Project Report: Provenance Engine Team 3

Amit Mendiratta, **Dan** Howard, **Roopak** Venkatakrishnan (Lead), **Roshan** George (Vice Lead), **Sashank** Reddy, **Shayan** Sinha, **Shyam** Prasad

NC State University, Computer Science Department

1 Abstract

The provenance engine in a cloud environment is responsible for measuring and recording pertinent information regarding all of the cloud resources. It is the heart and soul of measured service, and it informs nearly every aspect of cloud computing. Our team was tasked with understanding and enhancing the existing provenance engine for the Virtual Computing Lab (VCL) at North Carolina State University (NCSU). We needed to consider many of the components that make up a cloud computing environment, but our primary focus was on analytics, policy, and accounting. Most of our work was not aimed at client facing features. Instead, we focused primarily on features designed for administrator use.

We have finished the design, and have developed all of the code for generating and storing the required data. We have presented the VCL engineering team with wireframes and mockups for the front-end of our provenance engine enhancements. Our iterative development efforts were based upon the cycle of feedback and refinement derived from regular meetings with the engineers.

We have learned that it is important to track data, and that space is at a premium. In an ideal situation, all of the data regarding VCL could be retained forever, without adversely affecting system performance. In reality, choices have to be made regarding what data to keep, and how to store it in ways that are efficient and useful. Our implementation is designed to help the engineers analyze and provide an accounting of how resources are used. This information will be used to implement existing policy, and inform future policy.

Contents

List of Tables:

List of Figures:

2 Introduction

2.1 Problem Statement

In 2003, NCSU began development on a groundbreaking cloud computing environment known as VCL [1]. In 2008, after years of development, NCSU donated the code for VCL to the Apache Software Foundation[2]. At that time, the foundation accepted VCL as an official incubator project. In 2012, VCL became one of the foundation's top level projects [3]. Furthermore, in 2011, after many revisions, the National Institute for Standards and Technology (NIST) formally defined cloud computing as having five essential characteristics. These are: on-demand self-service, rapid elasticity, resource pooling, broad network access, and measured service [4]. Our project focuses on the last of these.

In order to provide measured service, a cloud computing environment must be able to record measured data points. When designing a provenance engine, a cloud architect needs to determine why, what, where, how, and when to record information. It is essential to do this in ways that are both useful and scalable. As with most problems in computer science, a cloud architect faces a time-space-complexity trade off when designing a provenance engine that meets these needs. Given the amount of time our group had for development, it was not practical to design an entire provenance engine. Our project's scope was limited to extending the functionality of the existing provenance engine for NCSU's VCL.

After meeting with the VCL engineering team, we were able to organize our project around four main goals. The first was to modify an existing VCL feature that displays information concerning the five most utilized images. We were asked to change the number of images from the top five to the top *n* images, where *n* represents a number chosen by the end user. The next two goals involved determining and displaying the load on each management node (MN), and the load on each virtual machine (VM). The final goal was to develop a system for tracking resource usage per user, as well as per user group.

2.2 Related Work

All cloud providers are interested in storing information regarding provisioning on their infrastructure, though the degree of detail stored varies depending on the purpose. While provenance information is essential for forensics, it is also useful in determining possible optimizations in use of the infrastructure.

An essential component of commercial cloud systems, according to the NIST definition, is metered usage and tracking. Consequently, all commercially available public clouds implement some form of billing and usage tracking.

In [5], Mills et. al. compare different virtual machine placement algorithms that aim to optimize the use of datacenter infrastructure by creating or migrating virtual machines as required. All of these approaches require provenance information regarding load on the machines hosting the virtual machines.

In [6], Wood et. al. propose the Sandpiper hotspot elimination system which uses virtual machine load information to determine if a VM should be migrated.

2.3 Motivation

There were two clearly identifiable motivations behind our project goals. They can broadly be defined as balancing and billing. These two concepts are related to many aspects of cloud architecture, but they are primarily associated with analytics, accounting, and policy. The VCL engineering team gave us some clear use cases for the functionality we would be generating. These use cases are listed below.

- Observing and balancing the load on VMs and MNs
- Determining over and underutilized resources based upon load
- Implementing a billing policy whereby VCL can accurately charge resource usage to users and groups
- Determine whether or not a group is making effective use of their resources (hardware and software)
- Gathering data to be used in business decisions, particularly whether to buy new equipment, or purchase access to resources on a third party cloud

There are other uses for the type of functionality we were asked to implement. The cases listed above help to lay a foundation for future work. Our group identified a few hurdles, and experienced a few false starts in trying to implement this functionality.

2.4 Issues

There were two types of issues we faced as a group. The first type had to do with our individual strengths and weaknesses. The second type arose from the problems we were trying to solve.

Issues of the first type were identified more easily, and manifested themselves earlier in the design process. These were largely based upon varying degrees of experience among the team members. For example, most of us had similar experience in working with databases, but there was a wide range of experience working with PHP and JavaScript. Additionally, we received some early feedback from the VCL team that helped us to communicate with them more efficiently, and more professionally. These challenges were overcome by meeting regularly as a group, and by recognizing how best to utilize our individual strengths.

Issues of the second type were more specific to cloud architecture. These issues were harder to recognize, and did not manifest themselves until later in the design process. Sometimes, we did not realize there was an issue until we had already tried a solution based on an incorrect understanding. Below is a list of the major issues we faced. These are discussed in detail later in this paper.

- In order to be OS-agnostic, we used black-box monitoring method but it wasn't clear if these numbers were good proxies for the statistics inside the VMs.
- Determining the time interval to sample at - too high and we have too much data, too low and much relevant information is lost.
- Using as few new tables as possible in the database.
- Determining how to compact the information in the database after a certain time period (since raw samples would take up too much space).
- Generating representative test data. Without access to actual data, it was not clear if our fake data was representative.
- Building a representative testbed involving more than one management node and more than one kind of hypervisor.
- Using the latest version of relevant software (KVM, for instance) without losing functionality.
- Understanding database behaviour. For instance, some database tables are frequently deleted from so the data there is not permanent.
- Understanding how mysql compiles and runs queries was something which we spent a lot of time on. This was necessary to optimize queries

2.5 Environment

Since our project deals with adding modules which helps the admin/management get more information about VCL, the environment we used was a smaller version of production VCL. We started off using a single VCL sandbox. However we needed to work with a multi management node setup and worked on setting up a "mini VCL" by tying together 3 VCL sandboxes.

3 Requirements

The project was divided into 3 major modules

- VM Statistics
  This module requires that the usage statistics of various VMs in the VCL environment can be tracked. This would help in tracking if the requirements for various images in terms of CPU, RAM etc are over or under provisioned. For example if the Xinu image is given 1GB ram for each reservation and we are able to see from the data this module acquires that all reservations seem to use only 512MB RAM at max, we can say that resources are over provisioned
- MN Load
  This module tracks the amount of work done by every management node in terms of the number of reservations processed by each management node. This is usually determined over a given period. This could potentially be used to track bursts of load at various

management nodes. For example the presence of a homework submission could cause a lot of reservations of image “X” just before the deadline. It would be very helpful to notice all of these reservations on an MN over a short period and then observe the load on the MN drop down until the next homework submission.

- Billing
  The billing module is intended to be used to track which images are being used by which users. Given a rate per minute per combination of user and image, it must be possible to determine the total cost accrued due to use of VCL resources by each user. This metric can then be used to determine whether certain images are in high demand. It can also be used to analyze use patterns in order to adjust resource allocation.

3.1 VM Usage Requirement

This section explains the technical details involved in tracking virtual machine usage statistics, which is useful in affecting load balancing by the VCL administrators. The module is developed and tested with VCL 2.3.1 version. The key objectives are:

- Identify the parameters and metrics to measure the load of the Virtual machines.
- Identify the granularity of the time-interval for getting accurate load statistics.
- Provide a visualisation of the VM load metrics on the User Interface.

Output Interface:

The output interface section, shown in table 1, describes the information that is displayed to users of the software.

ID	Requirement
OI 1	Output will be displayed via the VCL Web Interface.
OI 2	VM Load map displaying the CPU, Memory statistics per VMhost
OI 3	Admin or user with appropriate permissions able to view the map

Table 1 : Output interface specification for the VM Usage Module

3.2 MN Load Requirement

This section provides technical specifications for tracking the load on Management Nodes used in Apache VCL as deployed at (NCSU). The module was developed to work with version 2.3.1 of VCL. These components are tracked at the image level:

- The number of reservations on each management node during the specified interval
- The type of image on each reservation
- The location of the source of each reservation request (remote IP)

- The location at which each reservation is allocated (for practical purposes this would be the same as the location of the management node managing the request itself)

User Interface:

This section, detailed in table 2, describes the user-facing interface of the software.

Output Interface:

ID	**Requirement**
OI 1	Output will be displayed via the VCL Web Interface.
OI 2	Heatmap displaying the number of reservations at each management node
OI 3	Heatmap using number of reservations of certain image at each management node
OI 4	Weathermap of IPs connecting into VCL Management nodes

Table 2 : Output Interface Specification for the MN Load module

Input Interface:

The input interface section describes the information that a user must be able to enter into the software, shown in table 3.

ID	**Requirement**
II 1	Input will be obtained via the VCL Web Interface.
II 1.1	The type of image for which the heatmap should be generated if not “all images”
II 1.2	Start time and end time for a load period of particular interest
II 2	Input Interface for the weathermap -> date range and image in question

Table 3 : Input Interface specification for the MN Load module

3.3 Billing Requirement

User Interface:

This section describes the requirements for the user-facing interface of the software.

Output Interface:

The output interface section, shown in table 4, describes the information that is displayed to users of the software.

ID	**Requirement**
OI 1	Output will be displayed via the VCL Web Interface.
OI 2	The number of units of usage a user has accrued per image that they have used.
OI 3	Each user must be able to view OI 2 for himself.
OI 4.1	Administration staff must be able to view OI 2 for any user.
OI 4.2	Administration staff must be able to view OI 2 for any image.

Table 4 : Output Interface specification for the Billing module

Input Interface:

The input interface section, shown in table 5, describes the information that a user must be able to enter into the software.

ID	**Requirement**
II 1	Input will be obtained via the VCL Web Interface.
II 2.1	The rate per unit of usage for a specific user for a specific image.
II 2.2	The rate per unit of usage for all members of a specific usergroup for a specific image.
II 3.1	Administration staff must be able to specify II 2.1
II 3.2	Administration staff must be able to specify II 2.2

Table 5 : Input Interface specification for the Billing module

Storage Requirement

`OI 2` must be calculated at the end of each reservation and stored permanently. An estimate of `OI 2` for a specific reservation must be displayed to the user before reservation is made. If a reservation is extended, the actual end time of the reservation must be used to calculate `OI 2`.

4 System Environment

To develop these modules we needed a test bed VCL. For a simple scenario a VCL sandbox would have worked, but testing for a multiple management node architecture would not be possible with a VCL sandbox. Also it would make testing for different kinds of hypervisors harder. After discussion with VCL engineers it was decided that setting up an environment with a few VCL sandboxes which had 2 VMware based environments and one KVM based environment tied up into one “mini VCL” would be a good starting point for testing our code.

We got 3 long term reservations from VCL and loaded them in such a way that we had 2 VMware Sandboxes and 1 KVM Sandbox. We then updated the KVM Sandbox from VCL 2.3 to VCL 2.3.1. The next step involved tying up the 3 reservations with 1 Web interface and 1 database.

Thus the final overall architecture looks something like what follows, as depicted in Figure 1.

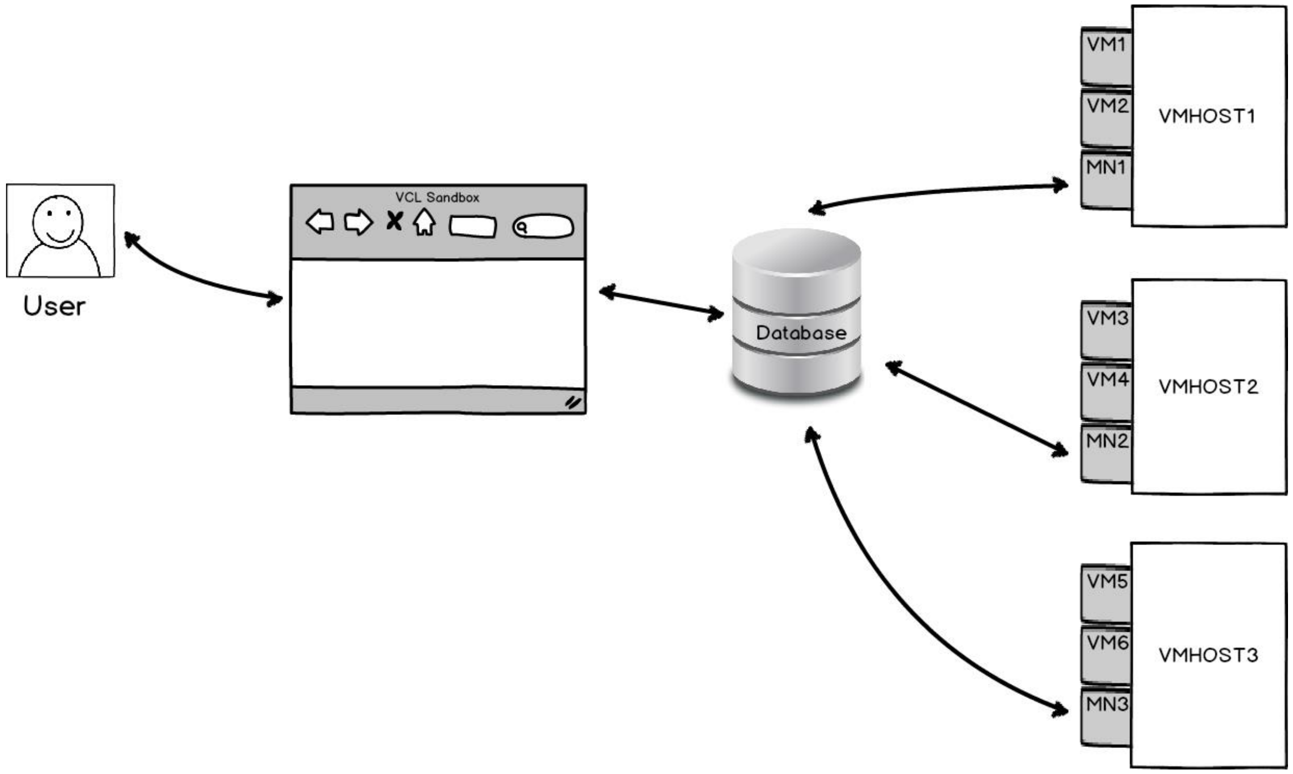


Figure 1 : Architecture of the VCL Testbed

5 Design, Implementation, and Results

5.1 General Provenance - Why

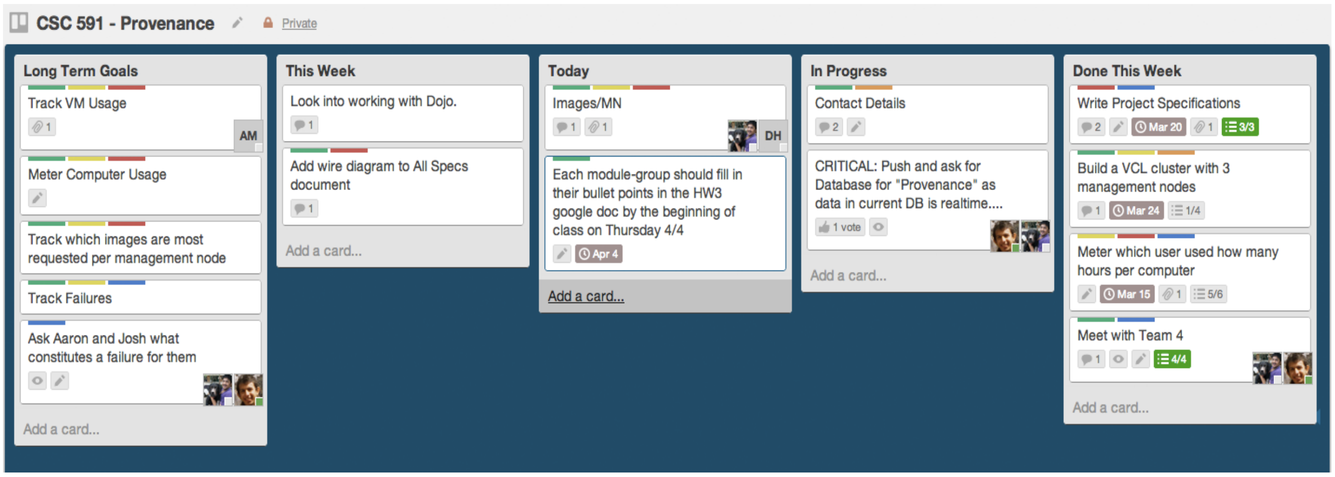
In order to provide measured service, it is necessary to record measured data. Without this data, one cannot feasibly understand or assess the quality of a cloud’s architecture. With this data, one can examine many aspects of a cloud including performance, accounting, and policy. The existing implementation of VCL at NCSU already had a database containing such information. The functionality we were tasked with incorporating required some additional tables and queries.

