, consider the applications multi-tenant that simultaneously accomplish within the virtual machine (VMs) on top of the identical virtual infrastructure. Virtual multi-tenancy takes place whenever virtualization is at the base side of the system architecture [6,14].

To clarify the virtual multi-tenancy, we have taken following example applications which are shown in Fig. 3 that utilize the Amazon Web Services (AWS).

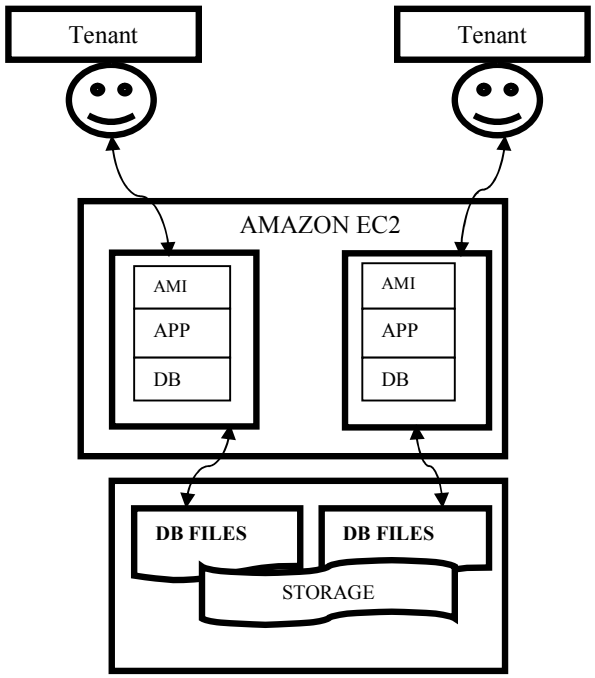


Fig. 3. Virtual mutitenancy [12]

4.2 Multi-Tenant at Two different Levels in the Stack

- Each AMI serve the CPU resources provided by Amazon EC2 [7]
- Each AMI serve the storage device resources provided by Amazon S3 [8]

Assume that the virtualization layer doesn't do its job exactly in organizing the shared resources–As for example, a safety hole in the virtualization software allows outrageous types to gain access to data within VMs they do not own. From a tenant's point of view, how this issue differs from the data leakage in a multi-tenant architecture that shares a deeper resource such as a database [11].

4.3 Organic Multi-Tenancy

Organic multi-tenancy is referred to as the other extreme design, a design in which each and every component inside the system architecture of a single software instance is being utilized among all the tenants. By each and every component such as hardware, operating systems, database, database schema, tables within the database schema, database servers, application servers, web servers, load balancers, firewalls, etc. The description of organic is used because this design, dissimilar with the virtual multi-tenancy, demands software developers to accurately consider multi-tenant design patterns and code them into an application [9]. Organic multi-tenancy occurs at the time when developers explicitly appraise and code multi-tenant design patterns into a fabric of an application.

To clarify organic multi-tenancy, we have taken the following application (Fig. 4) that uses Force.com.

Virtual multi-tenancy permit us to easily SaaS if an prevailing single-tenant application will work very little because a minimal amount of application redesign is required. In virtual multi-tenancy, an instance outage gives adverse effects only on the tenant using that specific VM [12].

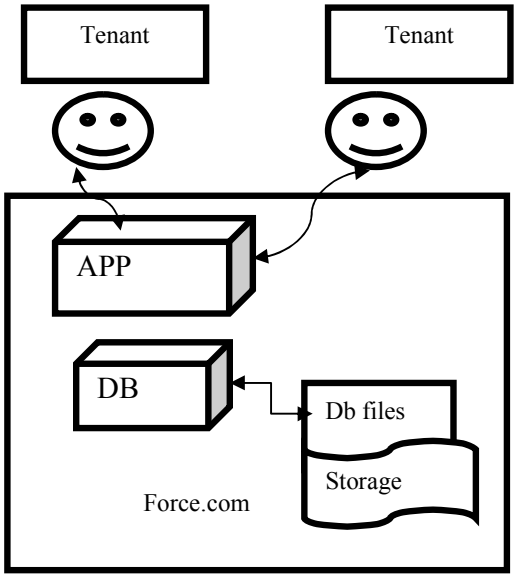


Fig. 4. Organic multitenancy [13]

Ultimately, organic multi-tenancy is the most operationally well organized and cost-efficient slant for new applications. If a failure is occurred in the system it may affects all the tenants in private cloud sector.

5 Proposed Scheme

Multi-tenancy is the key decisive of SaaS effectiveness. This ability is to have multiple concerns called as tenants in the SaaS nomenclature, coincide on the identical application as shown in the Fig. 5 without undermining the security of data for those companies explains the application as a multi-tenant one. There are many levels of Multi-tenancy like as Simple Virtualization in the Cloud where sharing is done with the hardware only. Single application with different databases per tenant; Single application and Shared database (highest efficiency, true multi-tenancy) is shown in this Fig. 5.

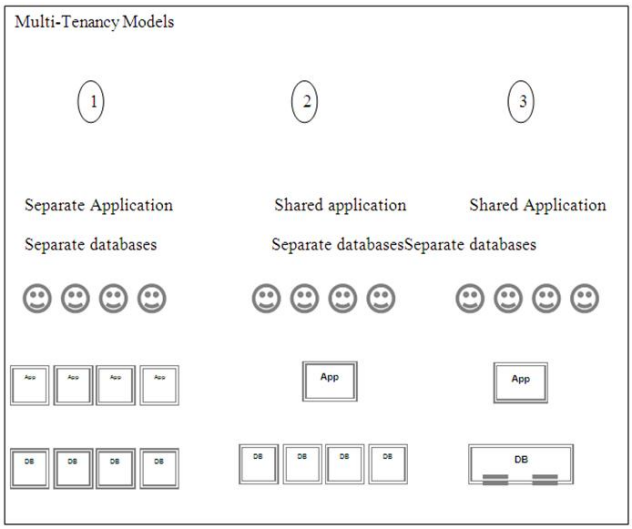


Fig. 5. Multi-tenancy model

Multi-tenant environment uses a single application and customized it by using different organization as because each have a distinct instance, a single and shared plate of software and hardware as shown in Fig. 6.

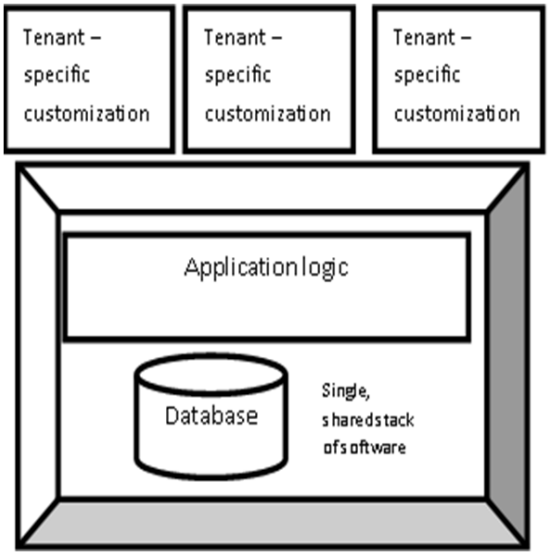


Fig. 6. Multiple tenants share the common software and hardware stack

In proposed scheme the platform Linux has been used with a database using MySql. The apache server is installed which will run our application built in PHP. Various tenants can access the application. The layered architecture is as follows as depicted in Fig. 7.

