

**CMPE – 283 – Project 2 Spring – 2014**

**Large Scale Perf-Statistics gathering, DRS and DPM Implementation in Virtualized Environment**

**Submitted to**

Professor Simon Shim

Department of Computer Engineering

**Submitted By - Team 17**

Anita Gnanamalar Jebaraj

Aparna Sivakumar

Jasleen Kaur

Vaijayanthi Muralidharan

Contents

1. Introduction 3

1.1. Goals 3

1.2. Objectives 3

1.3. Needs 3

2. Background 4

3. Requirements 4

3.1. Functional Requirements 4

3.2. Non-Functional Requirements 5

4. Design 5

5. Implementation 8

5.1. Environment 8

5.2. Tools 8

5.3. Implementation Approach 10

6. Assumptions 12

7. Limitations 12

8. Future work and its extensions 12

9. Individual Contribution 13

10. Installation and Execution manual 14

11. Testing 21

12. Screenshots 26

13. Challenges 34

14. References 35

**1. Introduction:**

With the growing need of cloud infrastructure in the organizations, the importance of virtualization has also grown tremendously. Virtualization is used for setting up the development and test environments; service virtualization concepts are inevitable in testing of software. Handling the logs to monitor the performance of the virtual machines and balancing the loads based on the usage is gaining importance. These scenarios should be handled in a cost-effective and scalable manner.

**1.1. Goals:**

One of the major challenge faced in the use of virtual machines, is the collection of logs from the virtual machines. Perform load balancing of virtual machines between hosts and handle power management by using distributed resources scheduler and distributed power management concepts.

**1.2. Objectives:**

The key objective of the project is to collect the real time performance statistics of the virtual machines. The data are also stored in a scalable database in order to support the continuous collection of performance statistics. This contains both the historical data and transactional data, which are consolidated and displayed for analytical purposes.

**1.3. Needs:**

The performance statistics collected from the virtual machines are presented in the form of graph which help in better analysis of the status of the virtual machines, which in turn is required for making decisions regarding usage, load balancing, memory allocation etc. Distributed resource scheduler is required for balancing the loads, so that over utilization or underutilization of the resources can be avoided. By means of doing the distributed power management, we can power off the unneeded hosts and power on only when it is required again.

**2. Background:**

In this project, collection of large-scale statistics is done and displayed graphically for analytical purposes. Distributed resource scheduler helps in distributing the load across the hosts that are available. Hence when a new virtual machine is added, it automatically searches for the host with the least CPU usage and adds the new virtual machine under that host and thereby eliminating the manual analysis. In a similar way, distributed Power management is achieved, it monitors the hosts and virtual machines CPU usage periodically and switches off the host that are least used after migrating the virtual machines to the available hosts.

# 3. Requirements

**3.1. Functional Requirements**

1. Collect the performance statistics logs of the available hosts and virtual machines.
2. The logs collected must be stored in a no-sql database for any sort of future analysis.
3. From the huge volume of logs that are collected, aggregating the logs for every 5 minutes and loading into the mysql database should consolidate it.
4. The data from the mysql database, should be presented in the form of charts by using visualization tool.
5. The visualization tool must be able to present the data on a real-time basis by updating itself at regular time intervals.
6. The load balancing between the hosts must be handled both when a new virtual machine is added or when a new host is added.
7. The Power management between the host must be done automatically without any manual interruption.
8. When the CPU usage of a host goes down below a specified threshold level, the virtual machines under that host, must be migrated and balanced across the other available hosts between switching it off.

**3.2. Non-Functional Requirements**

1. The system must be designed to scale automatically for any future use
2. The presentation of data using visualization tools must be easily understandable.
3. The distributed resource scheduling and distributed power management should take place without any manual interruption.

**4.Design:**

The project is focused on creating a large scale statistics gathering and analysis tool, which is designed to be used on scalable virtualized environment with an emphasis on resource scheduling. For collecting the logs from the hosts and the virtual machines, a performance manager program is written in Java using VMware infrastructure API. This is connected to the vCenter and it gathers the logs from the hosts and the virtual machines in the vCenter. The logs from each VM and Host are written into separate text files appended with the name of those machines. The program is designed to gather the statistics indefinitely, until it is interrupted manually. The statistical parameters that are gathered are CPU usage, Network Statistics, Memory Statistics, System Statistics etc, which identifies the work loads on the machines. These data are collected at a time interval of 60 seconds and the data are appended into the text file.