5.2 Dividing up our tasks

In order to manage the four goals associated with this project, we divided our team into three groups. We worked together to develop a solution to the first task, modifying the statistics page to display the top n images instead of the top five images. The remaining three tasks were divided among our three groups. We had two groups of two, and one group of three.

Because of the need for rapid, iterative development, our team decided to employ agile software development methods. We met every Thursday as a team, and sometimes more often than that. We focused our efforts on creating working software with incremental improvements. We communicated regularly with our clients via email and face to face visits. This interaction enabled us to verify that we were building what our clients asked us to build.

Trello:



CSC 591 - Provenance Private

Long Term Goals
- Track VM Usage (1) AM
- Meter Computer Usage
- Track which images are most requested per management node
- Track Failures
- Ask Aaron and Josh what constitutes a failure for them

Add a card...

This Week
- Look into working with Dojo. (1)
- Add wire diagram to All Specs document (1)

Add a card...

Today
- Images/MN (1) (1) DH
- Each module-group should fill in their bullet points in the HW3 google doc by the beginning of class on Thursday 4/4 (Apr 4)

Add a card...

In Progress
- Contact Details (2)
- CRITICAL: Push and ask for Database for "Provenance" as data in current DB is realtime.... (1 vote)

Add a card...

Done This Week
- Write Project Specifications (2) (Mar 20) (1) (3/3)
- Build a VCL cluster with 3 management nodes (1) (Mar 24) (1/4)
- Meter which user used how many hours per computer (Mar 15) (1) (5/6)
- Meet with Team 4 (1) (4/4)

Add a card...

Figure 2 : One week's task assignments on Trello

One of the tools we used to organize our project was Trello, depicted in Figure 2. Trello supports a project management method called Kanban that emphasises agile development practices. This method of management of tasks involves creating a card for each task as the task arises. Our workflow involved the following:

- Each task has a card created for it with relevant documents attached to it from Google Drive.
- Cards generally move from left to right, and any that are completed at the end of the week are archived for later reference.
- A card can be created in any of the lists.
- The first list is used to track current project progress on each of the sub-projects.
- Colour-codes are used to tag cards as requiring code, documentation, or analysis and to mark critical or blocking cards.

Design Details

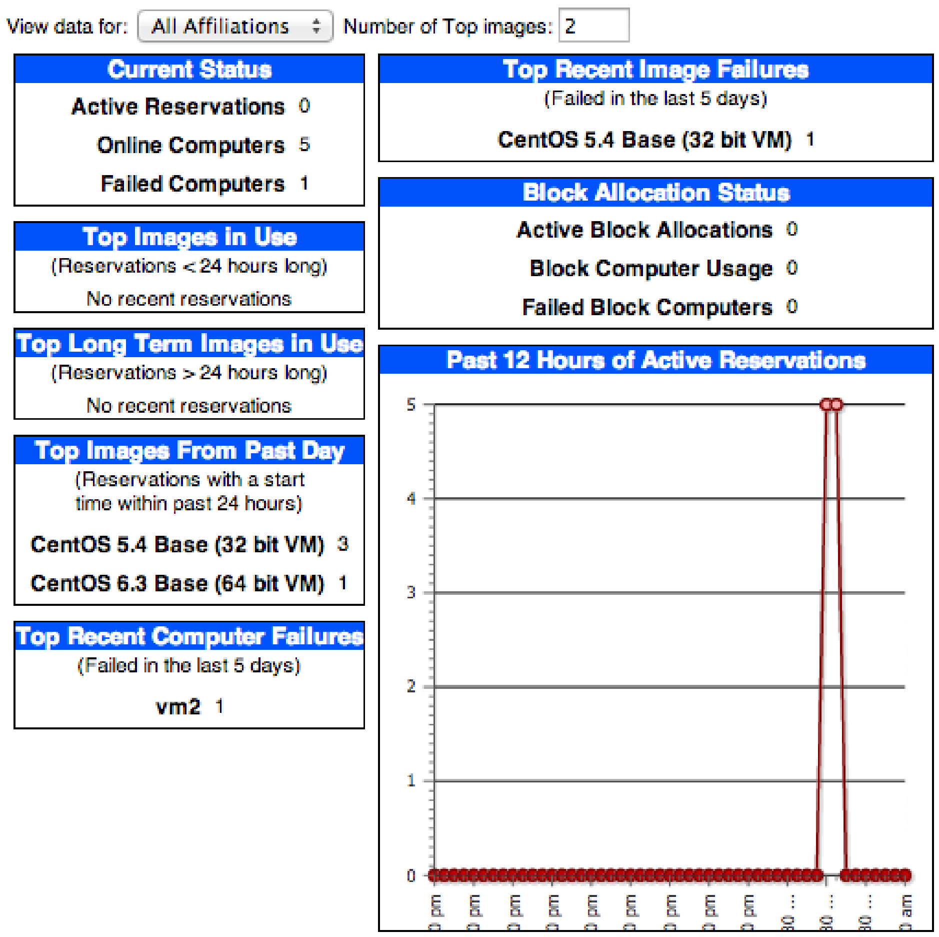
In the following section, we discuss our four project goals, and how we have worked to accomplish them. We begin each topic by describing the problem we were trying to solve. Then we discuss the iterative process we used to implement and refine our design, along with the activities and results that supported the process. We discuss what we tried, our successes and failures, and the direction of our future work.

5.3 Top n Images

This task was smaller and more simple than the remaining tasks. The VCL engineers presented it as a way to familiarize ourselves with the provenance environment, while adding useful functionality to VCL. As a group, we determined that this task could be accomplished by modifying the existing dashboard.js file.

The changes we made to this file include modifying the SQL query the script uses to retrieve the top images, as well as modifying the design of the page to accept user input. The modified dashboard view, shown in Figure 3, displays the top five images by default. It also provides a textbox that allows a user to change the number of top images to be displayed. This change updates all of the items relevant to displaying the top “n” images obtained from the textbox.

Upon completion of this task, we presented the new functionality to the VCL engineers for approval. They confirmed that we had correctly implemented the desired functionality. We performed manual testing to verify our results.


View data for: All Affiliations Number of Top images: 2

Current Status

Active Reservations 0
Online Computers 5
Failed Computers 1

Top Images In Use
(Reservations < 24 hours long)
No recent reservations

Top Long Term Images In Use
(Reservations > 24 hours long)
No recent reservations

Top Images From Past Day
(Reservations with a start time within past 24 hours)
CentOS 5.4 Base (32 bit VM) 3
CentOS 6.3 Base (64 bit VM) 1

Top Recent Computer Failures
(Failed in the last 5 days)
vm2 1

Top Recent Image Failures
(Failed in the last 5 days)
CentOS 5.4 Base (32 bit VM) 1

Block Allocation Status
Active Block Allocations 0
Block Computer Usage 0
Failed Block Computers 0

Past 12 Hours of Active Reservations

Figure 3 : A screenshot of the new Top Images statistic screen. The textbox at the top sets the number of images to be displayed

5.4 Load per MN

This module is used to determine and display the load on each MN. Equipping the VCL engineering team in this way will allow them to spread out VCL reservations and distribute the load equally. This is particularly useful in trying to distribute the load across different time periods. For example it may be interesting to note that certain image reservations happen at 8AM every morning. This information could be used to anticipate spikes in demand, and proactively redistribute resources.

1st iteration

Design

We initially assumed that the load on an MN would be defined as the average load across every VMhost it managed. To implement this strategy, we came up with a model that used the statistics generated by the VM Load team. We defined the load on each management node as the average load on each VMhost. The load on each VMhost was defined as the percentage of available resources utilized.

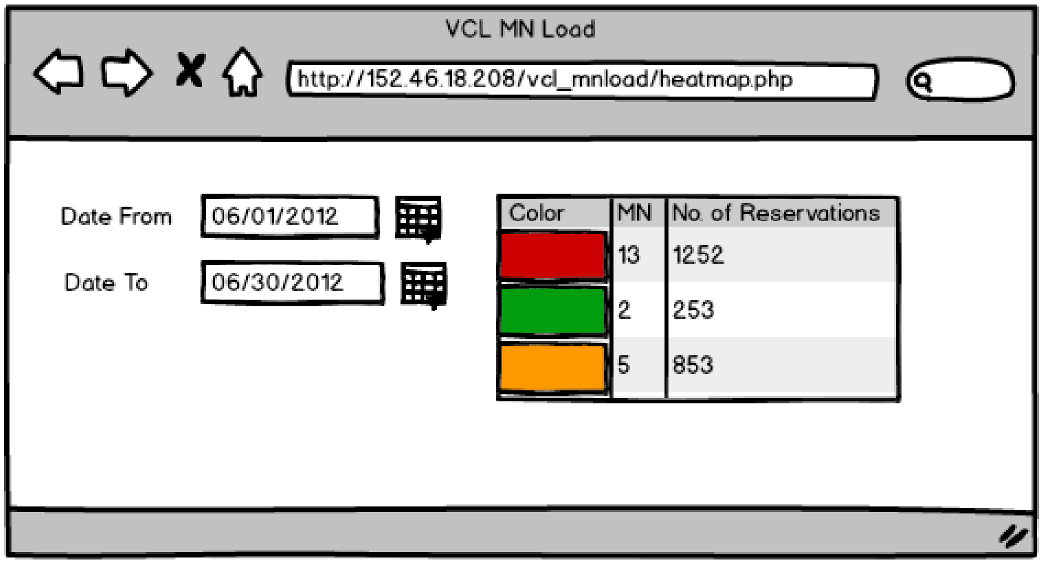
Result

During the next round of discussion with the VCL Engineers, we realized that this did not actually measure the load on each management node. The problem with this particular type of measurement is the fact that it did not take into consideration any work done by the management node. Instead, it looked at the work done by the various systems under a management node.

2nd iteration

Heatmap

After meeting with the VCL team, we realized that the load on an MN should be defined as the number of reservations processed by that MN during a specific time interval. Additionally, while working with the policy engine team (Team 4), we determined that we needed an "off-site database" for data storage and retrieval. The information would be write-once, then read only. We began working on wireframes for our heatmap and weathermap based on this design.



Date From 06/01/2012

Date To 06/30/2012

Color	MN	No. of Reservations
🟥	13	1252
🟩	2	253
🟧	5	853

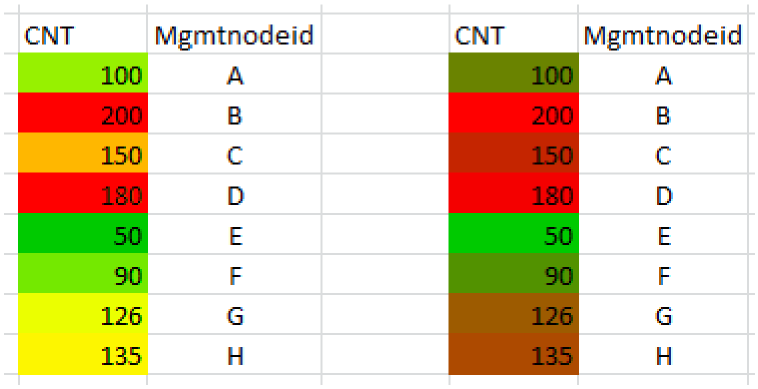
Figure 4 : A mockup of the heat map interface

3rd iteration

After discussions with the VCL engineers and our advisor, Mr.Streck, we decided to use the VCL database instead of creating our own separate off-site database. We had not previously realized that we could access and extend this database. We did this to avoid the duplication of data. Additionally, VCL backs up its main database to an "offsite" database at the Office of Information Technology(OIT).

With the source of our information determined, we began to write queries to generate data for a simple heat map. Additionally, we finished a mock up of the heatmap, which can be seen in Figure 4. Sample color schemes were generated based on input from the VCL engineering team.

After several iterations, a color scheme was chosen and work on the code for generating a heatmap began.

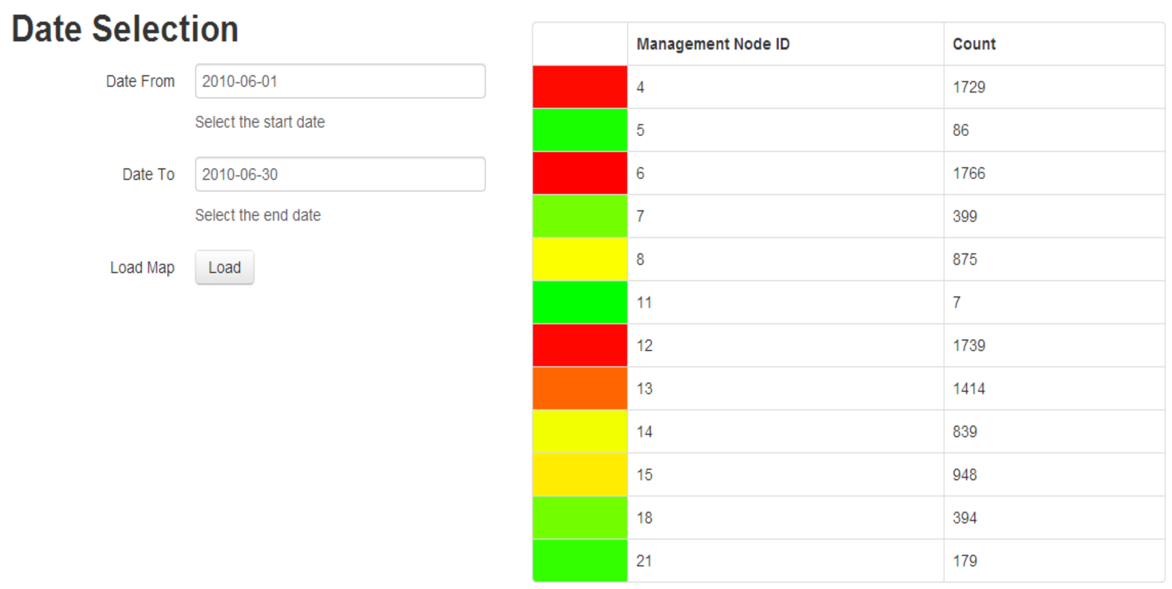


CNT	Mgmtnodeid
100	A
200	B
150	C
180	D
50	E
90	F
126	G
135	H

CNT	Mgmtnodeid
100	A
200	B
150	C
180	D
50	E
90	F
126	G
135	H

Figure 5 : An example of what the heatmap looks like

A tabular form of the heatmap, depicted in Figure 5, was chosen because of its potential for scalability. This design is well-suited to accommodate increases in the number of MNs as VCL expands. The design shown in Figure 6 was developed in order to elicit feedback from the VCL engineering team.



Date Selection

Date From	2010-06-01
	Select the start date
Date To	2010-06-30
	Select the end date
Load Map	Load

	Management Node ID	Count
🟥	4	1729
🟩	5	86
🟥	6	1766
🟩	7	399
🟨	8	875
🟩	11	7
🟥	12	1739
🟧	13	1414
🟨	14	839
🟨	15	948
🟩	18	394
🟩	21	179

Figure 6 : Model of simple heatmap generator

4th iteration

Weathermap

The weathermap was designed to use the remote IP of a reservation, as well as the IP of the management node that provisioned it. We intended to show a link for each reservation between its originating IP and the IP address of MN that provisioned the reservation. A GeoIP database from MaxMind was obtained for this purpose.

The VCL team informed us that the physical location of each MN, as well as the systems it manages are all in the same data center. We confirmed that it would be OK to assume that traffic to an MN itself would be very close to the traffic to an actual VM/Bare metal computer itself. Based on this information, it would not be ideal to use the location from the GeoIP database for the MN itself, because connections originating from the same NCSU network would not be displayed in a sensible manner. Table 6 shows the schema for the location of management node.

mnphysicallocation:

Column	**Description**
managementnodeid	The ID for the management node
latitude	latitude of location of management node
longitude	longitude of location of management node

Table 6 : The database schema of the mnphysicallocation table.

5th iteration

The GeoIP database did provide the physical location of the IP in question, however when trying to retrieve data for many reservations with one query, we noticed that this database was not optimized. It took about 3 seconds to get the location of one IP. This was too slow to use for our purposes, and would not scale well. We looked into various techniques that could be used to improve execution time. We finally decided to build indexes and write routines to optimize our queries.

Our next task was to display the information in a meaningful way. We decided to use use of the Google Maps API for this purpose, and began development along those lines.

Summary

After several iterations, we arrived at a design that met all of the stated requirements pertaining to the load on an MN. We provided mechanisms for measuring the load per MN that are based upon existing data in the database. We provided a heatmap and a weathermap for displaying those results. Our design evolved over time, and was based upon specific feedback obtained from the VCL team.