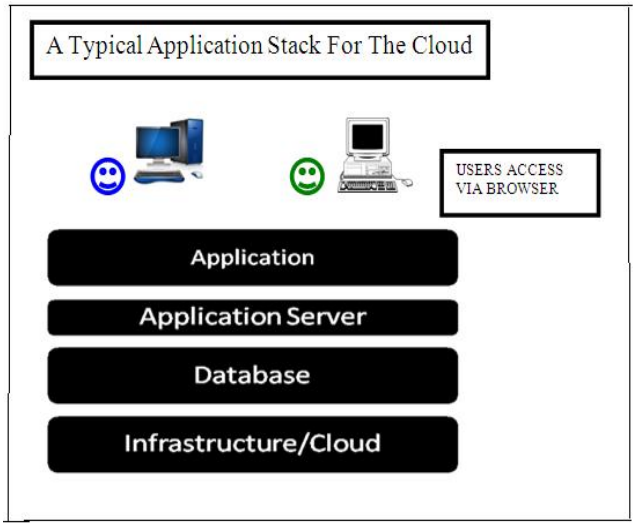


Fig. 7. Typical web application stack in the cloud

Mainly all the multi-tenant platforms are metadata driven. They manipulate metadata in the way so that it can configure the applications, platform for each individual tenant. As for example, the application carry a vendor entity, it is feasible to add additional attributes (for e.g. phone number, email) to the vendor only for one of the tenants (e.g. a firm using the application). These additional attributes are explained in the data center. Each and every part of the application can be specially made for specific tenants. Tenant-determined customizations are not made by duplicating all the application logic and converting it. Only the required changes are explained, they 'override' the prevailed logic.

Not only the logic can be tailored, it is very easy to apply the tenant-specific data model changes too. As all tenants' works on the same database, this database is required to be pliable and metadata driven to assemble disparate structures to disparate tenants.

- Multi-tenancy is very easily compared for creating the Java subclasses and injecting the objects of all these classes at runtime based on the tenant using that part of the code.
- In platform based model interpretation of multi-tenancy can be achieved by creating the model parts overwrites on the core model. If the platform needs to accomplish a business rule, firstly it looks on a customized rule exists for the present tenant. If so, this rule will be carried out, if not the default rule will be performed.
- Multi-tenant data storage can be executed by using a generic database structure and making queries at the runtime based on metadata.

Out of three primary Linux vendors, Canonical, Novell, and Red Hat, Red Hat has been selected for the scheme. It has many advantages:

- No lock-in with Red Hat.
- Different tracks of cloud sessions that relate its family of cloud outcome and services, across with its cloud strategy.
- A stable environment that permit you to run your workloads in your firm data center.
- Red Hat's dominance is its portfolio.

- Red Hat produce a lot of information about its cloud offerings,
- Red Hat is intricate in open standards for cloud computing.

6 Proposed Architecture

Multi-tenancy is a demanding technology which is mainly used to permit one instance of application helping different customers at the same time to share the Cloud resources and getting high operational planning. There are many distinct options of getting multi-tenancy, a multi-tenant architecture that grants managing distinct types of users of a system in a very easy way due to the large chunks of configurations that can be endorsed very easily has been proposed. Proposed layer is described within Fig. 8.

In proposed scheme a Linux Server has been installed with three virtual machines. That virtual machine can easily communicate with Server Main machine using IP address.

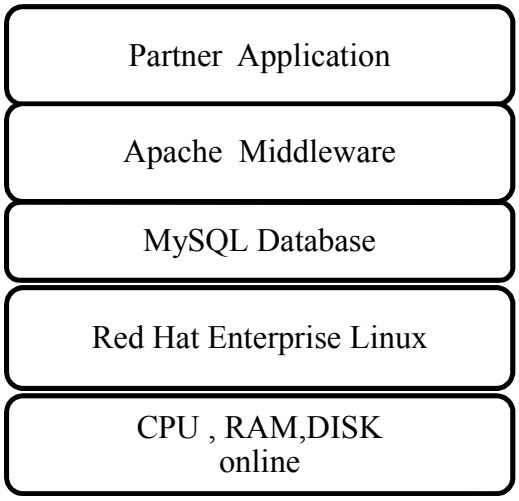


Fig. 8. Proposed layer

7 Clouds SIM: Simulator

Due to various limitations in the hardware it is not possible to show the results for large number of tenants. So a simulator known as the Cloud Sim has been used to show the results.

The conception of virtualization Associate in nursing multi-tenancy is enforced exploitation an application server as a middleware as shown within the Fig. 9. Multiple tenants share the employment of a multi-tenant application; however different tenants could have different goals and necessities.

A tenant is unlikely to have an interest. The provider implements the multitenancy, however they can accept the application to behave as the tenant is its whole sole user. The subsequent provides an inventory of the foremost important goals and necessities from a tenant's perspective.

7.1 Isolation

This is the foremost vital demand in a very multi-tenant application. Individual tenants' don't wish the activities of different tenants to have an effect on their use of the applying. They conjointly have to be

compelled to take care that different tenants cannot access their information. Tenants wish the applying to look like they need exclusive use of it.

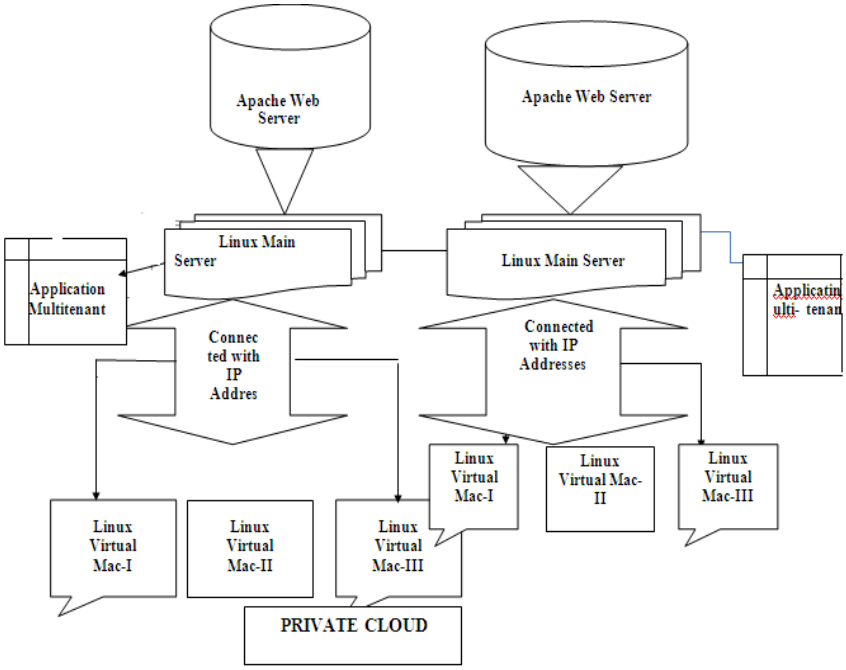


Fig. 9. Implementation of proposed architecture

7.2 Availability

Individual tenants wish the applying to be perpetually accessible, may be with guarantees outlined in associate SLA. Again, the activities of different tenants shouldn't have an effect on the other application.

7.3 Scalability

Multiple-tenants share multitenant application, a personal tenant can expect the application to be scalable and be able to meet his level of demand. The presence and actions of different tenants shouldn't have an effect on the performance of the applying on the other application.

7.4 Costs

One in every of the expectations of employing a multi-tenant application is that the prices are going to be under running an obsessive. Single-tenant application as a result of multitenancy permits the sharing of resources. Tenants conjointly got to perceive the charging model so they will anticipate the seemingly prices of using the application.

7.5 Customizability

An individual tenant could need the flexibility to customize the applying in numerous ways that like adding or removing options changing colors and logos, or may be adding their own code or script.