**Logstash Usage:**

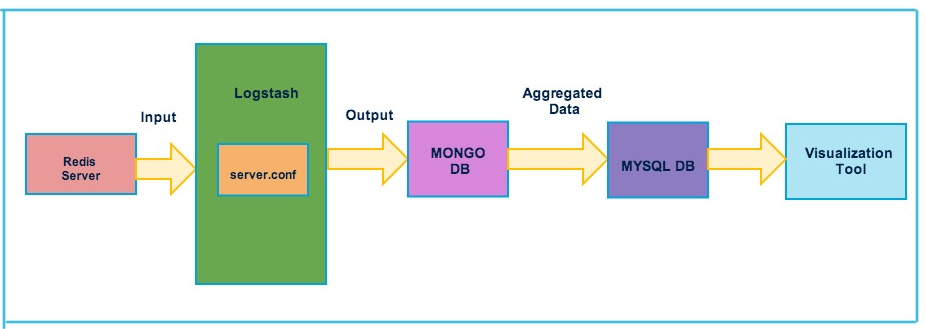
Logstash is a tool used for managing events and logs. It can be used to collect logs, parse them and store it for later use. The performance manager gathers enormous amount of logs, which is difficult to handle, hence it should be parsed and loaded into a database for analysis purposes.

****

Whatever the data entered into the Logstash is echoed back in a structured format. The performance manager java program is configured to output its log values into the text file in the form of json format, as it will be easier for loading data into the mongoDB at the later stage. The text files generated by using the performance manager is given as an input to the shipper.conf in Logstash and the output from the shipper.conf is loaded into the redis server. The Logstash is again used to load the data into the mongo DB, the data from the redis server is given as input into the server.conf file and the output is entered into the given database in mongo DB.

**Visualization Tool:**

Another action item of the project is to display the vast amount of logs in the form of graphs for easier analysis. High charts are used for the creation of graphs. The data in mongo DB contains the logs collected for every 60 seconds, the data is aggregated for every 5 minutes and entered into the MySql DB. A java program is used for aggregating and loading the data. During visualization the aggregated data from the mysql database is obtained and plotted in the form of graphs.

****

**Distributed Resource Scheduler:**

The distributed resource scheduling is done in order to balance the load across the hosts based on the CPU usage and availability. The balancing of load is done in two scenarios

1. Creating a new virtual machine
2. Creating a new host.

In the first scenario, when a new virtual machine is created and added to the environment, the load across the available CPUs are obtained from the log statistics that are collected and aggregated periodically in the mysql database. From this the load between the hosts are compared and the new virtual machine is added under the host that has the least CPU usage.

As per the second scenario, when a new host is added, the existing vm’s are load balanced accordingly.

**Distributed Power Management:**

Based on the CPU usage on each host . The host with the least CPU usage is selected and the corresponding vms are migrated to the other hosts based on the load balancing principle . Once the vms are migrated , the host is shutdown .

# 5.Implementation

**5.1. Environment**

Our system architecture is consists of a data center with two hosts, each host has VMware ESXi installed in it. The hosts are assigned with static Ips 130.65.133.52 and 130.65.133.53. The host 130.65.133.52 has two Virtual machines and the other host has one virtual machine. The hosts along with the virtual machines act as the client, from where the logs are collected using the java program for performance manager and the logs are collected in the text file. Another machine is used as the server, which has redis server, Logstash, mongo DB and MySQL installed in it, which is used for collecting, parsing and aggregating the data and displaying it for visualization and for resource scheduling.

**5.2. Tools**

**Eclipse:** Eclipse IDE is an open source tool, which is used for developing the java programs for the collection of performance metrics from hosts and virtual machines, Distributed Resource Scheduling and Distributed Power Management.

**OpenJDK 6:** OpenJDK 6 is an implementation on java SE 6 specification, based on jdk 7. It has the java class library and java compiler, which can be used for writing java applications.

**Logstash:** Logstash is an open source tool, we have used it for collecting, managing and parsing the logs and storing the logs into the mongo DB. The important components in the Logstash log management tool are,

1. Has Shipper component on all the agents (virtual machines and hosts) from where the logs will be collected.
2. Storing the data into the database.
3. Allows searching and viewing the log details on a specified time range.
4. Can be used for indexing the logs.
5. It has a broker, which receives the logs from different agents.

**High Charts:** High Charts is a simple and flexible API, which is used to creating graphs and virtualizing the performance statistics. It can be used along with JSP where the data are pulled from the My SQL database and plotted in the form of graph.

**Stress:** Stress is a stability testing tool used for varying the CPU load in the virtual machines by either increasing or decreasing the load. This is used primarily to test the workability of the performance manager by varying the CPU load to maximum and minimum loads and obtaining different statistics.