We have designed these features in a way that should scale well. The additional functionality we provided does not unduly increase the burden on the existing system. Our work has been integrated with the VM load and billing modules, and has been tested in a multi-MN environment.

5.5 VM Statistics

problem description

For the administrator to do effective load balancing of VM's, he needs to have statistics of load for all the VM's. Currently, the VCL design doesn't capture the load statistics of the vm's and as a result of this the VCL administrators cannot do an effective load balancing of the VM's. This brings us to the problem definition of collecting the load statistics of the VM's which can be used by the VCL administrators.

1st iteration

Design

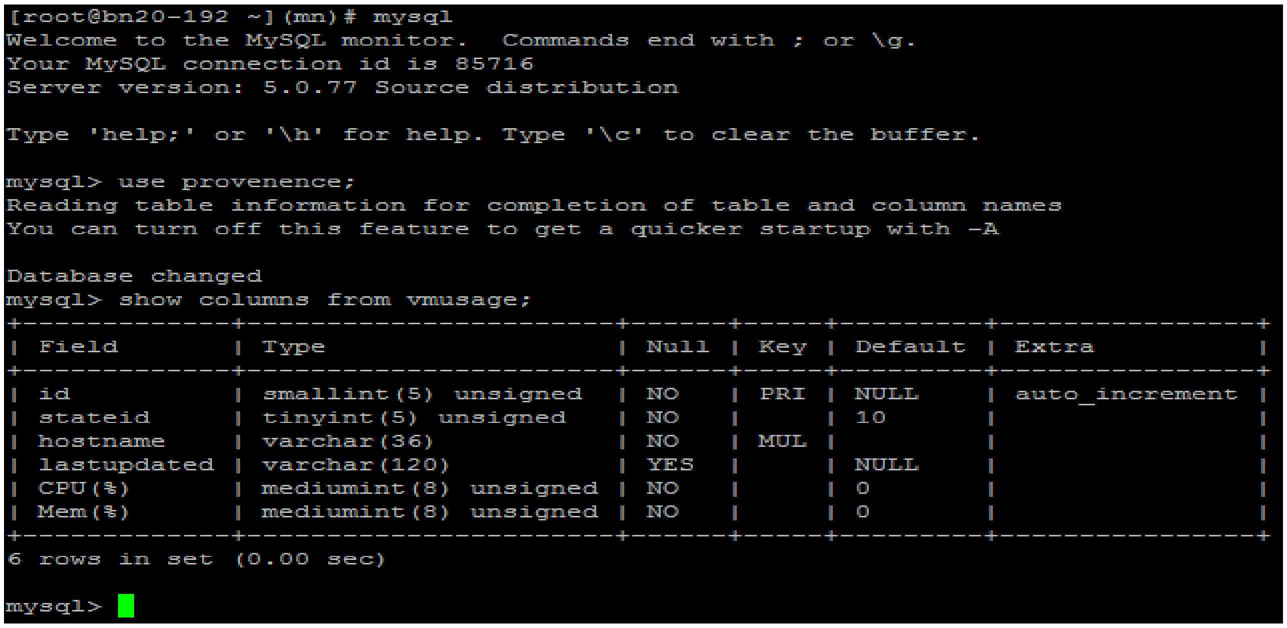
We identified CPU, Mem statistics as metrics for the measurement of load on the VM's. We gathered these metrics of each of the VM's from the VMhost and then captured the results into a new database. We planned to create two database tables name "vmusage" which gathers the CPU,Mem statistics and "VMmeta" which collects the VM characteristics of each of the VM. Furthermore we have a script which runs on management node and gathers the CPU/Mem utilization of each vm running on the vmhost and updates the data into the provenance database. And finally display these statistics on the UI in terms of graphs. To connect to the database we used CPAN libraries.[10]

Implementation

In order to capture the cpu and men utilization of each vm we tried fetching vm details using htop/top command [9]. On analyzing the results and also by studying few articles [ref] we realized htop/top is not reliable source of fetching these statistics. Therefore we decided to use "ps aux" command to get the cpu and men utilization for specific vm's[8] .We created three scripts. The main script "run.sh" runs as cron job with frequency of execution as 1 hour and remotely acceses the vmhost(via ssh) and gathers vm usage details and updates the statistics log. The log is maintained in the management node which gets rotated periodically. Finally the database is populated with the provenance data collected. Creating the script to collect the CPU and memory utilization was a challenging task in this iteration.

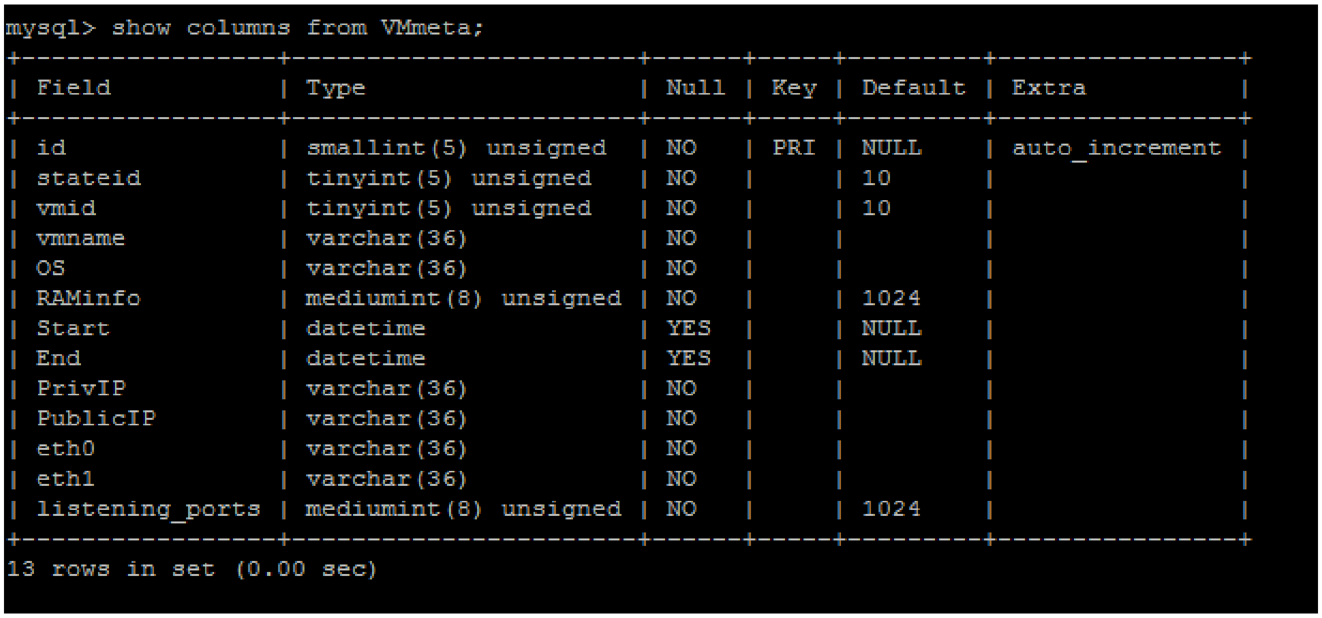
Results

At the end of this iteration we came up with two new database tables i.e. VMUsage and VMMeta with schemas as shown below in Figures 7 and 8:



```
[root@bn20-192 ~](mn)# mysql
Welcome to the MySQL monitor.  Commands end with ; or \g.
Your MySQL connection id is 85716
Server version: 5.0.77 Source distribution

Type 'help;' or '\h' for help. Type '\c' to clear the buffer.

mysql> use provenence;
Reading table information for completion of table and column names
You can turn off this feature to get a quicker startup with -A

Database changed
mysql> show columns from vmusage;
+-------------+-----------------------+------+-----+---------+----------------+
| Field       | Type                  | Null | Key | Default | Extra          |
+-------------+-----------------------+------+-----+---------+----------------+
| id          | smallint(5) unsigned  | NO   | PRI | NULL    | auto_increment |
| stateid     | tinyint(5) unsigned   | NO   |     | 10      |                |
| hostname    | varchar(36)           | NO   | MUL |         |                |
| lastupdated | varchar(120)          | YES  |     | NULL    |                |
| CPU(%)      | mediumint(8) unsigned | NO   |     | 0       |                |
| Mem(%)      | mediumint(8) unsigned | NO   |     | 0       |                |
+-------------+-----------------------+------+-----+---------+----------------+
6 rows in set (0.00 sec)

mysql>
```

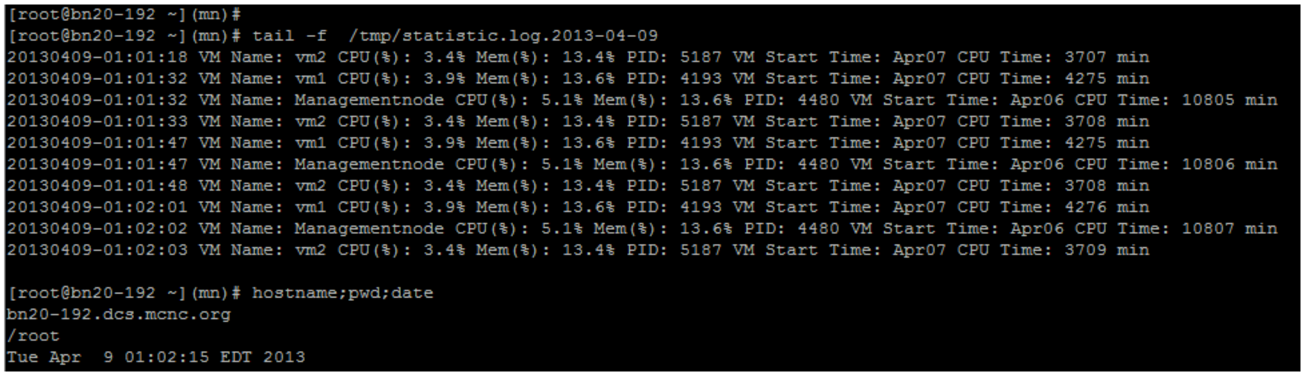
Figure 7 : The database schema of the vmusage table.



```
mysql> show columns from VMmeta;
+-----------------+-----------------------+------+-----+---------+----------------+
| Field           | Type                  | Null | Key | Default | Extra          |
+-----------------+-----------------------+------+-----+---------+----------------+
| id              | smallint(5) unsigned  | NO   | PRI | NULL    | auto_increment |
| stateid         | tinyint(5) unsigned   | NO   |     | 10      |                |
| vmid            | tinyint(5) unsigned   | NO   |     | 10      |                |
| vmname          | varchar(36)           | NO   |     |         |                |
| OS              | varchar(36)           | NO   |     |         |                |
| RAMinfo         | mediumint(8) unsigned | NO   |     | 1024    |                |
| Start           | datetime              | YES  |     | NULL    |                |
| End             | datetime              | YES  |     | NULL    |                |
| PrivIP          | varchar(36)           | NO   |     |         |                |
| PublicIP        | varchar(36)           | NO   |     |         |                |
| eth0            | varchar(36)           | NO   |     |         |                |
| eth1            | varchar(36)           | NO   |     |         |                |
| listening_ports | mediumint(8) unsigned | NO   |     | 1024    |                |
+-----------------+-----------------------+------+-----+---------+----------------+
13 rows in set (0.00 sec)
```

Figure 8 : The database schema for the vmmeta table.

We also came up with three scripts that are :- run.sh which is a central point of control and we ran this script at a frequency of 1 hour,check_top_ps.sh which gathers CPU and memory utilization on each VMHost, load_db.pl which loads the CPU and memory utilization data gathered into the database. Sample results from these scripts can be seen in Figure 9.



```
[root@bn20-192 ~](mn)#
[root@bn20-192 ~](mn)# tail -f  /tmp/statistic.log.2013-04-09
20130409-01:01:18 VM Name: vm2 CPU(%): 3.4% Mem(%): 13.4% PID: 5187 VM Start Time: Apr07 CPU Time: 3707 min
20130409-01:01:32 VM Name: vm1 CPU(%): 3.9% Mem(%): 13.6% PID: 4193 VM Start Time: Apr07 CPU Time: 4275 min
20130409-01:01:32 VM Name: Managementnode CPU(%): 5.1% Mem(%): 13.6% PID: 4480 VM Start Time: Apr06 CPU Time: 10805 min
20130409-01:01:33 VM Name: vm2 CPU(%): 3.4% Mem(%): 13.4% PID: 5187 VM Start Time: Apr07 CPU Time: 3708 min
20130409-01:01:47 VM Name: vm1 CPU(%): 3.9% Mem(%): 13.6% PID: 4193 VM Start Time: Apr07 CPU Time: 4275 min
20130409-01:01:47 VM Name: Managementnode CPU(%): 5.1% Mem(%): 13.6% PID: 4480 VM Start Time: Apr06 CPU Time: 10806 min
20130409-01:01:48 VM Name: vm2 CPU(%): 3.4% Mem(%): 13.4% PID: 5187 VM Start Time: Apr07 CPU Time: 3708 min
20130409-01:02:01 VM Name: vm1 CPU(%): 3.9% Mem(%): 13.6% PID: 4193 VM Start Time: Apr07 CPU Time: 4276 min
20130409-01:02:02 VM Name: Managementnode CPU(%): 5.1% Mem(%): 13.6% PID: 4480 VM Start Time: Apr06 CPU Time: 10807 min
20130409-01:02:03 VM Name: vm2 CPU(%): 3.4% Mem(%): 13.4% PID: 5187 VM Start Time: Apr07 CPU Time: 3709 min

[root@bn20-192 ~](mn)# hostname;pwd;date
bn20-192.dcs.mcnc.org
/root
Tue Apr  9 01:02:15 EDT 2013
```

Figure 9 : The statistics recorded by the monitoring scripts.

The discussion we had with the VCL team members led us to conclude that the data collected using this time interval of 1 hour was less granular and does not capture the real statistics. Additionally, the VMMeta table was not giving any new information because it was capturing the details which were already present in the vcl database, so we decided to drop the table.

2nd iteration

Design

After the first iteration we made changes to database design. The VMMeta table was dropped since the data collected in VMMeta table could be derived from existing tables in vcl database. The second change we made was to increase the frequency of execution of the main script “run.sh” in crontab from 1 hour to 5 minutes.

Implementation

We analyzed the VMMeta table that we proposed in the first iteration and it came out to provide no additional information as the data collected from VMMeta table could also be generated by joining various other tables already present in VCL database. In order to get better and more reliable data we executed the script at various time intervals and analyzed the results. There had to be a tradeoff in terms of granularity of data and overhead in terms of cpu utilization of the script. the results showed that as the frequency of the cron job is increased we get better statistics of CPU and memory utilization but it resulted in more database entries and more CPU utilization of the cron job process and this overhead will increase as the no. of VMHosts managed by a mn node increases. Finding the most appropriate frequency of execution of cron job and projecting it for a larger scale was a challenging task and required revising our original approach.

Results

Based on the results we got by trying different intervals (1 minute, 5 minutes, 10 minutes, 15 seconds, 1 hour) we decided to make the default cron setting as 5 minute which is tunable and balances the script overhead and granularity factor optimally. From the results of these iteration we found the statistics to be accurate for one vmhost but we had to make modifications to make this work on a production systems with multiple management nodes and vmhosts. This required a change in the database design and we decided to add one more column vmhost along with the ip address to uniquely identify a vm on a specific vmhost.

3rd iteration

Design

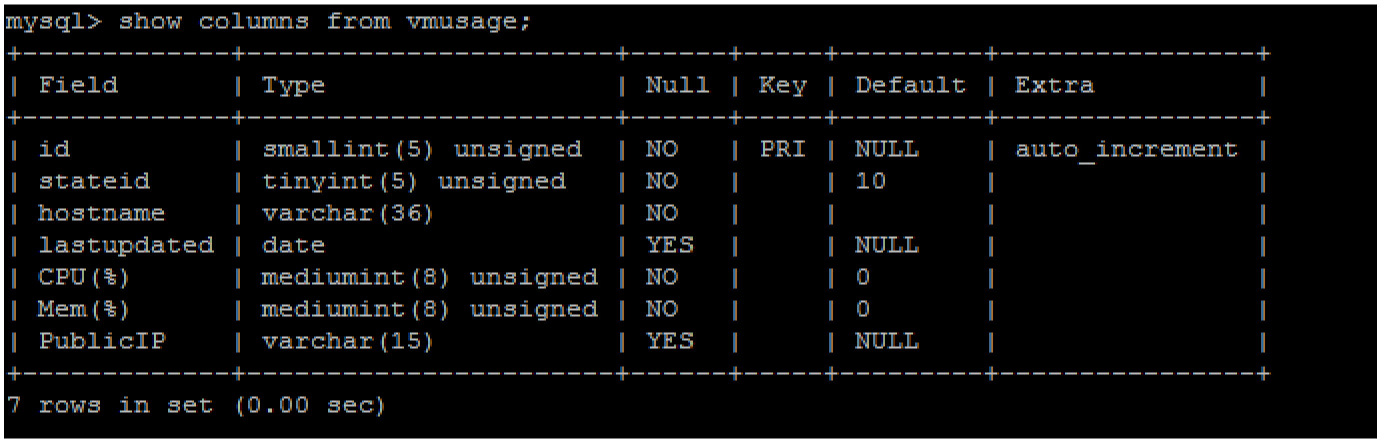
We used the results of the second iteration and made few changes to our design.We altered the existing VMusage table to add another column named “PublicIP” of VMHost.In order to display the CPU and memory utilization of each vm on a vmhost we decided to add a dropdown menu listing the VMhost and its public Ip, which will display the graph with CPU and memory utilization of all the VMS running on that particular VMHost. In each of the graph plot there will be two parallel y axis one denoting the cpu utilization and the other memory utilization. On the x axis the scale will be date wise and multiple plots will be generated one for each vm on a vmhost.