7.6 The Provider's Perspective

The providers of the multi-tenant applications also will have goals and needs. The subsequent provides an inventory of the foremost vital goals and requirements from a provider's perspective.

7.7 Meeting the Tenants' Goals and Requirements

The Service providers should make sure that the application meets the tenant's expectations. A provider could supply a formal SLA that defines the application how can meet the tenant's needs.

7.7.1 Profitability

Weather the provider provides the application in form of commercial service they desire to obtain return with an appropriate rate on the investment. Revenue from the application must be enough to hide both the capital and running costs of the application.

7.7.2 Billing

Cloud Service providers want the simplest way to bill the tenants. This might need the applying to observe resource usage if the provider doesn't wish to use a set rate charging approach.

7.7.3 Multiple service levels

The provider may want to offer various services on basis of different monthly rates, such as a standard or a premium subscription. These subscription levels may join different usage limitations of different SLAs, different functions or specify some combination of these factors.

7.7.4 Provisioning

Provider should be able to provision new tenants for the application. If there is little variety of tenants, this might be a manual method. For multi-tenant applications with an oversized variety of tenants, it's typically necessary to change this method by enabling self-service provisioning.

7.7.5 Maintainability

The provider should be able to upgrade the application and perform different maintenance tasks whereas multiple tenants are victimization it.

7.7.6 Monitoring

The provider should be able to monitor the application in the least times to spot any issues and to troubleshoot them. This includes monitoring however every tenant is victimization the application.

7.7.7 Automation

In addition to machine controlled provisioning, the provider might want to change different tasks so as to provide the specified level of service For example; the provider might want to change the scaling of the application by dynamically adding or removing resources as and after they are required.

7.8 Identification as Tenant in a Web Role

Every cloud service should have a novel DNS name. Therefore, if you have got one tenant per cloud service, every tenant will use a novel DNS name to access its copy of the application However, if you have got multiple tenants sharing net roles among a cloud service, you want to have a way to spot the tenant for

every net request that accesses tenant specific knowledge. Once you recognize that tenant the online request is related to you'll be able to make sure that any queries or knowledge updates operate solely on the information or different resources that belong there to tenant. It is your responsibility to make sure that your net roles will determine the tenant within the request, and to make sure that net roles preserve the isolation between your tenants.

Number of options to identify the tenant from a web request are-

- Authentication**.**
- The URL path**.**
- The subdomain**.**
- A custom domain

7.9 Goals and Requirements that Relate the Application Partitioning Queues and Worker Roles

In order to activate, priority is given to premium subscribers within the application there are two choices:

The first possibility is to use two totally different working roles with two queues, one for tenants with standard subscriptions and other with non-standards subscriptions.

7.10 Procedure

The virtualization and Multi tenancy has been implemented using Red Hat Linux and making a virtual machine.

Steps involved are:

1. Installing Linux (preferably Red Hat)
2. Installing YUM Server
3. Installing HTTP Server
4. Installing DNS server
5. Installing DHCP Server
6. Installing FTP server

7.11 Scripts for Cloud SIM

We can run cloud SIM script to show the results. In our implementation we have copied this script on the main server in the folder named examples in cloud SIM. When we run our multitenant application, this script automatically gets linked to the virtual machines and the results are produced according to the number of cloudlets used and number of data centers used.

Software requirements: Java version 1.6 or newer.

Cloud SIM has been tested and ran on Sun's Java version 1.6.0 or newer. The following Algorithms is used.

7.12 VM Load Balancing Algorithms

The algorithms used in the other two are explained below.

7.13 Algorithm: Load Balancer

1. Index table of VMs and the state of the VM like either BUSY or AVAILABLE is maintained by VM Load Balancer
2. Data Controller receives new request.
3. Data Controller asks the VM Load Balancer for the next allocation.
4. VM Load Balancer parses the allocation table

If found:

a. The VM Load Balancer put back the VM id to the Data Center Controller
b. The Data Controller sends the request to the VM identified by that id.
c. Data Controller notifies VM Load Balancer of the new allocation
d. VM Load Balancer edit and modify the allocation table accordingly If not found:
e. The VM Load Balancer returns -1.
f. The Data Controller queues the request

5. When the VM completes processing the request and the Data Controller receives the response, it notifies the VM Load Balancer of the VM de-allocation.
6. The Data Controller checks if there are any waiting requests in the queue, it continues from step 3 else
7. Continue from step 2.

The throttling threshold maintained by this algorithm is 1. It can be modified easily to make the threshold a configurable value.

7.14 Monitoring Load Balancer

This load balancing rules maintain equal workloads on all the available VMs.

1. VM Load Balancer maintains an index load balancing table of existing VMs and the number of requests allocated to the VM. Initially all VM's have 0 allocations.
2. When a new VM is allocated from the Data Controller, it parses the load balance table and identifies the least loaded VM. If there are more than one VM, the first one identified is selected.
3. VM Load Balancer returns the VM id to the Data Controller
4. The Data Center Controller passes the request to the VM which is identified by that id.
5. Now Data Controller keeps notice the VM Load Balancer of the new allocation
6. VM Load Balancer edits and adds the allocation table increasing the allocations count for that VM.
7. When VM completes its processing, and the Data Controller receives the response, it notifies the VM Load Balancer of the VM de-allocation.
8. The VM Load Balancer updates the allocation count by decreasing one in the allocation table for the VM.
9. Continue from step 2.

7.14.1 Service broker algorithms

Currently there are 3 different implementations of Service Broker.

7.14.2 Service proximity based routing

This is the simplest Service Broker implementation.

1. Service Proximity Service Broker maintains an index table of all Data Centers indexed by their region.
2. When the Internet receives a message from a user base it queries the Service Proximity Service Broker for the destination Data Controller.
3. The Service Proximity Service Broker retrieves the region of the sender of the request and queries for the region proximity list for that region from the Internet Characteristics.

This list orders the remaining regions in the order of lowest network latency first when calculated from the given region.

4. The Service Proximity Service Broker picks the first data center located at the earliest/ highest region in the proximity list. If more than one data center is located in a region, one is selected randomly.

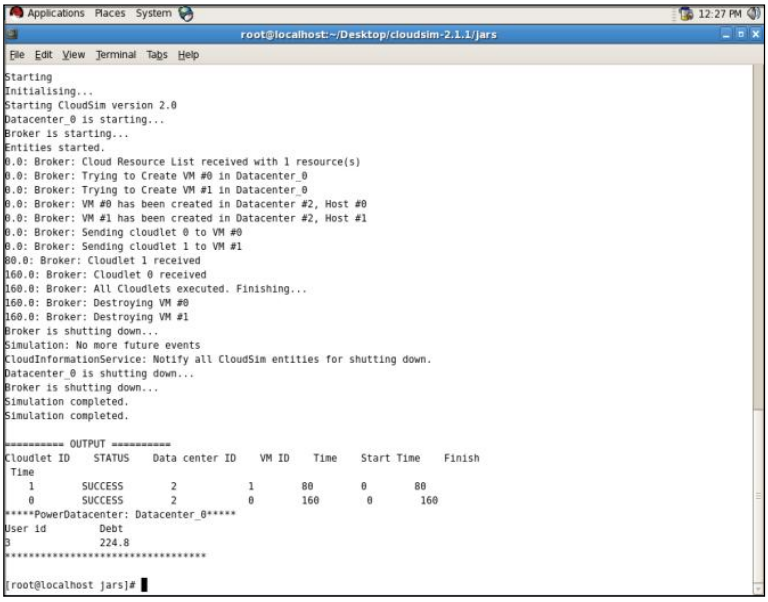
8 Results

The various snapshots show the results after running the Cloudsim scripts are:

Script1: Shows how to create a datacenter with one host and run one cloudlet on it. The output shows the status, Virtual machine ID, Start Time and Finish Time

Script2: Shows how to create a datacenter with one host and run two cloudlets on it. The cloudlets run in VMs with the same MIPS requirements. The cloudlets will take the same time to complete the execution. The total debt increases as the number of cloudlets increases. Hence we can compare the performance when the number of cloudlets increases.