**Mongo DB:** Mongo DB is a nosql DB, which is used for storing the large amount of unstructured data generated in the form of logs.

**MySQL:** MySQL is an open source relational database management system, which is primarily used for storing structured data. The data from the mongo DB are aggregated and loaded into MySQL.

**5.3. Implementation Approach:**

**Performance Monitor:**

Collecting performance metrics are not inbuilt in any logs and required the use of VMware java API and are directly sent to log files associate with each vm. The log file from each vm gets created / appended correspondingly as the performance metrics collector runs continuously on them. This log files is filtered to extract the 5 essential metrics of the VM namely, CPU usage, memory usage, power usage, network usage which are plotted against time period.

**Snapshot of the file:**

**Steps to collect the metrics:**

Performance manager is created for host and VM entity.

To list the available intervals present, QueryPerfProviderSummary is used



To list the available metrics in the scope of the performance manager, QueryAvaliablePerfMetric is used.



The above gives the keys associated to each metric, to get the verbose description of each,



The below are the filtered keys with the verbose content:

2: Average usage of (CPU) in MHZ (rate)

24: Average usage of (mem) in percentage (absolute)

29: Average usage of granted (mem) in Kbps (absolute)

143: Average usage (net) in Kbps (rate)

157: Average usage of (power) in watt (rate)

**Distributed Resource Scheduler:**

Adding a new vm scenario:

* Refresh method is called which sorted the hosts according to their CPU usage
* Then vm is cloned from a template on the host with lowest CPU utilization.

Adding a new host scenario :

* First host is added by calling the addHost method which adds a host on network to our vcentre
* After host is added and powered on the hosts are sorted by their cpu usage in an arraylist
* Similarly all the vms are sorted in an arraylist
* Then each vm is trie to place in an optimal host such that the cpu usage of target host does not reach 80% of its capacity.
* After the target host for each vm is found and migrated ,and placed into appropriate hosts.

**Distributed Power Management:**

Distributed Power Management is used to balance the work load between the hosts and then power-off the hosts that are not utilized. As the first step all the hosts that are available in the vCenter and are sorted based on their CPU usage. Their CPU usage values are pulled from the MySQL database where the real time values are updated. If the CPU usage of the lowest host is less than 20 percentage and CPU usage of the highest host is less than 80 percentage. The dpm method continues else it returns. All the vms are transported to another host by calling the load balancing method. After the load balancing , host is shutdown.

# 6. Assumptions:

We have setup 2 Virtual Machines and 2 Hosts in which one of the Virtual Machine is considered to be the server. It has redis server, Logstash and mongo dB installed which acts as our Log Collector. This collects data from the rest of the virtual machines and the hosts. We have installed Logstash, Open JDK 7, and Eclipse on the client virtual machines. In order to gain higher range of real-time values for the metrics collected, we assume that Prime95 and software’s like stress are in installed while running Logstash.

# 7. Limitations:

The whole framework that we implemented has some limitations.

1. The implemented framework currently filters the performance keys related to few specified logs like CPU usage, Memory consumption, Network usage etc. The other additional metrics are recorded for group analysis.

2. The log collector tool used here, Logstash, is not compatible with the all of the plug-ins for various inputs, filters and outputs formats.

3. We have restricted the number of host to 2, for DRS1 (Initial placement) and DPM tasks due to the limitation of the lab environment.

# 8. Future Work and its extension:

1. This project can further be extended to combine the implementation of Availability Manager to determine the uptime and the availability manager and also logs related to them.
2. Most advanced metrics can be collected and can be graphically represented.
3. Automatic parsing of the logs can be scripted and a search option for log refinement can be done for analytics purpose.

# 9. Individual Contribution:

|  |  |
| --- | --- |
| **Name** | **Contribution** |
| Anita Gnanamalar Jebaraj | * Overall system framework and architecture * Visualization of metrics * Testing * Documentation |
| Aparna Sivakumar | * Overall system framework and architecture * Distributed Power management. * Unit Testing * Mongo- Mysql connection |
| Jasleen Kaur | * Configuration of Logstash * Backend administration * Integration Testing * DRS – Add host scenario. |
| Vaijayanthi Muralidharan | * Performance Mointoring and log collection * Unit and Intergration Testing * DRS- Create VM scenario * Logstash – Mongo connectivity |

# 10. Installation and Execution manual

**Logstash:**

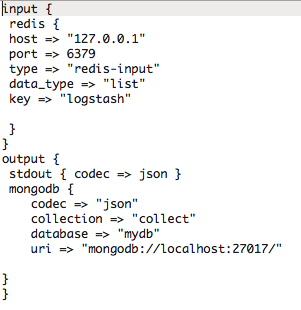
Logstash is used to collect and parse the logs and load it into the mongo DB. To install Logstash in Ubuntu Based systems by downloading the dependency files from the specified URL using the wget command and then extracting the files. Logstash requires the installation of java in the system as a pre-requisite as it is written in jruby.