Implementation

Identified the unique key which can uniquely identify the particular VMhost. We understood the flow of graph graph generation using php and dojox charting api[11]. Made changes to statistics.php and statistics.js and mapped them with the new database table to generate a plot. We have identified the changes we need to do for log rotation periodically. The most challenging part of this iteration was understanding the flow of execution of graph generation.

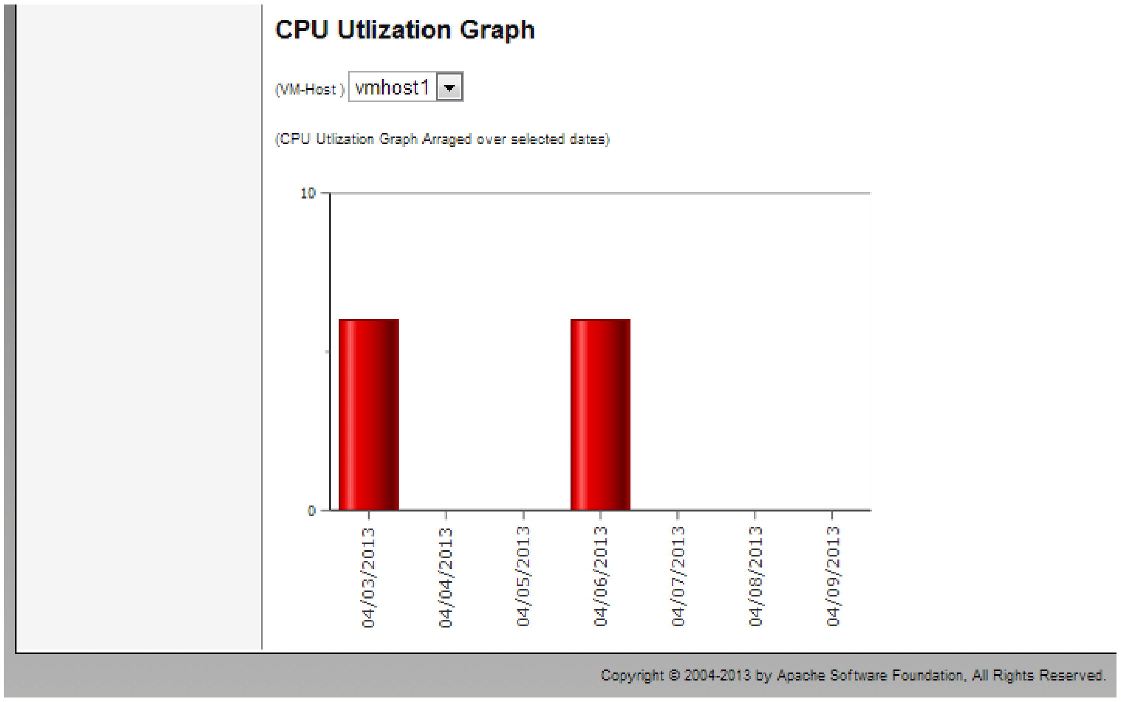
Results

We have implemented a rudimentary structure to display the CPU and Memory utilization of a particular VM on separate graphs for CPU and memory and are going to extend it to display the same for all vms on a VMhost in one single graph. The data for this is stored in the vmusage table depicted in Figure 10. Figures 11 and 12 show the graphs generated for CPU and Memory Load in separate graphs.



```
mysql> show columns from vmusage;
+-------------+-----------------------+------+-----+---------+----------------+
| Field       | Type                  | Null | Key | Default | Extra          |
+-------------+-----------------------+------+-----+---------+----------------+
| id          | smallint(5) unsigned  | NO   | PRI | NULL    | auto_increment |
| stateid     | tinyint(5) unsigned   | NO   |     | 10      |                |
| hostname    | varchar(36)           | NO   |     |         |                |
| lastupdated | date                  | YES  |     | NULL    |                |
| CPU(%)      | mediumint(8) unsigned | NO   |     | 0       |                |
| Mem(%)      | mediumint(8) unsigned | NO   |     | 0       |                |
| PublicIP    | varchar(15)           | YES  |     | NULL    |                |
+-------------+-----------------------+------+-----+---------+----------------+
7 rows in set (0.00 sec)
```

Figure 10 : The database schema for the vmusage table.



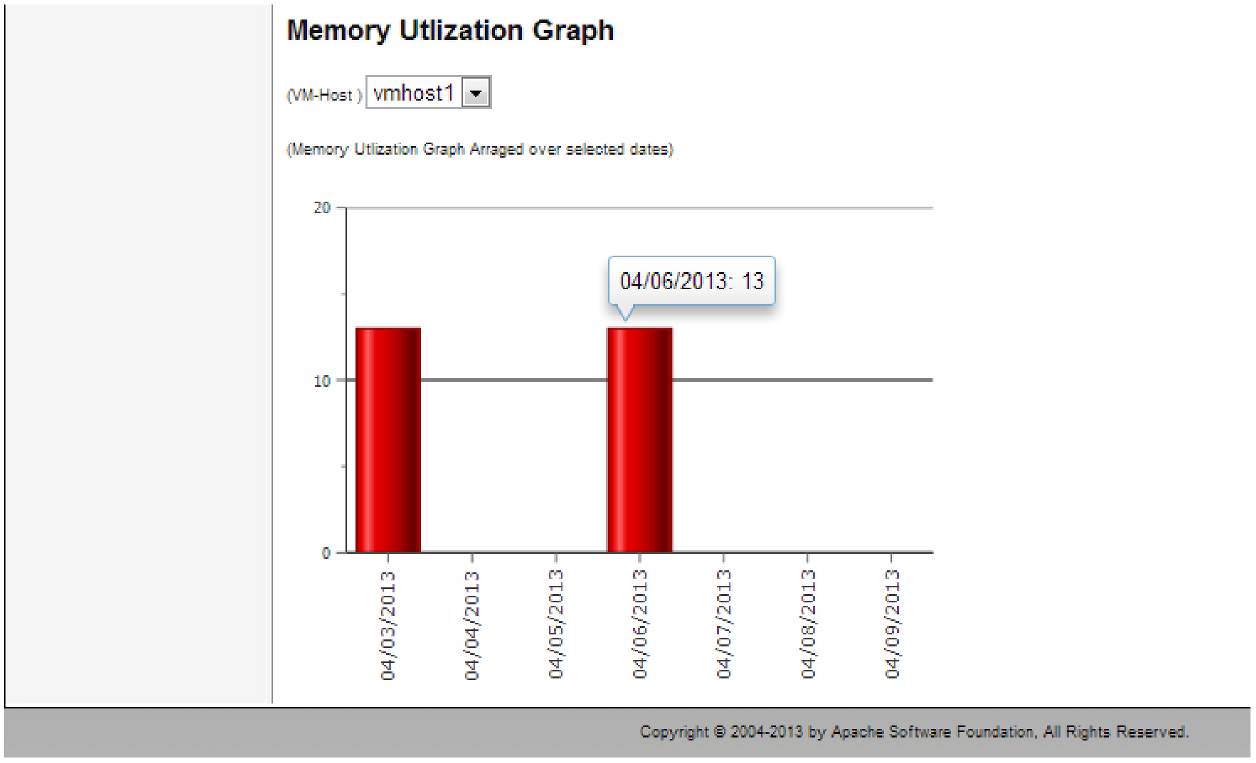
CPU Utlization Graph

(VM-Host) vmhost1

(CPU Utlization Graph Arraged over selected dates)

Copyright © 2004-2013 by Apache Software Foundation, All Rights Reserved.

Figure 11 : The CPU utilization graph for a specific vmhost.



Memory Utlization Graph

(VM-Host) vmhost1

(Memory Utlization Graph Arraged over selected dates)

Copyright © 2004-2013 by Apache Software Foundation, All Rights Reserved.

Figure 12 : The memory utilization graph for a specific vmhost.

4th iteration

Design
We used the results of the third iteration and made few changes to our design where in we decided to use line graph instead of bar graph so that we can display CPU and Memory utilization in same graph. Also after discussion with VCL team members we decided to give user/admin the option to select a particular VMhost and also VM on that VMHost and display CPU and memory utilization of the same rather than showing all VMs of a VMHost on one single graph.

Implementation
We used dojo api to display two drop downs one to select the VMHost and other to select the VM on that VMHost. After selecting the VMHost dynamically we populate the VMs in the other drop down menu. We also completed the coding required to implement the log rotation part where we are archiving the previous logs on daily basis. Changing the VMs in the drop down dynamically on changing the VMHost and displaying the graph dynamically based on the requirements of the admin was a challenging task in this iteration. We implemented the script for rotating logs present on management node on a daily basis using shell scripting[12].

Results
We are able to present the admin with a single graph to display the CPU and Memory utilization of a particular VM selected by admin on a VMHost. We are also able to rotate the logs present on the management node on a daily basis.

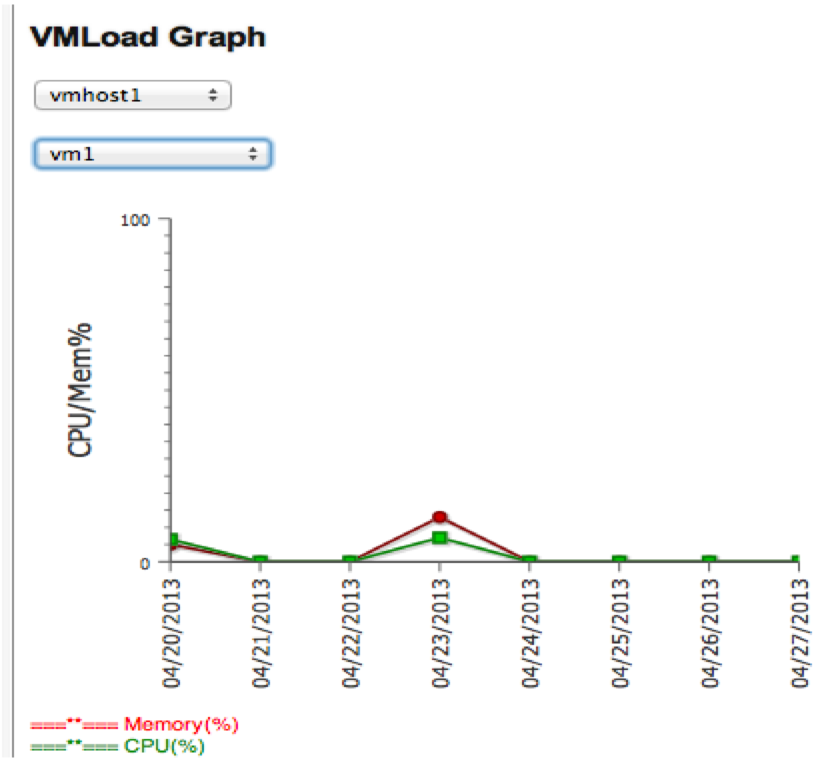


Figure 13 : A graph depicting the load on VM1 of vmhost1.

Summary:

At the end of the fourth iteration we have implemented all the requirements and we are able to present the admin with the appropriate VM Load Statistics in graphical format as in Figure 13. We are able to store the CPU and Memory Statistics of every VM in database. We are able to rotate the logs on a daily basis.

5.6 Billing

problem description:

While VCL stores information about reservations, their lengths, and the users that made them in the log table, this information is not presented in a manner that allows administrators and users to simply and concisely view the amount of time they've spent on specific reservations. The log table is generally only consulted for forensics, troubleshooting, or otherwise determining the state of the system. Tracking per user usage of images permits metered billing of users (an essential aspect of a cloud system according to the NIST definition) and also permits analysis of image usage that could be used for optimization.

1st iteration

Design

The initial design of the billing module involved billing to user groups. The idea was that a user has access to a particular image through a user group, and the user picks which user group the particular reservation is meant to be billed to. This involved tracking the membership of the user groups and presenting all the user groups that grant access to the image that the user is about to reserve. This design also utilized the request and reservation tables to derive values for the length of the reservation and which images were used. This was modified in the next iteration.

Implementation

As above, we initially relied on the request and reservation tables. The request table was used to determine which user made a request for a reservation and how long he wants it for. The reservation table was used to determine which image the user requested for. We tested the method by making reservations in the system and testing to see if these tables were populated appropriately throughout the life of the reservation and long enough after to derive the information for the billing tables.

Results

While the information in the tables was sufficient for the calculations required for the billing table, the tables only store the information for a short while, deleting entries once the reservations are no longer relevant to vcld. These tables were therefore not a reliable source of information for the billing table. In addition, it was determined that having to choose which group you wish to bill to was an unreasonably complicated process and that the interface presented to the user should be disturbed as little as possible from its current state.

2nd iteration

Design

After meeting with the VCL engineers, this design was revised to admit more granular information about users. The tracking was converted from measuring per user-group to measuring per user. All users are tracked individually in the billing table, and any reservations they make are directly associated with them. Billing is no longer associated with user groups, and therefore determining which user group grants access to an image is no longer required. In this iteration, the billing tables used information derived from the log tables.

Implementation

Since the information in the request and reservation tables is replicated in the log tables, we expected that we'd be able to retrieve the information from there instead. Again, test reservations were made and queries were written to see if the tables are appropriately populated as we wished them to be.

Results

This was successful and it was possible to calculate usage using just the log table along with the tables it referred to as containing foreign keys. However, the log table is not referenced anywhere in the code to read from. Instead it seems to be intended only to store information for forensics and troubleshooting and not to be used to derive additional information. In addition, the new design of tracking usage per user alone required the administrators to set the rate per user per image, which is very labour intensive.

3rd iteration

Design

The VCL system may be setup to permit different rates to different users depending on the user group they are associated with. The billing is still done per-user and not per-user-group, but the rates that a user is charged are partially inherited from the groups that they are member of. This was based on the understanding that users are likely to use the cheapest option available to them. Therefore, the rate charged to a user is the lowest amongst the rate available to each of the user groups that the user is associated with and the rate provided to the user. The reservation that the user makes is then charged to him at this rate. The method used to calculate lengths of reservations were also changed. Information is now stored redundantly in the billing table alongside the log table (which is no longer referred to). In this way, the log table is not reused for purposes it is not intended to be used for.

Implementation

Since we had started work on the codebase at this point of time, this required changes to the program. We then modified the section of the programs that inserted entries into the log table and ensured that certain information is duplicated into the billing tables as required. In addition, the rate chosen for each reservation was chosen to be the minimum rate available to the user either directly or through membership in a user group.

Results

Since the information required is replicated in the billing tables, any queries against it were successful. The new method of determining the rate for a user also ensured that the reservation front-end was not made more complicated and it worked under the assumption that the user would want to use the cheapest option available to him.

4th iteration

Design

By this time, the implementation was mostly decided upon, so changes to the design involved the interface presented to users and administrators that displays the usage. At first we'd intended to display this as part of the statistics page, but by this iteration it was decided that a separate page is a better idea.

Implementation

We created mockups on paper and wrote down user workflow diagrams. The initial mockup added the information to the statistics page, and after a few more tries, we settled on adding a new page that would display billing-related information.

Results

These revealed the shortcomings of including the information in the statistics page. It becomes crowded and for administrators it tends to push out other, perhaps more important, information. The billing page on the other hand was a better idea since it could also be used to set rate information as well as view usage.

5th iteration

Design

With the back-end design and the calculation of data done, the remaining work focussed on implementing modifications to the user-visible portion of the project to enable users and administrators to see charges incurred. In addition, administrators should be able to set the rate for usergroup-image combinations or user-image combinations.

Implementation

Figure 14 depicts the screen where an administrator would pick what to do. Figure 15 is what he would get if he were to choose to Set Rates. Figure 16 is the view of a user's bill from the admin's display.