Script3: Shows how to create a datacenter with two hosts and run two cloudlets on it. The cloudlets run in VMs with different MIPS requirements. The cloudlets will take different time to complete the execution depending on the requested VM performance**.** The debt also increases.

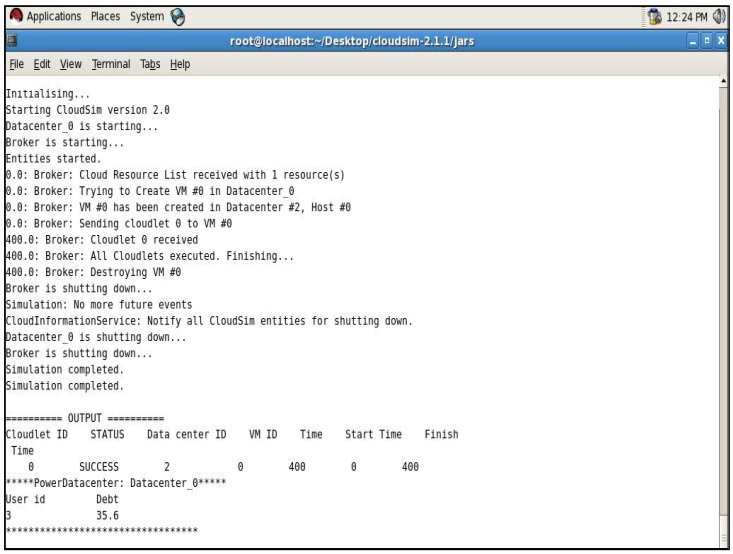


Result of script 1

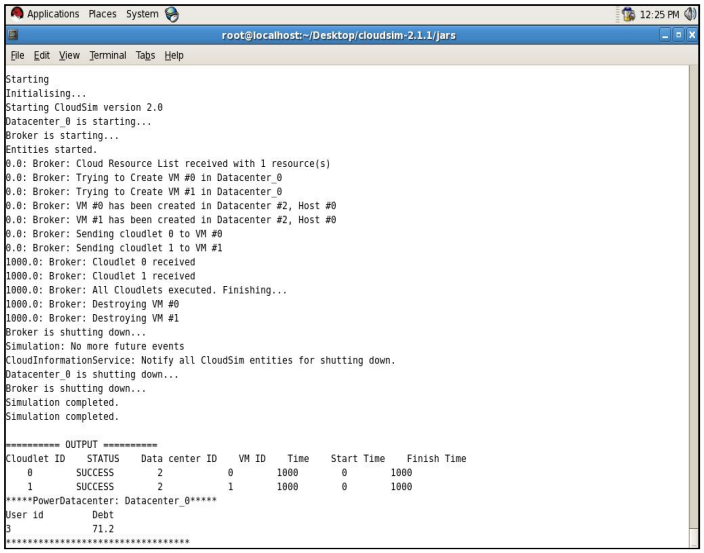
Script 4: This pageant how to build two datacenters with one host each and run the two cloudlets present on them. We can even link the two virtual machines with different data centers.

Script5: This pageant how to build two datacenters with one hosts each and run the cloudlets of two users on them.

Script 6: This shows the output when there are large numbers of cloudlets and many virtual machines are created.



Result of script 2



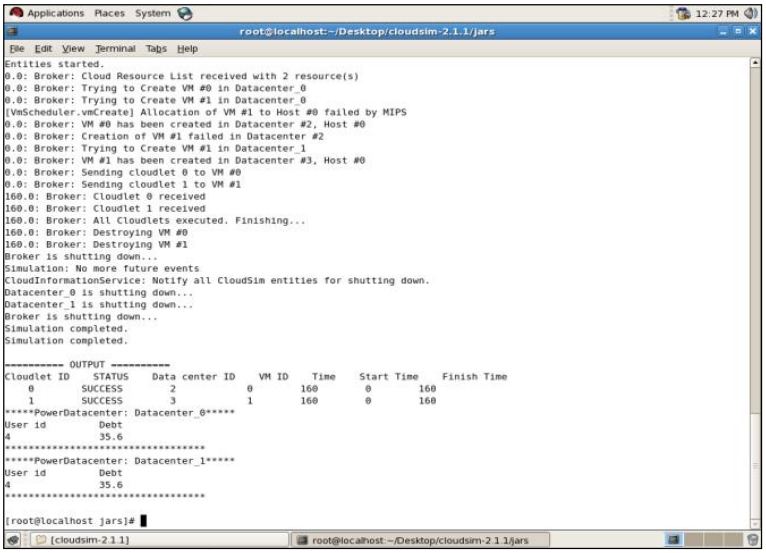
Result of script 3

8.1 Simulation Output

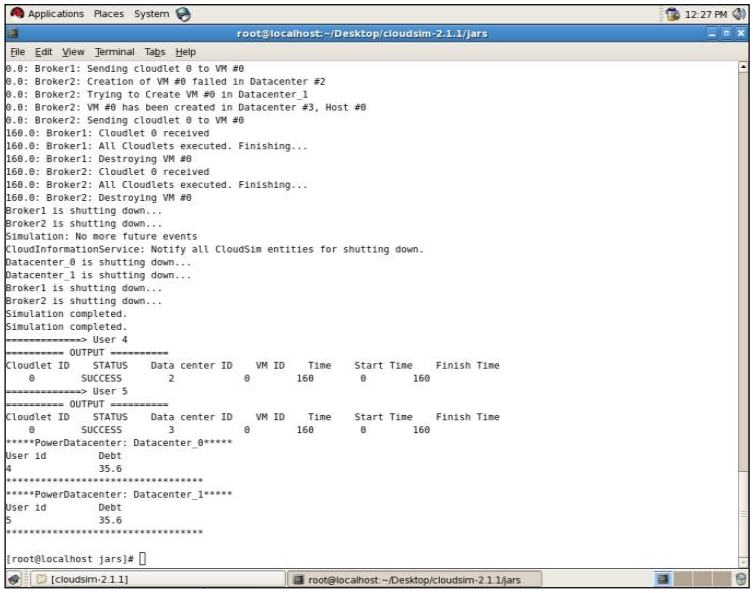
These are the statistical measures showed as a result of the simulation in the introductory version of simulator.