Logstash shipper component is used to transfer the data from the text file, which contains the logs from the virtual machines and hosts into the redis server. The server component is used to transfer the data from the redis server into the mongo DB. The data in the text file are loaded in the json format to eliminate other tedious parsing while loading data into the mongoDB. A database and collection is created in the mongo DB in advance and the names are used in the conf files. The following figure shows the shipper.conf, which has the input as the test file and the output as the redis server.



The following figure shows the server.conf, which has the redis server as input and mongo DB as output.



**Redis Server:**

Redis Server is available open-source and it is used for key-value data storage. It is used as an intermediate server, before transferring the data from the text files into mongo DB in Logstash, the steps involved in installing redis in Ubuntu based systems are,

The redis server files are obtained from the url using the wget command,

Wget <http://redis.googlecode.com/files/redis-2.4.16.tar.gz>

It is decompressed using the command,

tar xzf redis-2.4.16.tar.gz

then we have to navigate to the directory, where redis files are decompressed,

cd redis-2.4.16

Before proceeding with the installation the make test is run as it is recommended after which the installation is done.

make test

sudo make install

Then we navigated to the utils directory from where the script to install the server is run

cd utils

sudo ./install\_server.sh

The redis server runs on port 6379 as default, to start the redis server, the following command is used,

sudo service redis\_6379 start

The redis server’s host 127.0.0.1 and port 6379, the default port is given in the conf files, where the data is transferred initially before moving it into the mongo DB.

**Mongo DB:**

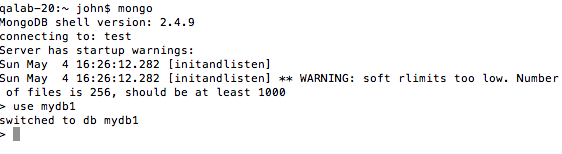
Mongo DB is a no-sql database, it is used to gather the huge amount of log, which will be in unstructured format. MongoDB can be installed in the Ubuntu based systems using the command,

sudo apt-get install mongodb-org

This command install mongo db in the Ubuntu Virtual machine along with all the dependencies that are required. To start the mongo Db server the following command is used,

sudo /etc/init.d/mongod start

this will start the mongo DB server in the port 27017. The Logstash will push the data from Redis server into mongo DB, before performing this step; a database should be created in Mongo DB. The name of the created database and collection will be specified in the server.conf of Logstash to move the data to the desired location.





The data from the mongo DB are then obtained using a java program and loaded into mysql DB programmatically.

**Stress:**

Stress is an open source tool, which is used to increase the load in the CPUs. It can be installed in the Ubuntu based systems using the command ‘apt-get’.

sudo apt-get install stress

Typing this command in the terminal installs the tool in the system by adding up all the dependencies of the tool. The to increase the load in the CPU, this following command can be entered in the terminal.

stress --cpu 2 --timeout 60

The requirements of the statistics gathering tool, demands collecting statistics continuously. Hence to test the system, the load has to vary for a specified time interval continuously. So the stress command is written in a shell script to run in an infinite loop for varying time intervals. Then depending upon the stress that is imposed in the CPU, the graph that is displayed using the visualization tool should vary accordingly. A shell script is used to run stress continuously, which helps in varying the statistics at regular time interval.

#! usr/bin

while true

do

stress --cpu 8 --timeout 10s;

sleep(300);

stress --cpu 4 --timeout 10s;

sleep (300);

stress --cpu 2 --timeout 10s;

sleep(300);

done

**Mysql:**

Mysql is a relational database that is used for storing structured data. It can be installed on Ubuntu based machines, using the command

sudo apt-get install mysql-server

this installs mysql in the system with all the required dependencies. The mysql server can be started using the command,

sudo service mysql restart

The data loaded in the mongo DB involves large amount of data as the data are collected for every 60 secs. The data are aggregated for every 5 minutes and loaded into mysql database using the java program. This structured data is used for plotting the graphs using high charts.

**11. TESTING:**

**Unit Testing:**

We have written test scripts in order to tests the smaller functionalities of our java code. Unit test/cases for collecting the number of hosts, VM’s , generating log files, writing the data in the files has be written

**Integration Testing:**

In order to perform integration testing, we combined the above files into one single file and ran the logStash to test it.