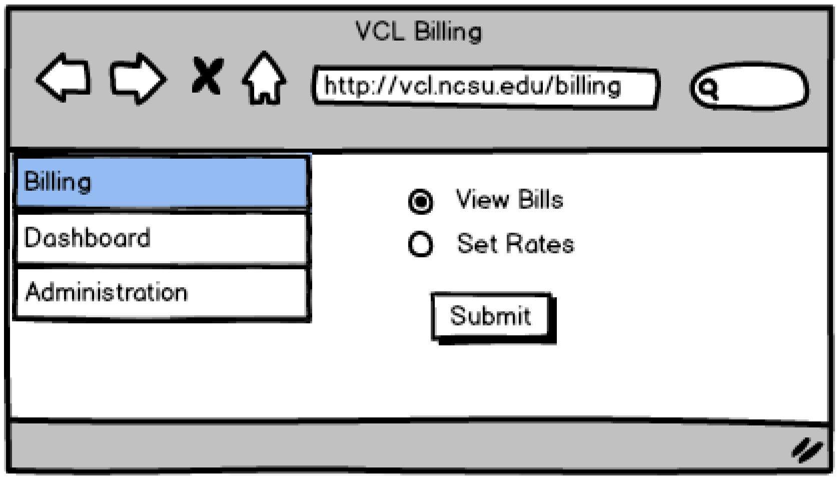
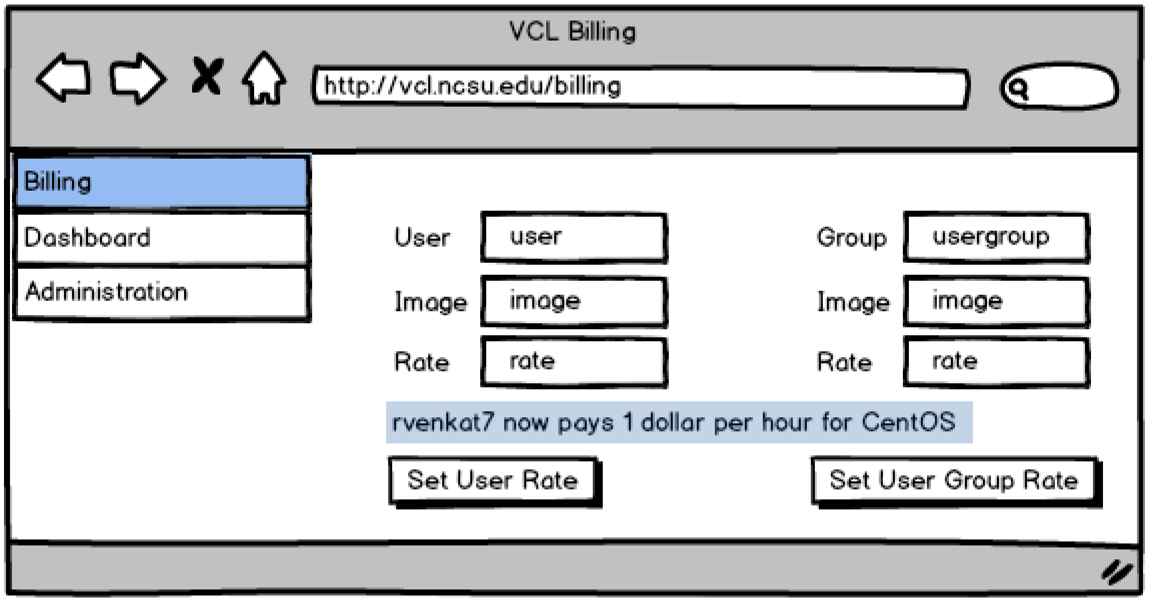
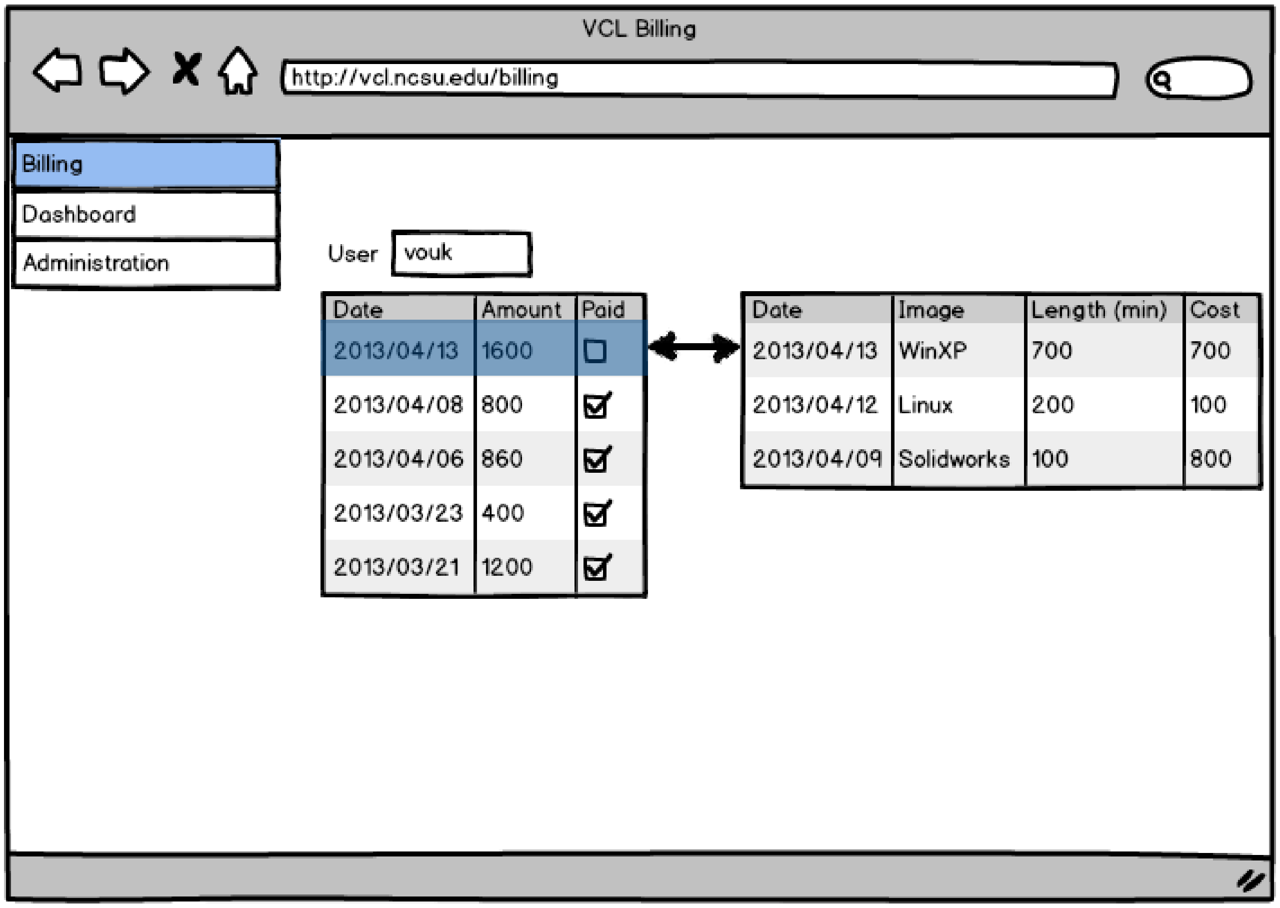


Figure 14 : A mockup of the initial billing screen as an admin would see it.



rvenkat7 now pays 1 dollar per hour for CentOS

Figure 15 : A mockup of the Set Rate screen where an admin can specify a rate for a combination of user and image or usergroup and image.



Date	Amount	Paid
2013/04/13	1600	☐
2013/04/08	800	✓
2013/04/06	860	✓
2013/03/23	400	✓
2013/03/21	1200	✓

Date	Image	Length (min)	Cost
2013/04/13	WinXP	700	700
2013/04/12	Linux	200	100
2013/04/09	Solidworks	100	800

Figure 16 : A mockup of the View Bill screen, where an admin can view the bill for any user.

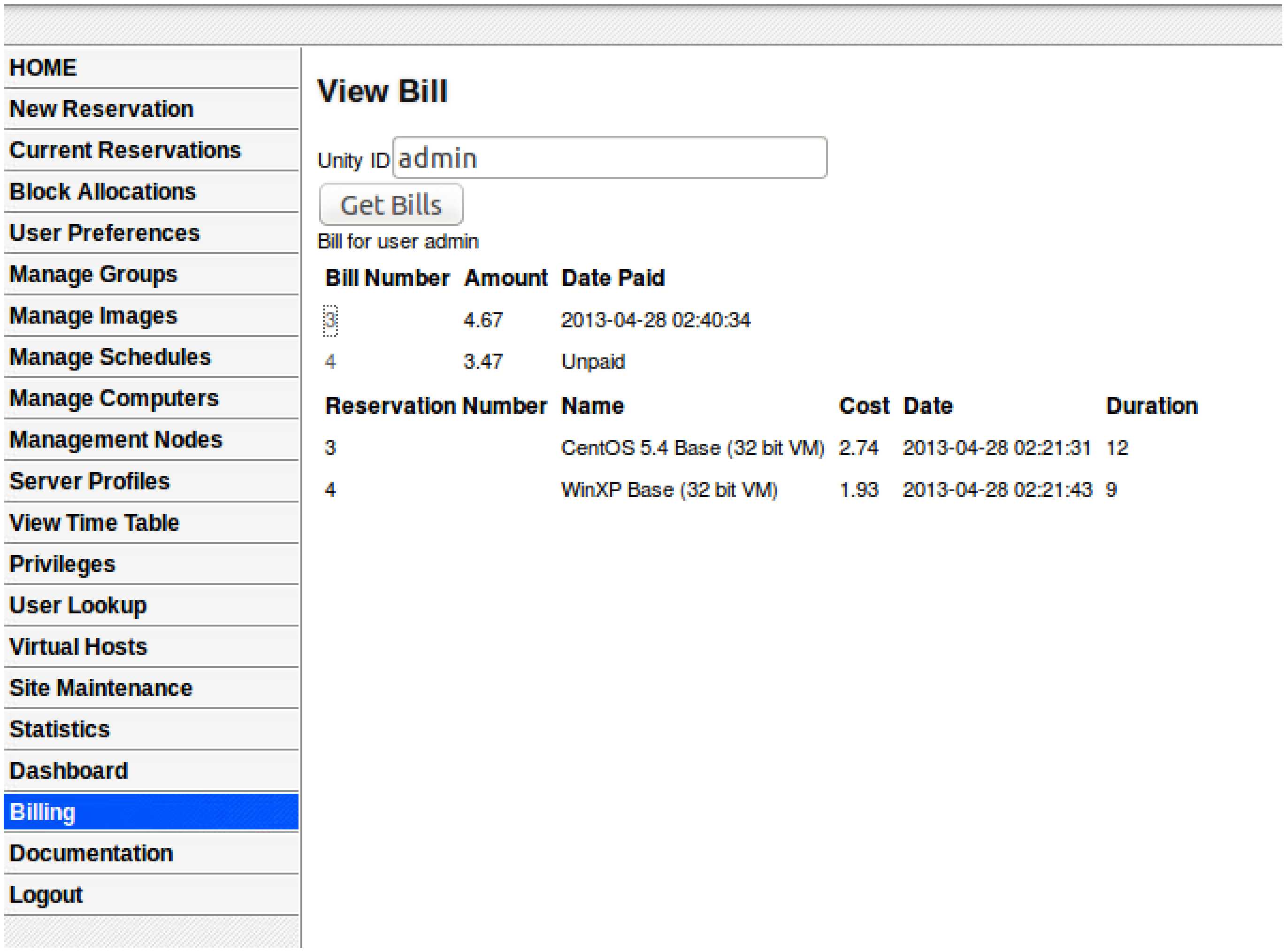
6th iteration

Design

Everything was mostly set in stone at this point, and only the interface remained to be implemented based off the mockups. Minor cosmetic modifications involved splitting the usergroup and user rate settings screens. In addition, we enabled wildcard entry of image rates to set the default image rates meant for when no particular distribution is intended for different usergroups.

Implementation

The UI was implemented by adding a new page to the navigation bar. Users with the right to access the dashboard can also access detailed billing information and set rates, but normal users can only view their own bills. Figure 17 shows the billing information for the user `admin`. Clicking on the Bill Number will display the itemized listing of costs below. Figure 18 shows the screen to set the rate for a usergroup. Placing a ‘*’ in the usergroup field or leaving it blank will set the rate for all usergroups for that image. Otherwise, the name of the usergroup should be used.



View Bill

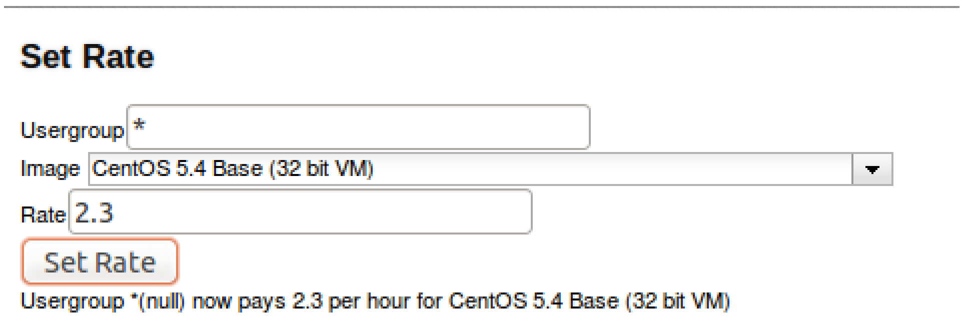
Unity ID admin

Bill for user admin

Bill Number	Amount	Date Paid
3	4.67	2013-04-28 02:40:34
4	3.47	Unpaid

Reservation Number	Name	Cost	Date	Duration
3	CentOS 5.4 Base (32 bit VM)	2.74	2013-04-28 02:21:31	12
4	WinXP Base (32 bit VM)	1.93	2013-04-28 02:21:43	9

Figure 17 : A view of the bill's incurred by the admin user and a line item listing for the first bill.



Set Rate

Usergroup *

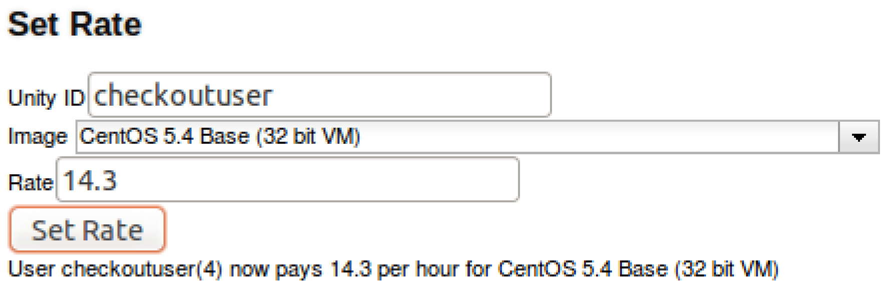
Image CentOS 5.4 Base (32 bit VM)

Rate 2.3

Usergroup *(null) now pays 2.3 per hour for CentOS 5.4 Base (32 bit VM)

Figure 18 : Setting the default rate for a particular image

Figure 19 shows the screen where the rate for a user can be specified per image. Unlike the case where usergroups have their rates set, the wildcard is not permitted here. Figure 20 shows the billing view for an unprivileged user. The unprivileged user is only shown their own billing information and no more. Figure 21 shows the view where an administrator can see how much each image has been used and by whom.