- Time of response of the simulated application.
- Minimum and maximum overall average, response time of all user requests simulated
- The broken down response time by user groups, situated within geographical regions
- The response time afterward broken down by the time reflecting the pattern of difference over the span of a day

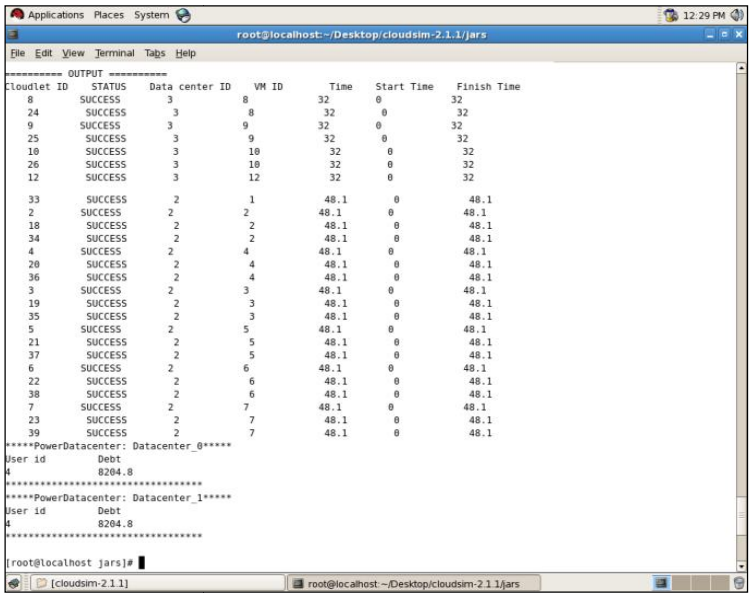
- The management patterns of the application
- Total number of users use the application at what time from distinct regions of the world, and the whole effect of that usage on the data centers hosting the application
- The time taken to service a user request by data centers.
- The comprehensive request processing time for the whole simulation
- The minimum maximum and average request processing time by each datacenter
- The response time changing pattern all along the day as the load changes
- The charge of the operation



Result of script 4



Result of script 5



Result of script 6

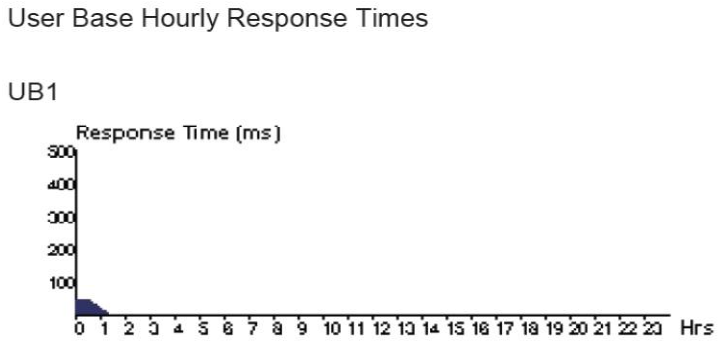
Simulation Results

Overall Response Time Summary

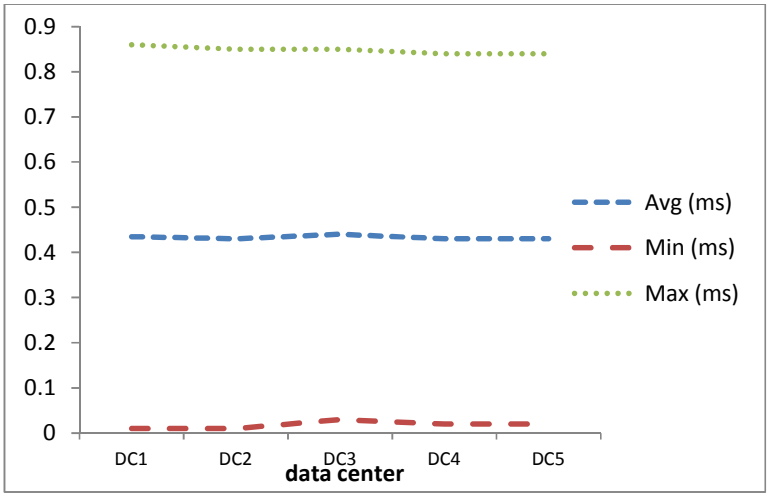
Factor	Average (ms)	Minimum (ms)	Maximum (ms)
Overall Response Time	310.60	40.10	617.61
Data Center Processing Time	0.32	0.01	0.86

Response Time by Region

User base	Average (ms)	Minimum (ms)	Maximum (ms)
UB1	50.01	40.10	60.61
UB1	200.68	161.11	247.11
UB1	293.33	241.61	369.12
UB1	500.76	390.11	617.11
UB1	500.26	382.60	610.12

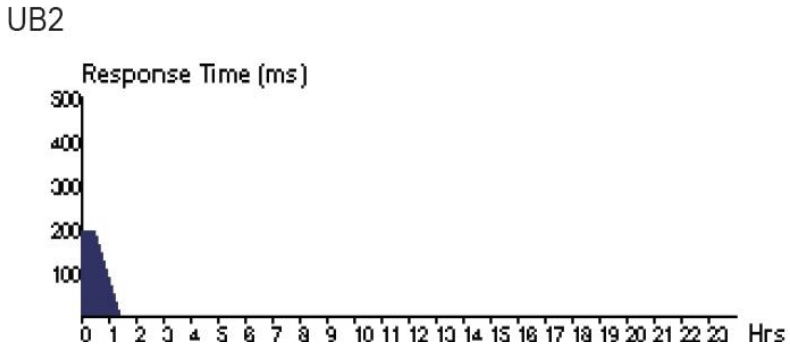


Data center	Avg (ms)	Min (ms)	Max (ms)
DC1	0.435	0.01	0.86
DC2	0.43	0.01	0.85
DC3	0.44	0.03	0.85
DC4	0.43	0.02	0.84
DC5	0.43	0.02	0.84



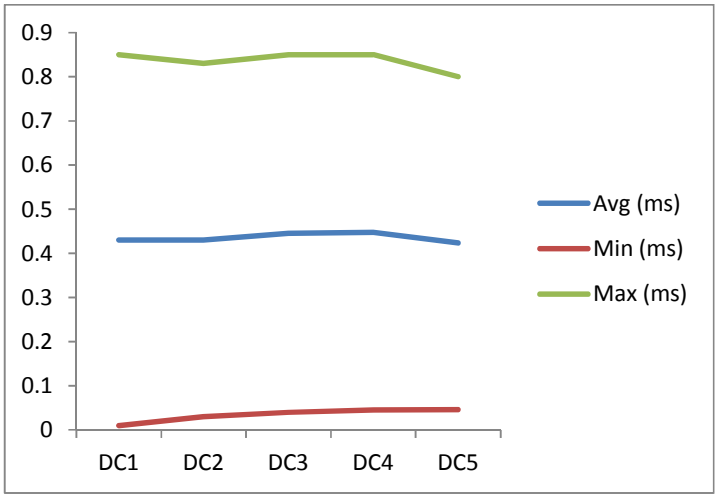
Graph 1. Represents response time from ub1

The response time is almost stable and there is slight burst at data center 3. This can be attributed due to sudden flush of tenants at this level at some point of time.



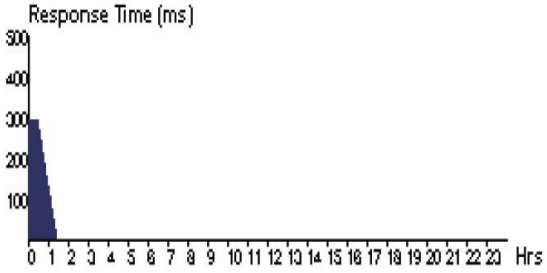
Data Center	Avg (ms)	Min (ms)	Max (ms)
DC1	0.43	0.01	0.85
DC2	0.43	0.03	0.83
DC3	0.445	0.04	0.85
DC4	0.4475	0.045	0.85
DC5	0.423	0.046	0.8

The response time is almost stable and there is slight decay at data center 5. This can be attributed due to instant response at this level at some point of time.