**Individual Contribution in Testing:**

**Jasleen :** Worked on the unit testing of

* LogStash
* Mongodb
* MySql

**Aparna and Anita:** Worked on integration testing of Logstash ,Mongodb , Mysql and visualization.

**Vaijayanthi:** Worked on Unit and Integration testing of

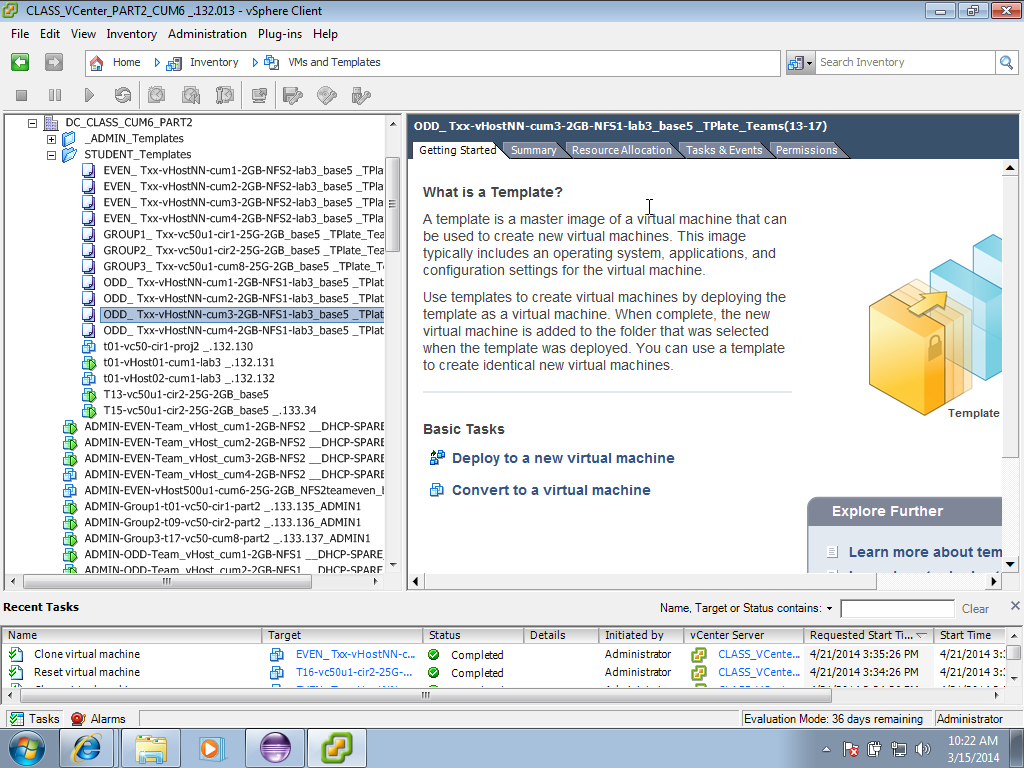
* Creation of log files
* Collecting total hosts
* Collecting total VM’s