Set Rate

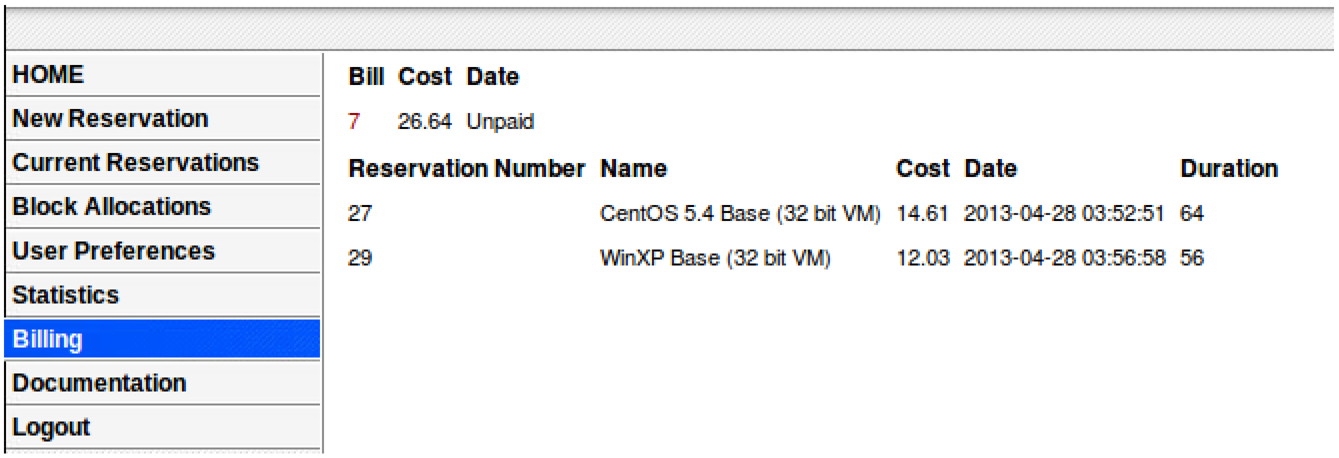
Unity ID checkoutuser

Image CentOS 5.4 Base (32 bit VM)

Rate 14.3

User checkoutuser(4) now pays 14.3 per hour for CentOS 5.4 Base (32 bit VM)

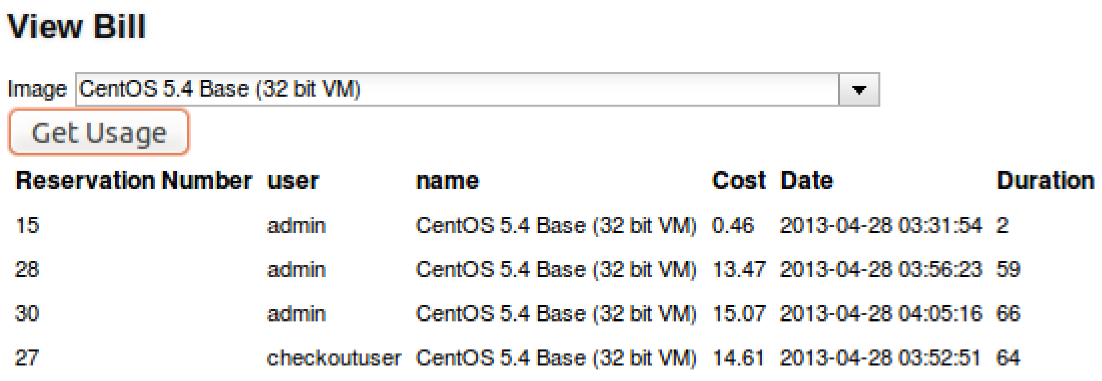
Figure 19 : Setting the rate for a user for an image.



Bill	Cost	Date
7	26.64	Unpaid

Reservation Number	Name	Cost	Date	Duration
27	CentOS 5.4 Base (32 bit VM)	14.61	2013-04-28 03:52:51	64
29	WinXP Base (32 bit VM)	12.03	2013-04-28 03:56:58	56

Figure 20 : The view for an unprivileged user.



View Bill

Image CentOS 5.4 Base (32 bit VM)

Reservation Number	user	name	Cost	Date	Duration
15	admin	CentOS 5.4 Base (32 bit VM)	0.46	2013-04-28 03:31:54	2
28	admin	CentOS 5.4 Base (32 bit VM)	13.47	2013-04-28 03:56:23	59
30	admin	CentOS 5.4 Base (32 bit VM)	15.07	2013-04-28 04:05:16	66
27	checkoutuser	CentOS 5.4 Base (32 bit VM)	14.61	2013-04-28 03:52:51	64

Figure 21 : The usage recorded on a single image.

Summary

In summary, the billing logic, usage tracking, and user interfaces are complete. Figure 14 depicts an intermediate screen that depicts whether the admin would like to view bills or set rates. Figure 15 depicts the screen where the admin can set rates, fulfilling II 3.1 and II 3.2. Figure 16 depicts the screen where an admin can view the bill for any user, fulfilling OI 4.1. A similar screen without the the choice of user will be displayed to a normal user, thereby fulfilling OI 3. Figure 21 is a screenshot of the interface to view usage information for a specific image. This is only available to admin users and therefore fulfills OI 4.2.

Final Database Schema

`Rate (id, userid, usergrpid, imageid, rate):`

Column	**Description**
id	The ID of the charge entry
userid	Reference to the user.id
usergrpid	Reference to usergroup.id
imageid	Reference to image.id
rate	The cost per unit time in USD for the specific image for the specific user or for a specific usergroup

Table 7 : The database schema for the Rate table.

The Rate table maps a pair of users and images to a particular rate in USD per unit time. It is populated through the Input Interface described by at least one of II 3.1 or II 3.2. Neither II 3.1 nor II 3.2 may permit the administrative staff to enter an id. Rate.id will be auto-incremented. The other fields may be manually specified.

When a user is permitted to reserve an image that he was previously unable to access (via the privilege tree, say), an entry is created in the Rate table with rate 1 USD per unit time. The administrator is then advised to set the rate.

Editing the rate for a specific combination of user and image is achieved by specifying the combination, not by addressing by Rate.id.

For each row, only either userid or usergrpid can be set. The other must be NULL.

The unit of time is defined as minutes. Any calculation of duration is rounded up to the next unit. The schema for this table is described in Table 7.

`Bill (id, userid, status, timepaid):`

Column	**Description**
id	The ID of the bill
userid	Reference to user.id
status	Can be ‘paid’ or ‘pending’

timepaid	Stores the date that the bill was paid

Table 8 : The database schema for the Bill table.

The schema for the bills and line items associated with them is depicted in Table 8 and Table 9. A single bill line item corresponds to a charge for use of a single resource at a specific time. The bill.id is the bill that the bill line item is on. A user would like to pay for a bill consisting of multiple reservations at the same time. Associating each reservation with a line item and then a group of line items with a bill will permit the user to do this by simply paying the bill. The following policy is followed to associate a new line item with bills:

1 The last bill (determined by highest id) on this user's account is checked by examining the status field. If it is 'paid', a new bill is created as described in S3 and chosen to be the active bill. If it is 'pending', it is chosen as the active bill.
2 A new line item is created corresponding to a reservation that the user has made. The elements of the line item are as follows:
   - id is auto-incremented over the previous id
   - billid is set to the id of the active bill
   - cost is set to 0
   - loaded is NULL
   - finalend is set to same as log.initial end
   - imageid is set to the corresponding id of the image loaded
3 When a reservation has ended for whatever reason, the line item is modified as follows:
   - length is calculated to be the time difference between BillLineItem.finalend and BillLineItem.loaded in the time unit
   - cost is set to the product of length and rate for that combination of user and image

`BillLineItem (id, billid, length, logid, cost):`

Column	**Description**
id	The ID of the line item
billid	Reference to bill.id
cost	The total cost calculated for the specific image at the time of reservation
loaded	The time at which the reservation was loaded
finalend	The time at which the reservation ended

imageid	Reference to image.id

Table 9 : The database schema for the BillLineItem table.

A bill is a holder for a group of Bill Line Items. The idea is to allow a user to pay for a group of reservations at the same time, and perhaps view these reservations as a group later. A new bill is created as follows:

- id is auto-incremented over the previous id
- userid is set to the id of the user with whom this bill is associated
- status is set to ‘pending’
- timepaid is set to the beginning of the epoch

When a bill is paid, the following changes are made:

- status is set to ‘paid’
- timepaid is set to the current time

The total cost of a bill is the sum of the costs of the entries in BillLineItem that correspond to it. A bill can only be paid in full. A bill can only be paid if its status field is set to ‘pending’ and if there are no open reservations still on that bill. If these conditions are both met, it can be paid at any time.

6 Verification and Validation

We have worked closely with the VCL engineers to validate our design and implementation throughout project development. This has been an iterative process which has allowed us to refine design elements that are correct, and discard or change those elements which are incorrect. We have completed some verification of design elements during development. We plan to implement more robust testing during the final phases of development.

6.1 Validation

The validation of this project has been carried out in multiple phases. In the initial phase of the project, we tried to understand the requirements of the project appropriately and then come up with some design questions which were discussed with the team as a whole and then communicated to VCL Engineers for validation. As the project progressed, each of the sub module teams had regular meetings to validate the work they have done. As a whole the validation happened through

- Meetings
  Each time when an individual comes up with a specific design it is validated with entire team through regular meetings and then it is communicated to VCL team engineers for final validation.

- Emails
  Once the team has implemented the subtasks, the results and other follow up questions were communicated via email to VCL engineers.

- Wireframes and mockups
  Teams have come up with wireframes of the submodules and they are communicated to VCL team.

- DB schema
  The database schema for VM Load, Billing, Weather Map have been designed by the sub modular teams and got them validated from the rest of the teams and finally by the VCL team.

We will continue the same approach of what we have been doing over the course of the project by getting it validated with the team members and finally through the VCL Engineers.

6.2 Verification

Verification has been accomplished mainly through the use of manual testing in isolated instances of VCL sandbox. We have focused primarily on producing a valid design, and developing working software. Initial testing is done on isolated instances of VCL sandbox, and then tested on our larger multiple MN environment.

6.2.1 Test Plan

Our test plan consisted of running test scenarios that are intended to mimic real-world use cases. In addition to these, we intend to run unit tests for each of the functions that we are using.

Top n Images

Test Case 1.1: Test Listing of Top Images From Past Day

Procedure:

- Create 3 reservations of Centos 5.4 as user `admin`.
- Create a reservation of Centos 5.3 as user `admin`.
- Create a reservation of Windows XP as user `admin`.
- Click ‘Statistics’ on the sidebar.
- Set the number of images to 2

Pass Criteria:

- Only two of the images are listed in the Top Images in The Past Day section.
- The first image is Centos 5.4.
- The second image is Centos 5.3.

VM Load

Test Case 2.1 : Testing CPU/Mem as load parameters for analyzing VMLoad.

Procedure :

- Make two reservations (vm1,vm2) on sandbox through UI.
- Go to management node to view the statistic log to get cpu and memory utilization of virtual machines on the vmhost.
- Increase the cpu utilization of one of the VM's using a cpu hogging process running on one of the vm's.
- Check memory and CPU utilization of the vmhost.
- Increase the memory utilization of one of the VM's using a memory hogging process running on one of the vm's.
- Login to individual vm's and check the utilization inside vm's approximates the utilization of vm's on the vmhost and the utilization of vmhost increases proportionally with increase in load on vm's.

Pass Criteria:

- Check the CPU utilization of vm's running on the vmhost.
- Check Memory utilization of vm's running on the vmhost.
- From observed data in log:
  VM1's load metrics when idle: CPU: 8.6% RAM: 0.4%
  VM1's load metrics after increasing load: CPU 60.6%, RAM: 3.7% (with CPU/Mem hogging process)

Test Case 2.2: Test the granularity of time interval ideal for collecting vm statistics.

Procedure :

- Create the database table in a new provenance database vmusage.
- Make two reservations (vm1,vm2) on sandbox through UI.
- Set the main script(run.sh) in crontab with varying frequency of execution(15 seconds, 1 minute, 5 minute, 10 minute).
- Run each frequency for 6 hours and record the delay measurements in to the statistic log.
- Go to management node to view the statistic log to get cpu and memory utilization of VM's.

Pass Criteria:

- Check the updates in the database table and the number of entries recorded.This is to ensure that entries are not corrupted or missing.
- Observed that with lesser timer-intervals the time-delay is more. Below are the results observed with time-interval .

Frequency	**Delay (in sec)**

15 sec	2.52
1 min	2.46
5 min	2.43
10 mins	2.426

Table 10 : Identifying granularity of Polling

Test Case 2.3: Test to check if UI reflects the cpu and memory utilization in graphical format.

Procedure:

- Login to VCL using admin credentials.
- Check if the database has all the relevant entries related to the date range for which graph generation is needed.
- Go to statistics tab and select the time interval.
- Select the vmhost from drop down menu for which cpu/mem statistics is needed.
- View and confirm if the graph reflects the values present in the database.

Pass Criteria:

- Selected date range should be present on the x axis and y axis should have the values of the cpu and memory of the corresponding dates of the vm.
- Query the database to get the average cpu and memory utilization for a specific vm grouped date wise and match them with the results obtained from UI.

Test Case 2.4: Integration testing with multiple vmhosts and vms.

Procedure:

- Login to VCL using admin credentials.
- Check if the database has multiple vmhosts and all the relevant entries related to the date range for which graph generation is needed.
- Go to statistics tab and select the time interval.
- Select the vmhost and vm from the drop down menu for which cpu/mem statistics is needed.
- View and confirm if the line curves reflect the CPU and memory utilization values present in the database.

Pass Criteria:

- Selected date range should be present on the x axis and y axis should have the values of the cpu and memory of the corresponding dates of the vm.
- Query the database to get the average cpu and memory utilization for a specific vm grouped date wise and match them with the results obtained from UI.

Test Case 2.5: Test to check if selection of the vm on UI depicts the VM load graph correspondingly for that vm.

Procedure:

- Login to VCL using admin credentials.

- Check if the database has all the relevant entries related to the date range for which graph generation is needed.
- Go to statistics tab and select the time interval.
- Select the vmhost and a “vm” from drop down menu for which cpu/mem statistics is needed.
- View and confirm if the graph reflects the values present in the database.
- Now select a different vm and observe the change in graph.

Pass Criteria:

- Change in the selection of “vm” should reflect the VM load graph of that “vm” .
- Selected date range should be present on the x axis and y axis should have the values of the cpu and memory of the corresponding vm.

MN Load

Test Case 3.1 : Testing that reservations show up in MN Load Heatmap/Weathermap

Procedure:

- Make reservations for various Images on the testbed
- Ensure that reservations are distributed over more than 1 management node
- Keep a count of the total number of reservations made
- Look up the number of reservations on each management node using the dashboard and database.
- Go to dashboard and generate Heatmap for the period through when reservations were made

Pass Criteria:

- The total number of reservations over all management nodes should be equal to the count of reservations we made
- The number of reservations at each management node should match the data found previously.

Test Case 3.2: Test Coloring for Heatmap works when only 1 management node is present

Procedure:

- Make reservations such that they map to only management node
- Go to dashboard and generate heatmap for the period in which the reservations were made.

Pass Criteria:

- The heatmap should show reservations only one management node
- The color for that management node should be red

Test Case 3.3: Test coloring works when there is more than 1 management node

Procedure:
- Make reservations which distribute over multiple management nodes
- Generate the heatmap using the dashboard

Pass Criteria:
- The heatmap should show all management nodes which had a reservation.
- The color for management node with maximum number of reservation(call this max_reservarions) should be red
- The color for other management nodes should be be a shade between green and yellow if less than 50% of max_reservations
- The color for other management nodes should be be a shade between yellow and red if more than 50% of max_reservations

Test Case 3.4: Test GeoIP maps location correctly

Procedure:
- Make reservations from different locations (using proxies/RDP etc)
- Generate Weathermap

Pass Criteria:
- Map correctly shows location of the remote user

Test Case 3.5: Weathermap is able to depict levels of detail correctly

Procedure:
- Make reservations from within NCSU network.
- Make a couple of reservations from one other location.

Pass Criteria:
- Reservations from the same network don’t clutter up the weathermap in an un-viewable fashion

Billing Module:

Test Case 4.1: Testing of estimated cost

Procedure:
- Clear all entries in the rate table
- Add a new entry for image Centos 5.4 for user admin in the rate table with rate 0.7
- Login through the web interface as user admin
- Click on the Dashboard link on the sidebar
- Choose the Centos 5.4 image and choose length of reservation as 2 hours

Pass Criteria:
- An estimated cost of the reservation is displayed
- The cost estimate is $1.4

Test Case 4.2: Storage of billed usage

Procedure:

- Clear all entries in the Rate table
- Clear all entries in the Bill and BillLineItem table
- Add a new entry for image Centos 5.4 for user admin in the rate table with rate 0.7
- Login through the web interface as user admin
- Click on the Dashboard link on the sidebar
- Choose the Centos 5.4 image and choose length of reservation as 2 hours
- Create the reservation
- End the reservation after 6 minutes

Pass Criteria:

- The cost of this reservation is added to the current bill of user `admin`.
- User `admin` must have 0.07 dollars of unpaid bills.

Test Case 4.3: Storage of multiple Bill Line Items to the same bill

Procedure:

- Clear all entries in the Rate table
- Clear all entries in the Bill and BillLineItem table
- Add a new entry for image Centos 5.4 for user admin in the rate table with rate 0.5
- Add a new entry for image Windows XP for user admin in the rate table with rate 2
- Login through the web interface as user admin
- Click on the Dashboard link on the sidebar
- Choose the Centos 5.4 image and choose length of reservation as 2 hours
- Create the reservation
- Choose the Windows XP image and choose length of reservation as 1 hour
- End both reservations after 6 minutes

Pass Criteria:

- The single bill for user `admin` should show 0.25 dollars of unpaid bills.
- The BillLineItem table should be populated with two individual line items:
    - Windows XP: 10 minutes
    - Centos 5.4: 10 minutes

Test Case 4.4: Viewing of bills

Procedure:

- Repeat Test Case 4.3.
- Login as user `admin` through the web interface.
- Click the link ‘Billing’ on the sidebar.
- Select ‘View Bills’ and Click Submit.
- Click the first bill, if present.