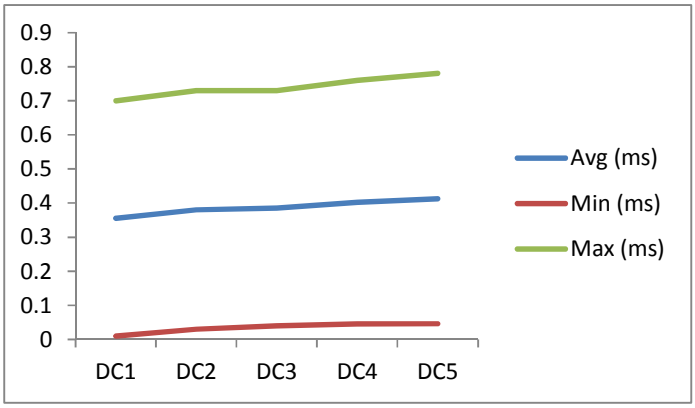


Graph 2. Represents response time from UB2

UB3

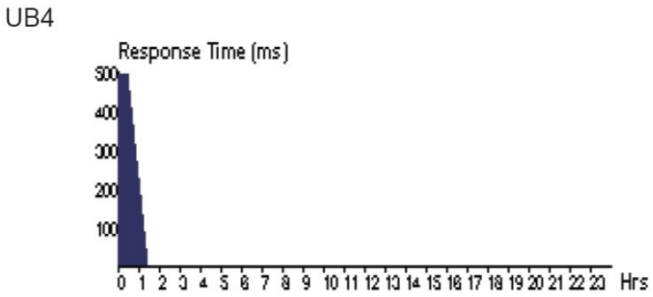


Data center	Avg (ms)	Min (ms)	Max (ms)
DC1	0.355	0.01	0.7
DC2	0.38	0.03	0.73
DC3	0.385	0.04	0.73
DC4	0.4025	0.045	0.76
DC5	0.413	0.046	0.78

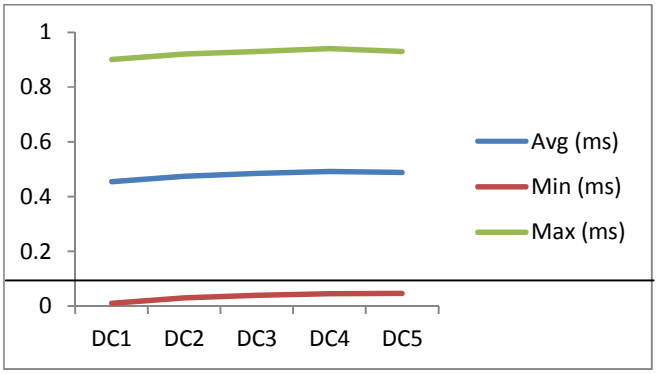


Graph 3. Represents response time from UB3

The response time is almost stable and there is slight increase towards last data center. This can be attributed due to delayed response at this level which can be attributed to busy processor or other delays.



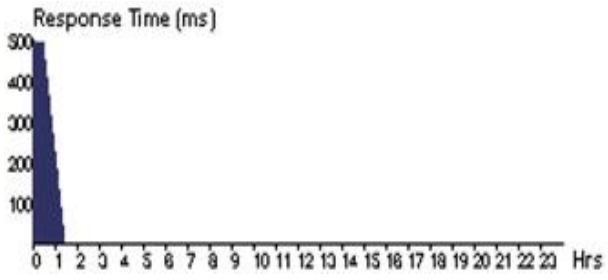
Data center	Avg (ms)	Min (ms)	Max (ms)
DC1	0.455	0.01	0.9
DC2	0.475	0.03	0.92
DC3	0.485	0.04	0.93
DC4	0.4925	0.045	0.94
DC5	0.488	0.046	0.93



Graph 4. Represents response time from UB4

The response time is almost stable. This graph depicts best performance at this level. All data centers are having almost equal time response for the processing.

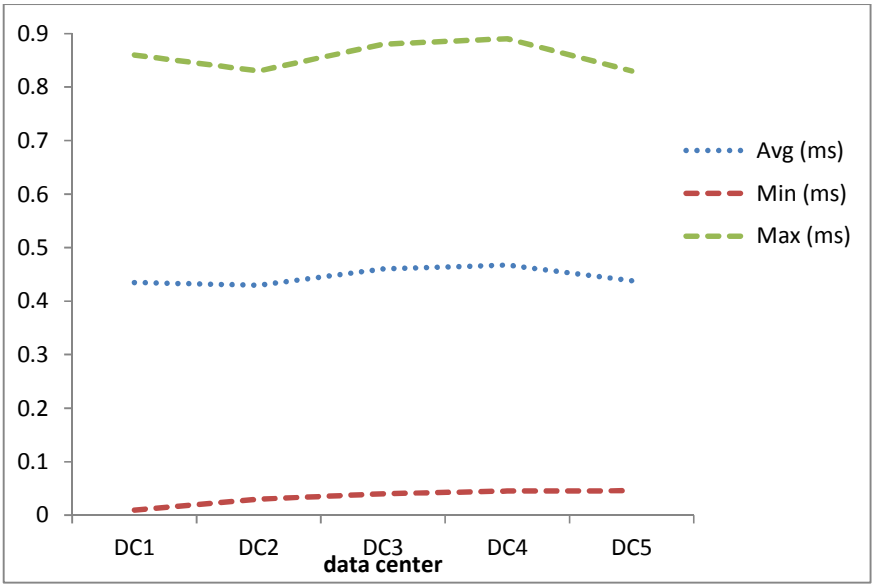
UB5



Data center	Avg (ms)	Min (ms)	Max (ms)
DC1	0.32	0.01	0.86
DC2	0.43	0.03	0.83
DC3	0.46	0.04	0.88
DC4	0.4675	0.045	0.89
DC5	0.438	0.046	0.83

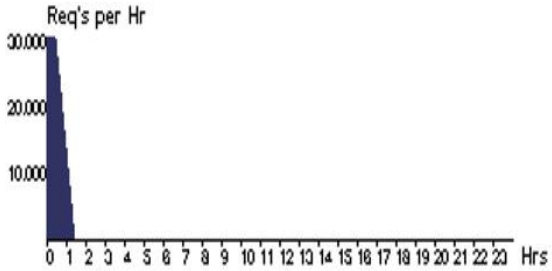
The response time is slightly wavered. Data center 2 shows a dip which in turn explains sharp response, also DC4 shows similar response.

Data Center Hourly Loading

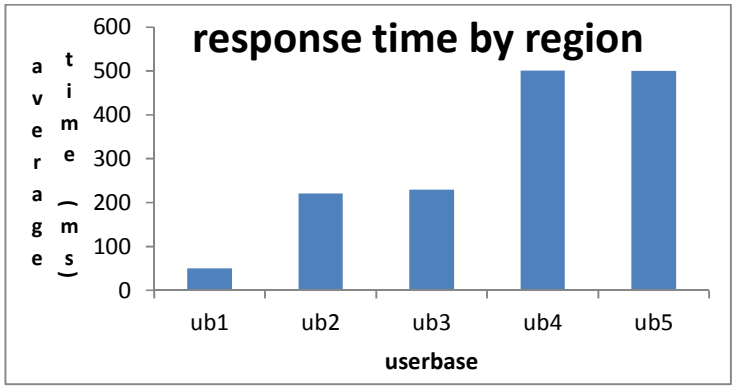


Graph 5. Represents response time from UB5

DC1



Total Virtual Machine Cost ($)	0.50
Total Data Transfer Cost ($)	0.32
Grand Total ($)	0.82



Graph 6. Loading time of data center 1

The similar trend occurs for all user bases. This means all user bases are giving true responses using multi tenancy concept and rising graph shows increase in response time region wise. It may be the reason that user base 5 takes maximum time because of the delay it gets in system set up and response.

9 Comparisons with SWAT Model

In this work a load balancing technique has been introduced. Though load balancing is very common for load sharing, in this technique a new method has been used. This method targets multi-tenancy and uses database technique to solve it. In this a new database replica swap has been introduced. This concept has been used as a building block. In most cases primary replica receives more work load and load on secondary replica is mostly leading to zero. In this technique a load sharing has been done to achieve the target. The migration of work is done by swapping primary and secondary replica of database for the purpose of fault tolerance. Using this technique a new algorithm called SWAT [14] has been introduced. It has been extension of habituating Microsoft SQL Server for cloud computing. Here a subset of tenants is selected for replica basis and desired results are achieved.