| **Test Case Id** | **Test case Description** | **Test case Status** | **Expected Result** | **Actual Result** | | Input and Output Files |
| --- | --- | --- | --- | --- | --- | --- |
| **Pass** | **Fail** |
| **1** | **RealTimePerfMonitor\_Host\_vj.java** |  |  |  |  |  |
| **1.1** | Positive Test Case:  Test case ensures that the performance statistics like CPU usage, Network Statistics, System Usage, Memory Statistics and Storage Statistics of all the Hosts and the virtual machines are obtained when the VMs are turned on.  Verification Screen shot:  The status of the available hosts and VMs from the vSphere client and the text files with data. |  | The Statistics of the hosts and the VMs that are obtained are appended to the appropriate text files. | yes |  | Screenshot showing the available hosts and VMs in vSphere Client    Text files containing the data gathered |
| **1.2** | Negative Test Case:  Test case ensures that the performance statistics like CPU usage, Network Statistics, System Usage, Memory Statistics and Storage Statistics of all the Hosts and the virtual machines are obtained when the VMs are turned off.  Verification Screen Shot:  The status of the available hosts and VMs from the vSphere client and the test files with data. |  | The Statistics of the hosts and the VMs should be modified as per the current status of the VMs. | yes |  | Text files containing the data gathered. |
| **2.** | Data Transfer into Mongo DB using Logstash.  shipper.conf and server.conf |  |  |  |  | Screenshots showing data transferred into mongo DB |
| **2.1** | Test Case ensures the data from the text files are transferred into MongoDB.  Verification Screen Shot:  Data loading into mongo DB and mongo DB with updated data. |  | The data from the text files are loaded into the redis server using Logstash, which is done using shipper.conf and then the data from the redis server is transferred into mongo DB using Logstash, which is done using server.conf | yes |  | Screenshots showing the data loaded into mongo DB |
| **3.** | **MONGOtosql.java** |  |  |  |  |  |
| **3.1** | **Positive Test Case:**  Test case ensures that the data loaded into the mongo DB is aggregated for every 5 minutes and loaded into mysql DB.  Verification Screen Shots:  Data in mongo DB and aggregated data in mysql. |  | The data from mongo DB is aggregated and pushed into mysql | Yes |  | Screenshot showing data aggregated and loaded in mysql. |
|  | **Negative Test Case:**  Test case ensures that the program throws an error when the data is not loaded into mysql Db.  Verification Screen shots:  Data in mongo DB and the error thrown due to unavailability of mysql. |  | The program throws a human readable error when the mysql db is not available. | Yes |  |  |
| **4.** | **Visualization:** |  |  |  |  |  |
| **4.1.** | Test Case ensures that the available hosts and VMs are listed in the drop down.  Verification Screen shot:  Screenshot showing the available VMs and Hosts in Vsphere Client and the listed VMs and Host in the drop down. |  | The available hosts and VMs are obtained and their names are loaded into the drop down. | Yes |  | Screenshot showing the data loaded into the select box: |
| **4.2.** | Test Case ensures the values in the mysql database and the values displayed in the graph matches  Verification Screenshots:  Values from mysql DB and the graphs displayed. |  | The values are obtained from the mysql DB and plotted as graph. | Yes |  | Screenshots showing the data plotted in the graphs and the data in mysql |
| **4.3** | Test Case ensures that the values in the graph are updated on a regular time interval depending upon the updated values in the mysql DB.  Verification Screenshots:  Previous screenshots and the updated screenshots after the specified time interval. |  | The values in the graph are updated after a specified time interval. |  |  | Screenshots showing updated data in graphs: |
| **5.** | **LoadBalance.java** |  |  |  |  |  |
| **5.1** | Test case ensures that the distributed resource scheduling is done when a new VM is added.  Verification Screen shot:  Screenshots showing the addition of a new VM, CPU usage of the available hosts and the VM being added to the host with the least CPU usage. |  | The new VM added is attached to the host with the least CPU usage at that particular time. | Yes |  | Screenshots showing DRS after adding new VM: |
| **5.2** | Test case ensures that the distributed resource scheduling is done when a new host is added.  Verification Screen shot:  Screen shots showing the addition of new host and the VMs migrating to the new hosts to balance the load. |  | The VMs that are available are migrated to the new host in order to balance the load when a new host is added. | Yes |  | Screenshots showing DRS after adding new host: |
| **5.3** | Test case ensures when the user chooses to perform distributed power management the load is balanced and the unused hosts are turned off.  Verification Screen shots:  Migration of VMs from the least used host and the powering off of hosts. |  | The user chooses to perform DPM, the usage of the available hosts are obtained from the mysql DB and the vms from the least used host is migrated to the other hosts and the load is balanced, after which the unused host is powered off | yes |  | Screenshots showing DPM load balancing: |