Pass Criteria:

- A single bill should show as unpaid and costing 0.25 dollars.
- On clicking the bill, the line items should display.

Test Case 4.5: Entering Rate Information For a Specific User

Procedure:

- Clear the Rate table.
- Login through the web interface as user `admin`.
- Select Billing on the sidebar.
- Select ‘Set Rates’ and click Submit.
- Type in the name of the user `admin` in the field ‘user’
- Type the letter ‘C’ and select the option ‘Centos 5.4’ when it appears in the field ‘Image’.
- Type the rate .4 in the field ‘rate’.
- Click Set User Rate

Pass Criteria:

- A text box saying “admin now pays .4 dollar per minute for Centos 5.4” appears above the ‘Set User Rate’ button.
- The table ‘Rate’ now contains an entry (Centos 5.4, admin, NULL, 0.5).

Test Case 4.6: Entering Rate Information For a Usergroup

Procedure:

- Clear the Rate table.
- Login through the web interface as user `admin`.
- Select Billing on the sidebar.
- Select ‘Set Rates’ and click Submit.
- Type in the name of the usergroup `adminUsers` in the field ‘usergroup’
- Type the letter ‘C’ and select the option ‘Centos 5.4’ when it appears in the field ‘Image’.
- Type the rate .5 in the field ‘rate’.
- Click Set Usergroup Rate

Pass Criteria:

- A text box saying “admin now pays .5 dollar per minute for Centos 5.4” appears above the ‘Set Usergroup Rate’ button.
- The table ‘Rate’ now contains an entry (Centos 5.4, NULL, admin, 0.5).

Test Case 4.7: Entering Wildcard Rate Information for an Image

Procedure:

- Clear the Rate table.
- Login through the web interface as user `admin`.
- Select Billing on the sidebar.
- Select ‘Set Rates’ and ‘usergroup’ and click Submit.
- Set the usergroup text field to ‘*’
- Select the Windows XP image.
- Type in 1.4 in the rate field.
- Click Set Rate.

Pass Criteria:

- A textbox saying “Usergroup *(null) now pays 1.4 dollars per minute for Windows XP” appears below the ‘Set Rate’ button.

- The table ‘Rate’ now contains an entry (Windows XP, NULL, NULL, 1.4).

6.3 Metrics for Success

Some of the metrics we use to determine our success are as follows

- $\text{Requirement Coverage} = (\text{Requirements Met}) \div (\text{Total No. of Requirements})$
- $\text{Test Case Pass rate} = (\text{Tests Passed}) \div (\text{Total Tests})$
- $\text{Code Base Test Coverage} = (\text{Total no. of Sections Tested}) \div (\text{Total no. of Sections})$

Since we relied on agile development methods, there were many iterations of our design. The results of each of these iterations were run by the VCL engineers to determine their suitability in meeting their requirements and appropriate changes were scheduled for the next iteration, if necessary. Since we had direct access to the customers of the project, development was boosted by the ability to have a back-and-forth discussion regarding design and implementation details.

6.4 Test Cases Verification

Test Number	Test Case	Result	Comment
1.1	Test Listing of Top Images From Past Day	Pass	
2.1	Testing CPU/Mem as load parameters for analyzing VMLoad.	Pass	
2.2	Test the granularity of time interval ideal for collecting vm statistics.	Pass	
2.3	Test to check if UI reflects the cpu and memory utilization in graphical format	Pass	
2.4	Integration testing with multiple vmhosts and vms.	Pass	
2.5	Test to check if selection of the vm on UI depicts the VM load graph correspondingly for that vm.	Pass	
3.1	Testing that reservations show up in MN Load Heatmap/Weathermap	Pass	Reservations were emulated, as this module works with an old “dump” database

3.2	Test Coloring for Heatmap works when only 1 management node is present	Pass	
3.3	Test coloring works when there is more than 1 management node	Pass	
3.4	Test GeoIP maps location correctly	Pass	
3.5	Weathermap is able to depict levels of detail correctly	Pass	
4.1	Testing of estimated cost	Pass	
4.2	Storage of billed usage	Pass	
4.3	Storage of multiple Bill Line Items to the same bill	Pass	
4.4	Viewing of bills	Pass	
4.5	Entering Rate Information For a Specific User	Pass	These functions depend upon sanitising code present in VCL. That code improperly converts anything starting with a 0 to 0. Therefore, 0.5 has to be entered as .5.
4.6	Entering Rate Information For a Usergroup	Pass	As above
4.7	Entering Wildcard Rate Information for an Image	Pass	As above

Table 11 : Test Case Verification

7 Schedule and Personnel

We started of on our project by meeting with the VCL engineers once a week to understand the requirements of our project. Each meeting was followed by team meetings to discuss the requirements and understand how we would go about implementing what was required.

We started of with a simple module to update the dashboard to show top N images as opposed to just to top 5. The idea behind this module was to give us some knowledge on the structure of VCL and how exactly everything worked. We were able to set up a working version of this module in the first week of the project development.

Another part of the starting phase was to get a proper working environment. After discussion with the VCL team it was decided to tie up a couple of sandboxes for this purpose. So 3 long term reservations were made, of which one was based on KVM.

The rest of the meetings with the VCL engineers and Mr.Streck led us to break down what we were working on into 3 sub-teams.

7.1 General Work

Task	Start Date	End Date
Modify VCL Web interface to work for Top “N” images	Mar 11	Mar 12
Set up our testbed which has 3 VCL sandbox instances tied up together	Mar 19	Mar 22

Table 12 : The schedule for common tasks that involved the entire team.

7.2 VM Statistics Team (Shayan Sinha, Sashank Reddy, Amit Mendiratta)

Schedule

Task	Personnel	Start Date	End Date
Meet VCL Engineers to discuss requirements for VMLoad project	Sashank	Mar 4	Mar 4
Understand what we have on the requirements through VCL documentation	Sashank,Amit and Shayan	Mar 6	Mar 8
Identification of key requirements and come up with a design	Sashank,Amit and Shayan	Mar 9	Mar11
Identify the metrics and the methodology to capture the load statistics	Sashank,Amit and Shayan	Mar 10	Mar 13
Started coding for collecting the CPU statistics	Shayan	Mar 12	Mar14
Discussion with VCL Engineers on the basic implementation	Sashank,Amit and Shayan	Mar 14	Mar 14

Implemented a shell script,batch script for gathering info to mn	Sashank,Amit and Shayan	Mar 15	Mar 15
Implementation of new DB module and tables	Shayan & Amit	Mar 17	Mar 19
Integration work and testing of the basic version	Sashank,Amit and Shayan	Mar 17	Mar 20
VM statistics update to DB	Amit & Shayan	Mar 18	Mar 22
Granularity identification of time-interval through sampling	Sashank,Amit and Shayan	Mar 24	Mar 29
Discussion on identification of DB queries and design of UI	Sashank,Amit and Shayan	Mar 25	Mar 28
Understanding PHP to implement graphs on the UI	Shayan & Amit	Mar 30	April 5
Understand the log rolling feature	Sashank	April 3	April 7
In depth study of PHP and Dojo charting	Sashank, Shayan & Amit	April 8	April 11
Log rotation	Sashank & Amit	April 9	April 1
Implementation of line curves for CPU and Memory separately	Sashank, Shayan & Amit	April 12	April 15
Work on merging of CPU/Memory into a single graph	Shayan & Amit	April 16	April 17
Final Presentation slides of VM Load	Sashank	April 17	April 17
Code optimisation for SQL queries and backend	Sashank, Amit & Shayan	April 18	April 20
UI coding with multiple vm and VMhost support	Sashank & Shayan	April 19	April 23
Integration testing	Sashank, Shayan & Amit	April 23	April 27

Final Documentation	Sashank,Shayan & Amit	April 25	April 28

Table 13 : The schedule for the VM statistics team.

7.3 MN Load Team (Dan Howard, Roopak Venkatakrishnan)

This team worked using pair programming for the most part. Some of the smaller modules were done separately and later discussed to check if the ideas that were come up with were not biased

Schedule

Task	Personnel	Start Date	End Date
Identify metrics and how to measure load on MN	Dan & Roopak	Mar 7	Mar 9
Look through VCL documentation and understand what we have to work with	Dan & Roopak	Mar 11	Mar 13
Come up with design for first set of requirements received from VCL Engineers	Dan & Roopak	Mar 13	Mar 15
Research into how a heat map could be generated using HTML and open source libraries	Roopak	Mar 16	Mar 22
Identify specific sources and how this data can be used for generating heatmap & weathermap	Dan	Mar 16	Mar 22
Identify a mechanism for finding physical location based on remote IP.	Dan & Roopak	Mar 25	Mar 26
Set up and get basic working copy of GeoIP database on a local system and test the set up	Roopak	Mar 26	Mar 29
Design color schemes for heatmap and choose one to work with based on	Dan	Mar 26	Mar 29

feedback from VCL Engineers			
Put together a working model of a “tabular” heatmap based on the VCL data dump	Dan & Roopak	Mar 31	Apr 6
Analyze best method to come up with a weather map, and come up with algorithm for the same	Dan & Roopak	Apr 8	Apr 12
Optimize database for GeoIP queries	Roshan & Roopak	Apr 15	Apr 18
Code for a working version of the weathermap	Dan & Roopak	Apr 18	Apr 22
Integrate heatmap and weathermap with VCL	Dan & Roopak	Apr 23	Apr 26
Work on documentation	Dan & Roopak	Apr 25	Apr 28

Table 14 : The schedule for the MN load team.

7.4 Billing Team (Roshan George, Shyam Prasad)

Schedule

Task	Personnel	Start Date	End Date
Meet VCL Engineers to discuss requirements for billing project.	Roshan	Mar 4	Mar 4
Write Requirements Specification Draft	Roshan, Shyam	Mar 4	Mar 6
Write First Draft of Implementation Specification	Roshan, Shyam	Mar 5	Mar 6
Observe database behaviour by creating reservations and testing queries and database structure	Roshan, Shyam	Mar 5	Mar 7
Revise implementation specification using information gained from observations	Roshan, Shyam	Mar 8	Mar 9
Repeat behaviour tests to verify new implementation	Shyam	Mar 10	Mar 12

Mockup original interface	Roshan, Shyam	Mar 13	Mar 15
Meet VCL Engineers to discuss changes to requirements	Roshan	Mar 19	Mar 19
Rewrite implementation specification.	Roshan, Shyam	Mar 20	Mar 20
Identify locations where billing logic is used	Shyam	Mar 20	Mar 20
Write back-end billing code	Roshan, Shyam	Mar 20	Mar 27
Mock up second iteration interface	Roshan	Mar 21	Mar 23
Meet VCL Engineers to discuss progress and changes to requirements	Roshan	Mar 27	Mar 27
Write final iteration of billing requirements document	Roshan, Shyam	Mar 27	Mar 28
Mockup final interface	Roshan, Shyam	Mar 28	Mar 30
Rewrite back-end billing code to accommodate changes	Shyam	Mar 31	Apr 1
Write bill viewing and rate setting interface code	Roshan	Mar 31	Apr 6
Integrate UI into VCL	Roshan	Apr 4	Apr 15
Verify functionality	Roshan, Shyam	Apr 10	Apr 23
Ensure code-base test compliance	Roshan, Shyam	Apr 21	Apr 23
Complete module	Roshan, Shyam	-	Apr 24
Enhancements and Documentation	Roshan, Shyam	Apr 25	Apr 26

Table 15 : The schedule for the billing team.

7.5 Project Schedule

Target Completion Date of Entire Project : Apr 26th 2013
Target Completion Date of Report : Apr 29th 2013

Appendices

Billing Example Tables

In the following examples, an appropriate view of the database is presented. Therefore, not all columns for a specific table are displayed. Instead, only relevant information is shown.

user:

id	unityid
0	rvenkat7
1	rgeorge2

Table 16 : An example of data that could be contained in the certain columns of the user table.

usergroup:

id	name
0	admins
1	test

Table 17 : An example of data that could be contained in the certain columns of the usergroup table

image:

id	name
0	centos
1	winxp

Table 18 : An example of data that could be contained in the certain columns of the image table.

rate:

id	userid	usergrpid	imageid	rate
0	0	NULL	0	1
1	0	NULL	1	2
2	1	NULL	0	0.5
3	1	NULL	1	1.5
4	NULL	test	0	0.5
5	NULL	admins	0	0.25
6	NULL	test	1	1
7	NULL	admins	1	0.1

Table 19 : An example of data that could be contained in the certain columns of the rate table.

If the system is set up in the way specified above, we will describe what will happen if user rvenkat7 makes reservations for centos and winxp for 30 minutes, pays for them, and then reserves centos for 60 minutes. Immediately after rvenkat7's first reservation, user rgeorge2 makes a reservation for centos for 120 minutes but the reservation has not yet completed. The time is now 2012-03-21 01:00 (1 AM on the 21st of March).

`log:`

id	**loaded**	**initialend**	**finalend**	**imageid**
0	2012-03-21 00:00	2012-03-21 00:30	2012-03-21 00:30	0
1	2012-03-21 00:00	2012-03-21 02:00	1970-01-01 00:00	0
2	2012-03-21 00:00	2012-03-21 00:30	2012-03-21 00:30	1
3	2012-03-21 00:30	2012-03-21 01:30	1970-01-01 00:00	0

Table 20 : An example of data that could be contained in the certain columns of the log table.

`bill:`

id	**userid**	**status**	**timepaid**
0	0	paid	2012-03-21 01:00
1	1	pending	1970-01-01 00:00
2	0	pending	1970-01-01 00:00

Table 21 : An example of data that could be contained in the certain columns of the bill table.

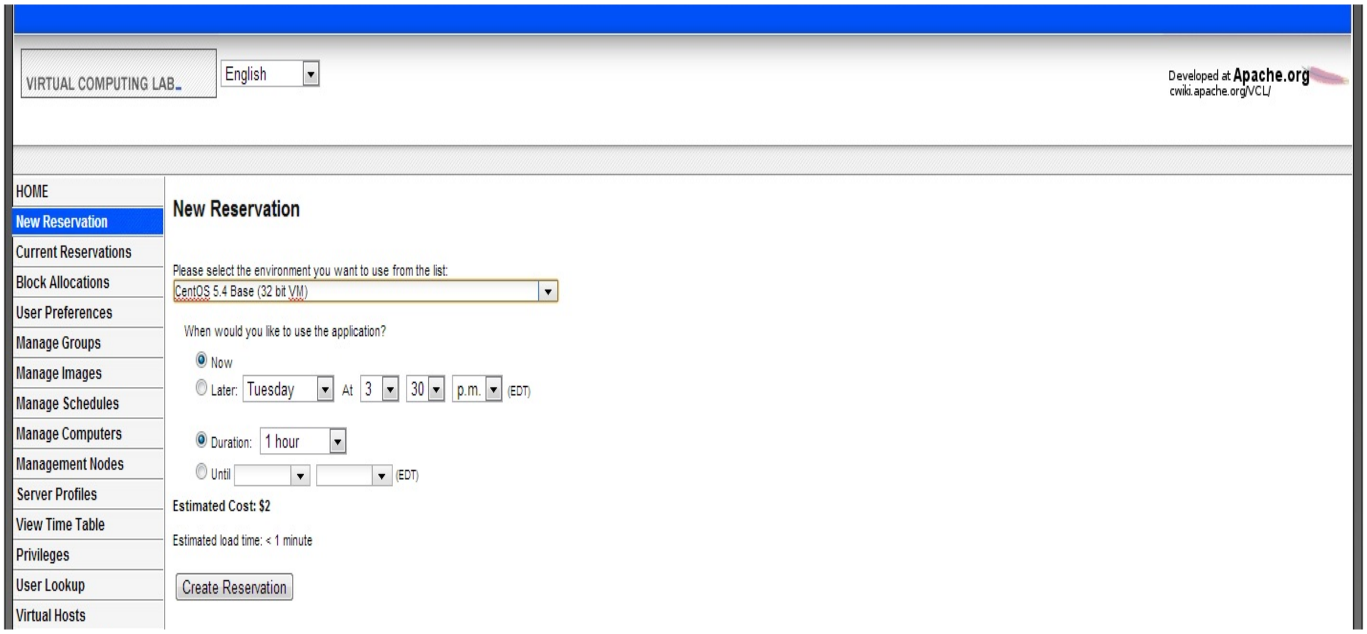
`bill_line_item:`

id	**billid**	**length**	**reservationid**	**cost**
0	0	30	0	30
1	1	0	1	0
2	0	30	2	60
3	2	0	3	0

Table 22 : An example of data that could be contained in the certain columns of the bill_line_item table.

Therefore, we see that rvenkat7 has paid for bill 0 which consists of charges of 90 USD in total, and the other bills remain unpaid. Since the reservations have not yet completed, neither of those bills can be paid.

Billing Screenshots



New Reservation

Please select the environment you want to use from the list:
CentOS 5.4 Base (32 bit VM)

When would you like to use the application?