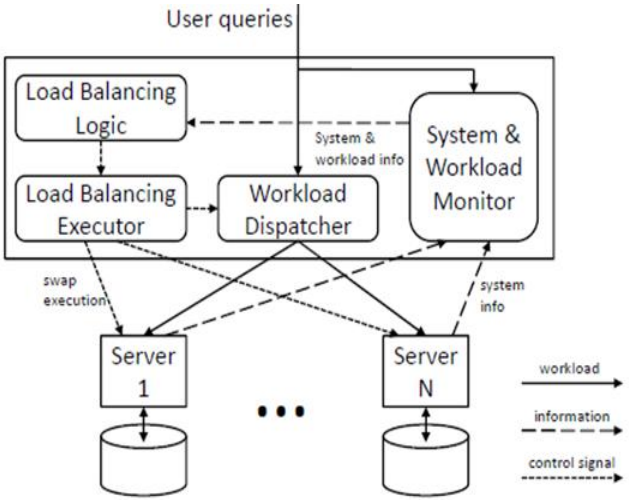


Fig. 10. SWAT architecture

Fig. 10 shows the architecture of SWAT model. A user workload reaches at the middleware layer, where a workload carrier routes it to the right database server, which apply it for shared algorithm architecture. The

system and monitor continually examine the workload level for each and every tenant and the server and sends the information and data to the load balancing module. This module spurt load balancing algorithm and assign a sequence of swap operators to be accomplished to load balancing executor. The executor spurts the swap operators attaining load balancing.

The Fig. 11 is the swap procedure of the model.

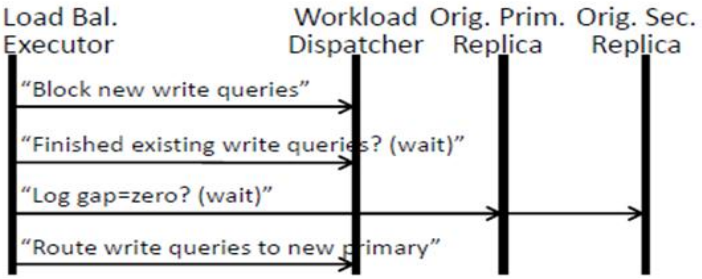
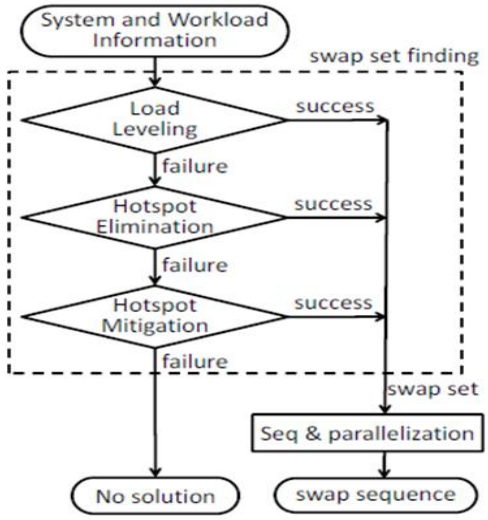


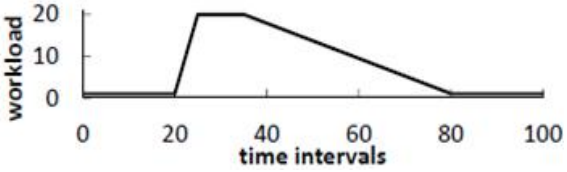
Fig. 11. Swap procedure

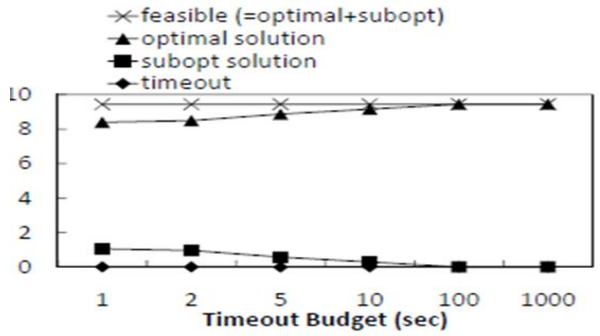
The SWAT overview can be described as:

Swap set concluding problem has been branched into three sub problems. First attempt is load leveling and balance the load. SWAT is dependent on load balancing a lot. Second component is elimination of hotspots. This eliminates overload. The last option is hotspot mitigation. This minimizes the overload.