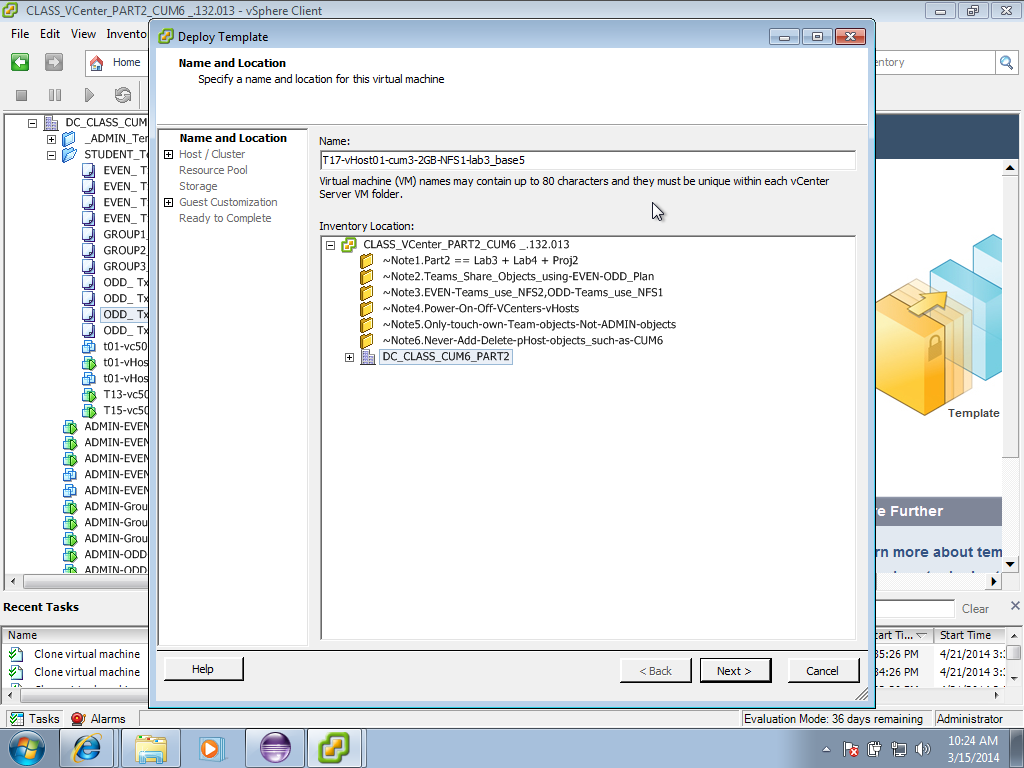
**12. Screenshots:**

**Environment Setup:**

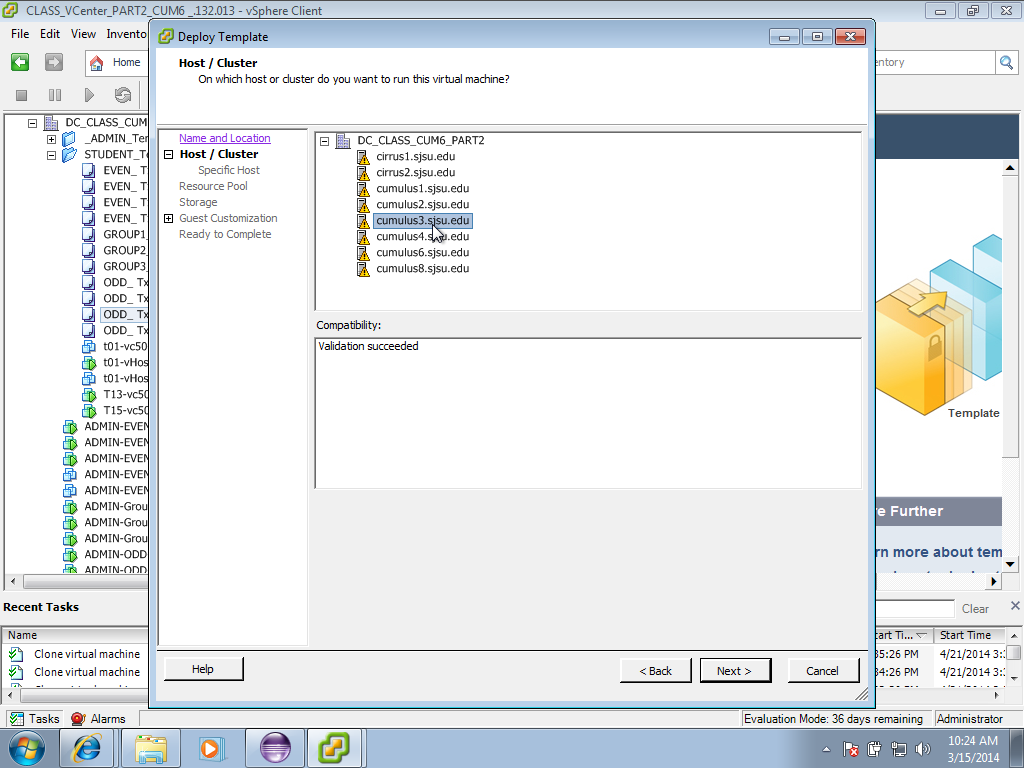
**Choosing the template:**

****