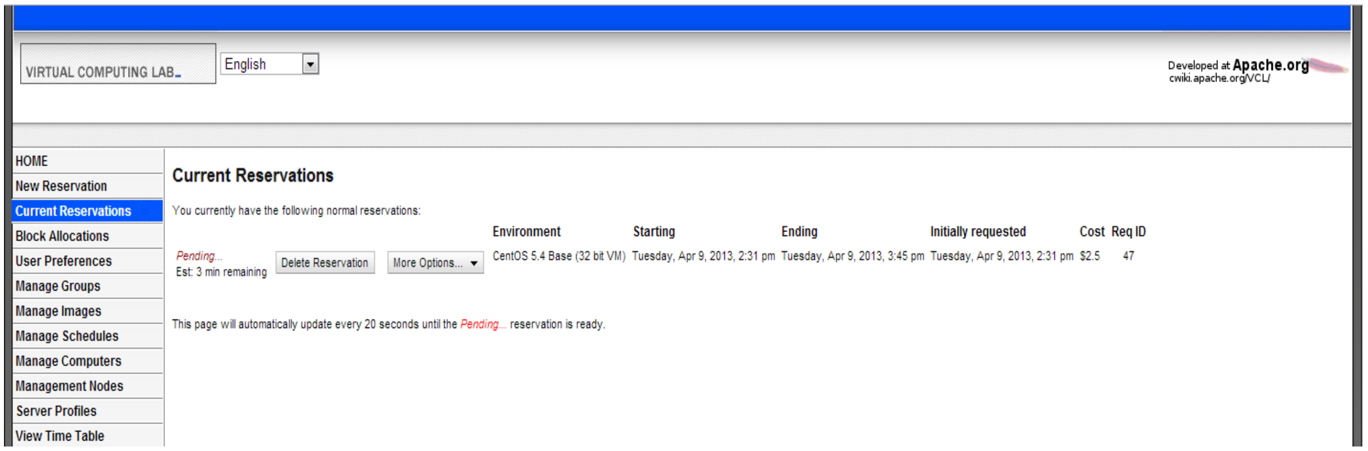
[✓ Now]
Later: Tuesday At 3 30 p.m. (EDT)

[✓ Duration: 1 hour]
Until (EDT)

Estimated Cost: $2

Estimated load time: < 1 minute

Figure 22 : A screenshot showing the estimated cost for a reservation



Current Reservations

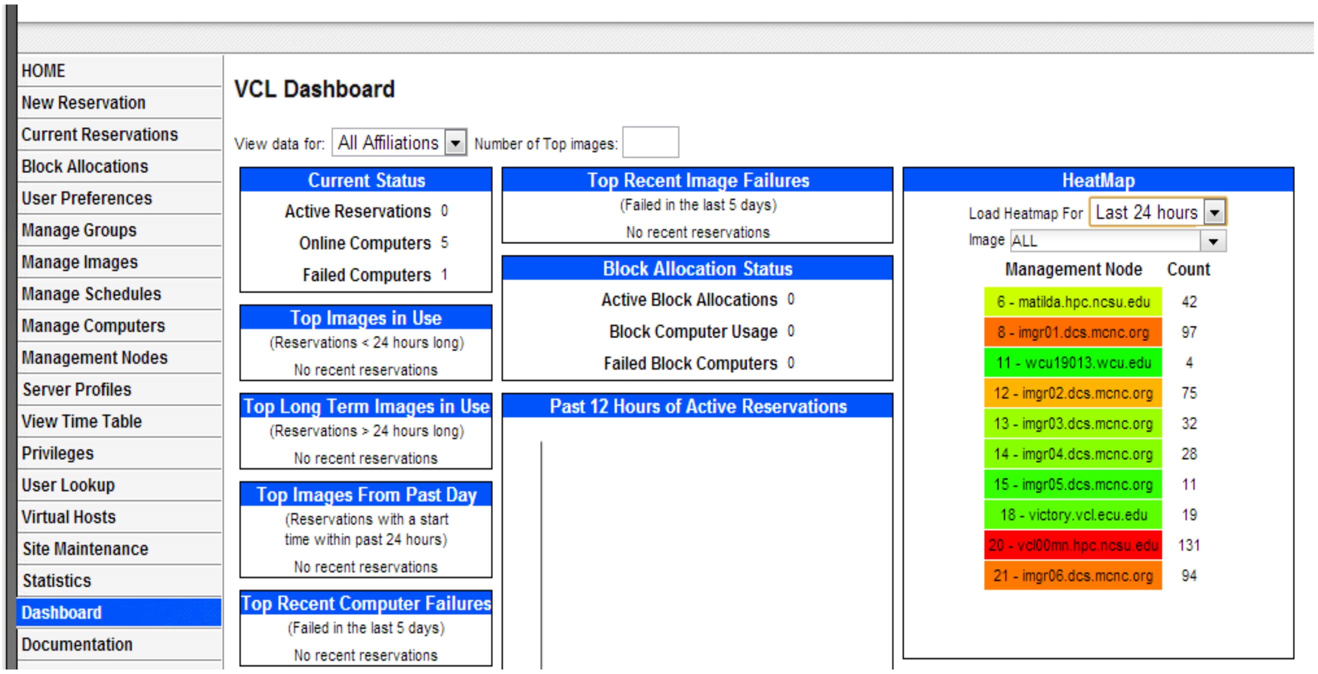
You currently have the following normal reservations:

	Environment	Starting	Ending	Initially requested	Cost	Req ID
Pending... Est: 3 min remaining	CentOS 5.4 Base (32 bit VM)	Tuesday, Apr 9, 2013, 2:31 pm	Tuesday, Apr 9, 2013, 3:45 pm	Tuesday, Apr 9, 2013, 2:31 pm	$2.5	47

This page will automatically update every 20 seconds until the Pending... reservation is ready.

Figure 23 : A screenshot showing the cost of the entire reservation at current rates.

MNLoad Screenshots



HOME
New Reservation
Current Reservations
Block Allocations
User Preferences
Manage Groups
Manage Images
Manage Schedules
Manage Computers
Management Nodes
Server Profiles
View Time Table
Privileges
User Lookup
Virtual Hosts
Site Maintenance
Statistics
Dashboard
Documentation

VCL Dashboard

View data for: All Affiliations Number of Top images:

Current Status
Active Reservations 0
Online Computers 5
Failed Computers 1

Top Images in Use
(Reservations < 24 hours long)
No recent reservations

Top Long Term Images in Use
(Reservations > 24 hours long)
No recent reservations

Top Images From Past Day
(Reservations with a start time within past 24 hours)
No recent reservations

Top Recent Computer Failures
(Failed in the last 5 days)
No recent reservations

Top Recent Image Failures
(Failed in the last 5 days)
No recent reservations

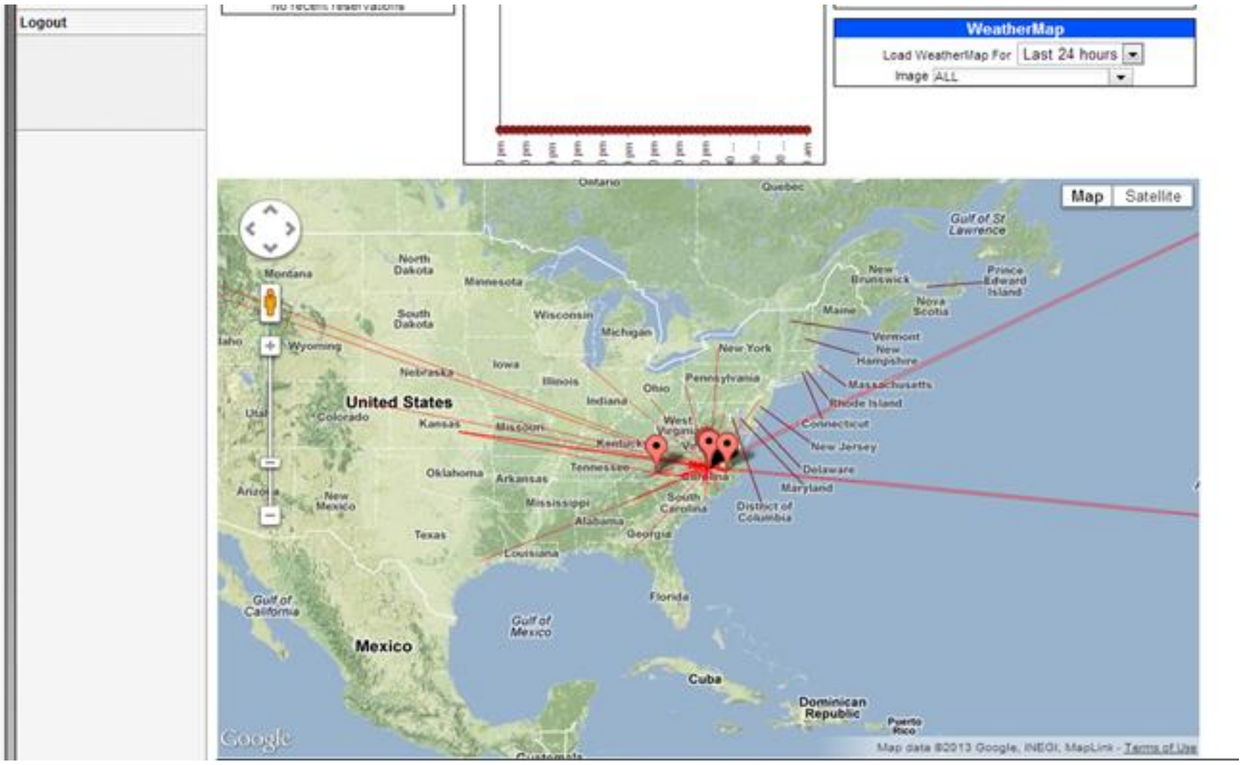
Block Allocation Status
Active Block Allocations 0
Block Computer Usage 0
Failed Block Computers 0

Past 12 Hours of Active Reservations

HeatMap
Load Heatmap For Last 24 hours
Image ALL

Management Node	Count
6 - matilda.hpc.ncsu.edu	42
8 - imgr01.dcs.mcnc.org	97
11 - wcu19013.wcu.edu	4
12 - imgr02.dcs.mcnc.org	75
13 - imgr03.dcs.mcnc.org	32
14 - imgr04.dcs.mcnc.org	28
15 - imgr05.dcs.mcnc.org	11
18 - victory.vcl.ecu.edu	19
20 - vcl00mn.hpc.ncsu.edu	131
21 - imgr06.dcs.mcnc.org	94

Figure 24 : A screenshot showing the heatmap for a 24 hour period.



Logout

WeatherMap
Load WeatherMap For Last 24 hours
Image ALL

Figure 25 : A screenshot showing weather map for a 24 hour period.

VMLoad Screenshots

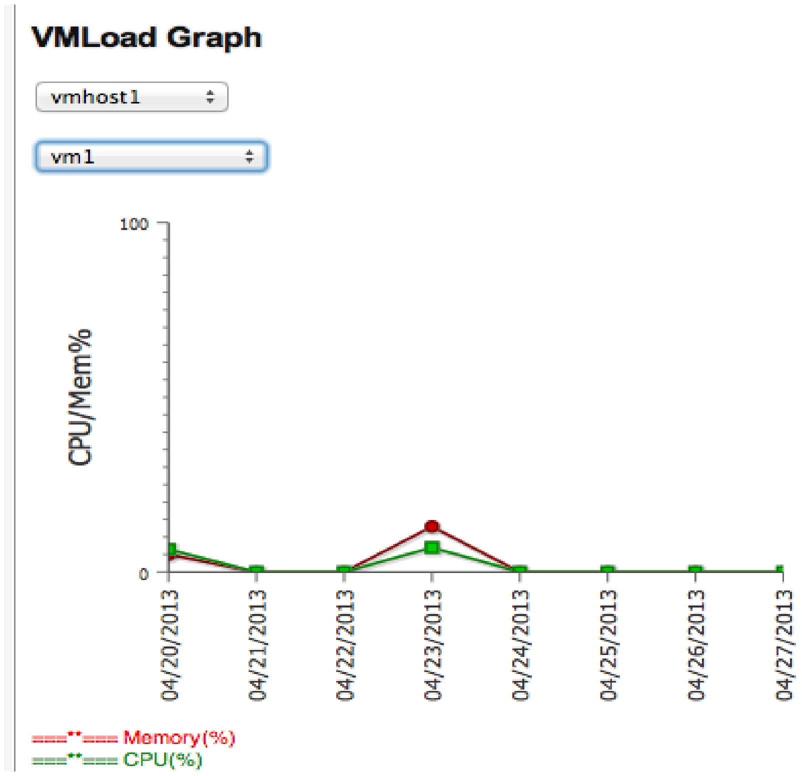


Figure 26 : A screenshot showing VM Load graph for the selected VMHost, VM.

Additional Documentation:

- In order to create the VCL cluster we used the VCL documentation hosted on the Apache Wiki at https://cwiki.apache.org/VCL/apache-vcl.html
- The VCL Database Documentation was available on the main VCL website under the Development Documentation section [7].

8 Future Work

During our demo with Mr. Streck and the VCL engineering team, we gained some valuable feedback regarding final improvements to our design. They asked us to implement a few changes in the MN Load and VM Load modules. Additionally, they asked for some notes on future implementations of the Billing module. We have listed the requests below. It may be noted that most of these requirements are additions to what we currently have, and would greatly assist admins of VCL to be able to see/view trends in VCL over periods of time. We hope to have the changes implemented by Friday, the 10th of May.

MN Load

- The current iteration allows an administrator to view the load on an MN over the past n hours, but it does not provide a longitudinal view.
- We will be implementing a feature that allows administrators to view trends over a given time period, in addition to the current implementation of a heat map.

VM Load

- The current iteration allows an administrator to view one VM at a time, but does not facilitate a comparative view.
- We will be incorporating a whisker plot displaying load statistics (i.e. memory and CPU utilization) across multiple VMs.
- Additionally, we will be changing criteria that form the basis of the information we track. Currently, we track VMs whose state is ‘available’. We will be changing this to track VMs whose state is ‘in use’.

Billing

- There are no changes planned for the billing module.
- In the future, the billing module should be incorporated with the policy engine.
  - Billing information can be used to ensure that usage policies are enforced.
  - A policy can be enforced based on the duration of reservations or on the total cost incurred by a reservation.
  - If the system is replicated using different weights, the total duration can be weighted by a different weight, depending on images.
  - This information could then be used to monitor and restrict access to VCL resources.

9 References

[1] Henry E. Schaffer, Samuel F. Averitt, Marc I. Hoit, Aaron Peeler, Eric D. Sills, and Mladen A. Vouk, “NCSUs Virtual Computing Lab: A Cloud Computing Solution," *IEEE Computer*, pp. 94-97, July 2009.

[2] A. Peeler, J. Thompson, A. Kurth, M. Vouk, H. Schaffer, et. al., “VCL Essentials," *Cloud Computing CSC 591*, North Carolina State University, slide. 5, Spring 2013.

[3] Apache VCL Website, *http://vcl.apache.org*, Retrieved April 8, 2013

[4] M. Hogan, F. Liu, A. Sokol, J. Tong, “NIST Cloud Computing Standards Roadmap,” National Institute of Standards and Technology, Gaithersburg, MD, Special Publication 500-291, 2011.

[5] Mills, K.; Filliben, J.; Dabrowski, C., "Comparing VM-Placement Algorithms for On-Demand Clouds," *Cloud Computing Technology and Science (CloudCom), 2011 IEEE Third International Conference on* , vol., no., pp.91,98, Nov. 29 2011-Dec. 1 2011
doi: 10.1109/CloudCom.2011.22

[6] Timothy Wood, Prashant Shenoy, Arun Venkataramani, and Mazin Yousif. 2007. Black-box and gray-box strategies for virtual machine migration. In *Proceedings of the 4th USENIX conference on Networked systems design implementation* (NSDI'07). USENIX Association, Berkeley, CA, USA, 17-17.

[7] Apache VCL Database Documentation, *http://vcl.apache.org/dev/database-schema.html*, Retrieved April 8, 2013.

[8] Process Monitoring with Nagios Plugins, http://www.nagios.com/solutions/linux-process-monitoring, Retrieved April 9, 2013

[9] Htop vs Top, http://htop.sourceforge.net/ ,
http://htop.sourceforge.net/index.php?page=comparison, Retrieved April 9, 2013

[10] CPAN DBI Module, http://search.cpan.org/dist/DBI/DBI.pm, Retrieved April 9, 2013

[11] Dojo Charting API, http://dojotoolkit.org/reference-guide/1.8/dojox/charting.html,
http://dojotoolkit.org/api/dojox/charting/Chart2D, Retrieved April 9, 2013

[12] Log Rotation, http://www.techpaste.com/2011/07/bash-script-rotate-logfilesclean-archive-zipped-logs/ Retrieved April 9, 2013