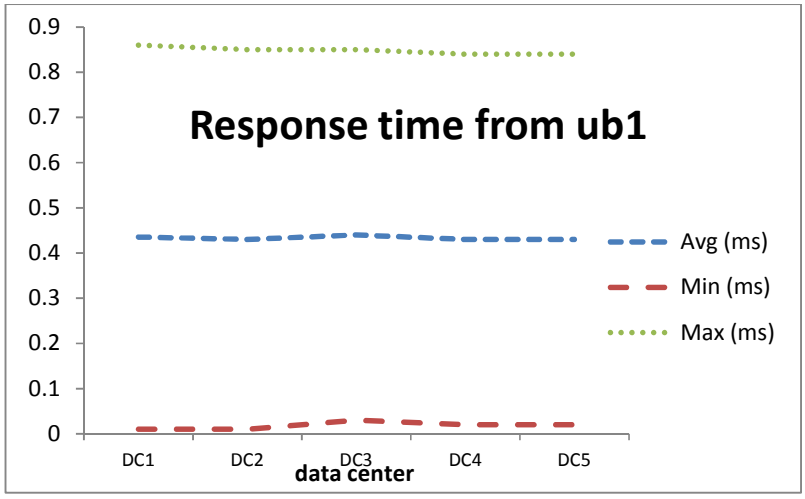


This is the workload multiplication factor of SWAT

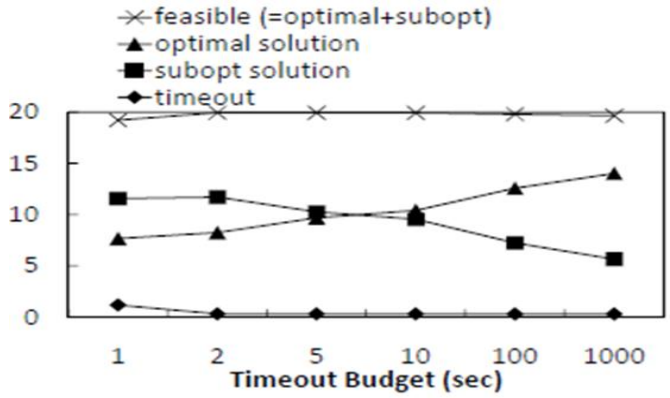


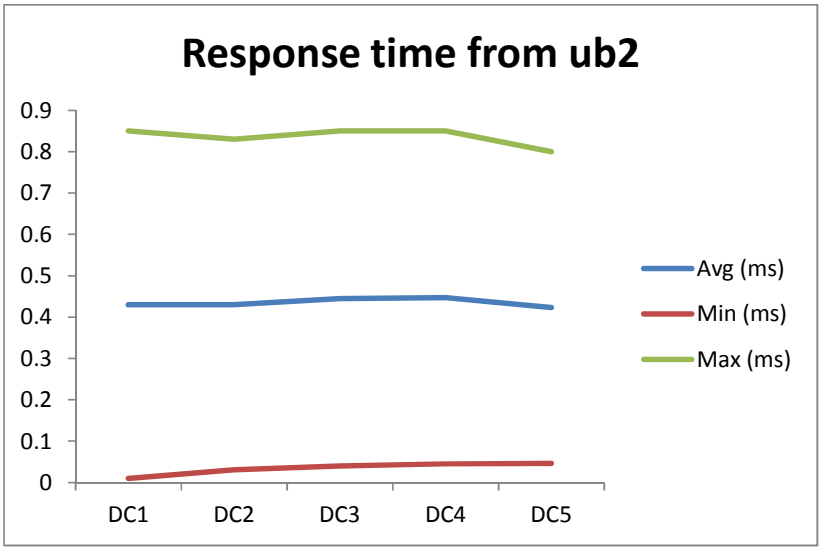


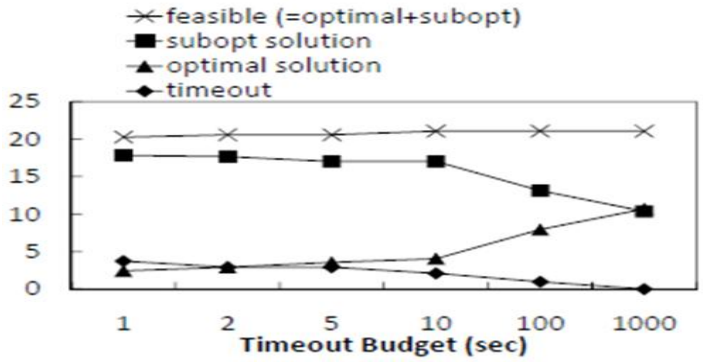
Response time by region 1 (50 tenants as per SWAT model)

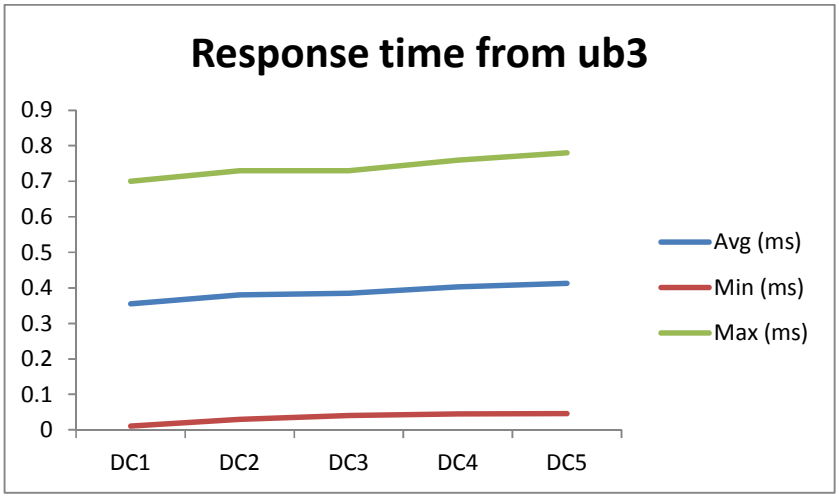


Response time form UB1 using the proposed scheme. In the above comparison timeout has not been taken as it was not coming into feasible measures in our case. In most cases our results match the SWAT model. The slight decline can be contributed to the fact that in our cases multi tenancy was more randomly selected. The same procedure was repeated for three more cases and results have been shown as:




Response time from ub2
0.9
0.8
0.7
0.6
0.5
0.4
0.3
0.2
0.1
0
DC1
DC2
DC3
DC4
DC5
Avg (ms)
Min (ms)
Max (ms)


feasible (=optimal+subopt)
subopt solution
optimal solution
timeout
25
20
15
10
5
0
1
2
5
10
100
1000
Timeout Budget (sec)


Response time from ub3
0.9
0.8
0.7
0.6
0.5
0.4
0.3
0.2
0.1
0
DC1
DC2
DC3
DC4
DC5
Avg (ms)
Min (ms)
Max (ms)

10 Conclusions and Future Work

Multitenancy is an architectural way that pays dividend to both request providers and users. The focus of the scrutiny was multitenancy; it has been attained employing several virtual mechanisms implementations. An increasing number of applications having multitenant have been entered using virtual setups and results have been shown using graphs. The single application instance adequately transform at runtime for any specific tenant at any given stipulated amount of time. The results are in close line with the study. Though the changes can be contributed to many reasons; as the SWAT model uses database replicas system, in our case pure simulations have been done which at time may not use the load balancing feature correctly. Still our work achieves the target in most of the cases. To verify the protection and isolation among customers or tenants of services from the dangers they pose to one another is the major issue. Amazon, is currently using Linux and Xen, in future it can implement KVM. With the blend of Xen and KVM, next step could be the mapping among various VMs concerning the virtualized environments.

Competing Interests

Authors have declared that no competing interests exist.

References

[1] Lijun Mei, Chan WK, Tse TH. A tale of clouds paradigm comparisons and some thoughts on research issues. IEEE Asia-Pacific Services Computing Conference. APSCC. 2008;464-469.

[2] Hong CAI, Ning Wang, Ming Jun Zhou. A transparent approach of enabling SaaS multi-tenancy in the cloud. IEEE 6th World Congress on Services. 2010;40-47.

[3] Chang Jie Guo, Wei Sun, Ying Huang, Zhi Hu Wang, Bo Gao. A framework for native multi-tenancy application development and management. 9th IEEE International Conference on E-Commerce Technology and The 4th IEEE International Conference on Enterprise Computing, E-Commerce and E-Services. 2007;551-558.

[4] Zeeshan Pervez, Sung young Lee, Young-Koo Lee. Multi-tenant, secure, load disseminated saas architecture. Proceedings of 12th International Conference on Advanced Communication Technology (ICACT). 2010;1.

[5] Jack Brass. Physical Layer Network Isolation in Multi-tenant Cloud. International Conference on Distributed Computing Systems Workshops; 2010.

[6] Pankaj Goyal. Policy-based event-driven service oriented architecture for cloud services operation & management. Proceedings of IEEE International Conference on Cloud Computing. 2009;135-138.

[7] Guoling Liu. Research on independent saas platform. School of Information Science and Technology Shandong Institute of Light Industry Jinan, China; 2010.

[8] Qiang Li, Qinfen Hao, Limin Xiao, Zhoujun Li. Adaptive management of virtualized resources in cloud computing using feedback control. Proceedings of The 1st International Conference on Information Science and Engineering (ICISE). 2009;99-102.

[9] Multi-Tenant SOA Middleware for Cloud Computing. Proceedings of IEEE 3rd International Conference on Cloud Computing. 2010;458-465.

[10] Matthias Schmidt, Niels Fallenbeck, Matthew Smith, Bernd Freisleben. efficient distribution of virtual machines for cloud computing. Proceedings of 18th Euromicro Conference on Parallel, Distributed and Network-based Processing. 2010;567-574.

[11] Tharam Dillon, Chen Wu and Elizabeth Chang. "Cloud computing: Issues and challenges," 24th IEEE international conference on advanced information networking and applications. IEEE Computer Society; 2010.

[12] Zhen Chen et al. "Collaborative network security in multi-tenant data center for cloud computing". Published In IEEE Xplore. 2014;19(1).

[13] Jia Ru et al. **"**Software engineering for multi-tenancy computing challenges and **i**mplications**"** Pages 1-10 ACM digital library. NY USA; 2014.

[14] Hyun Jin Moon, Hakan Hacigemus, Yun Chi, Wang-pin Hsiung. SWAT: A Light weight load balancing method for multitenant databases. NEC Laboratories America. Cupertino. CA, USA; 2013.