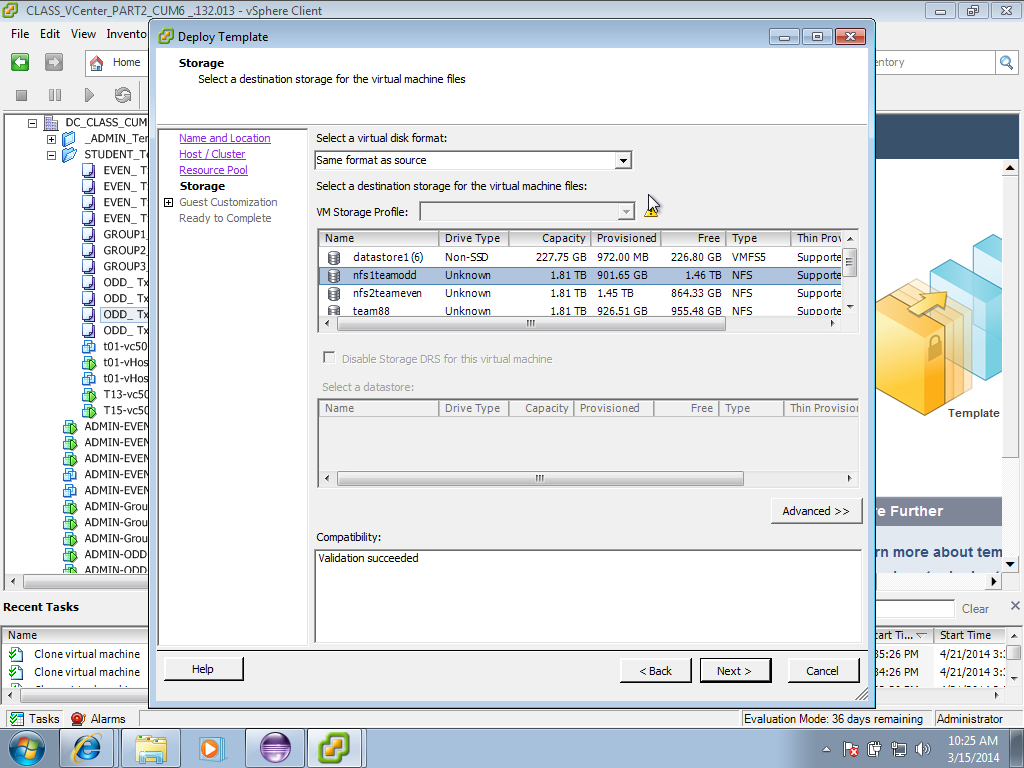
**Adding a new host**

****

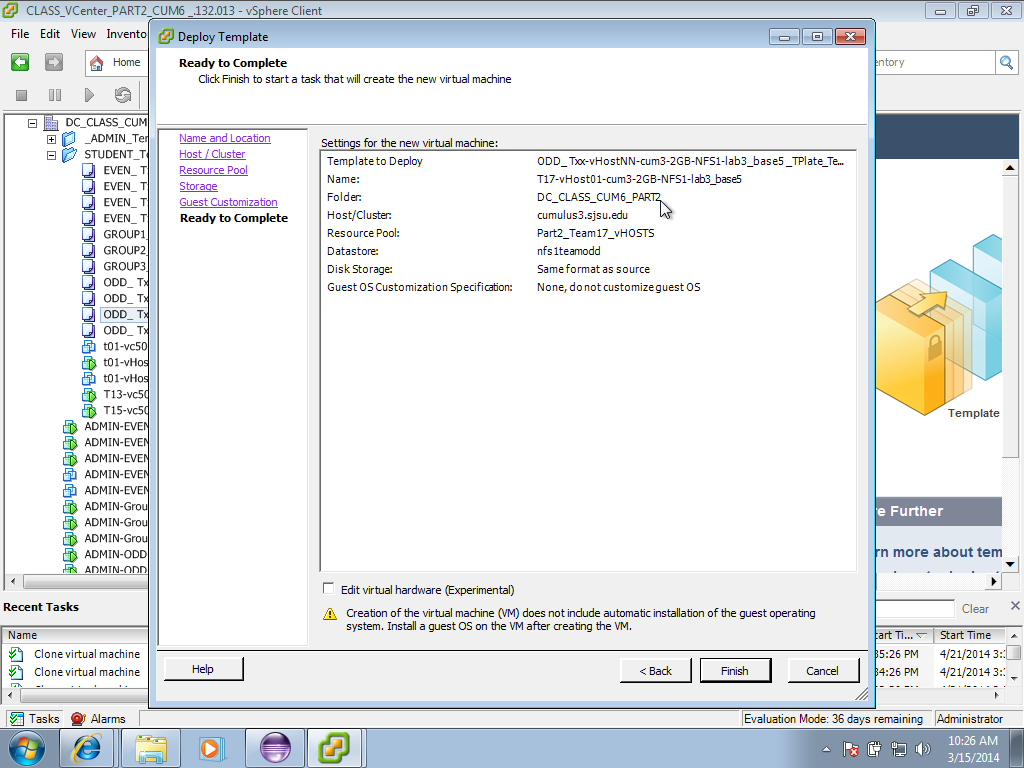
**Validating:**

****

**Choosing the storage for the host:**

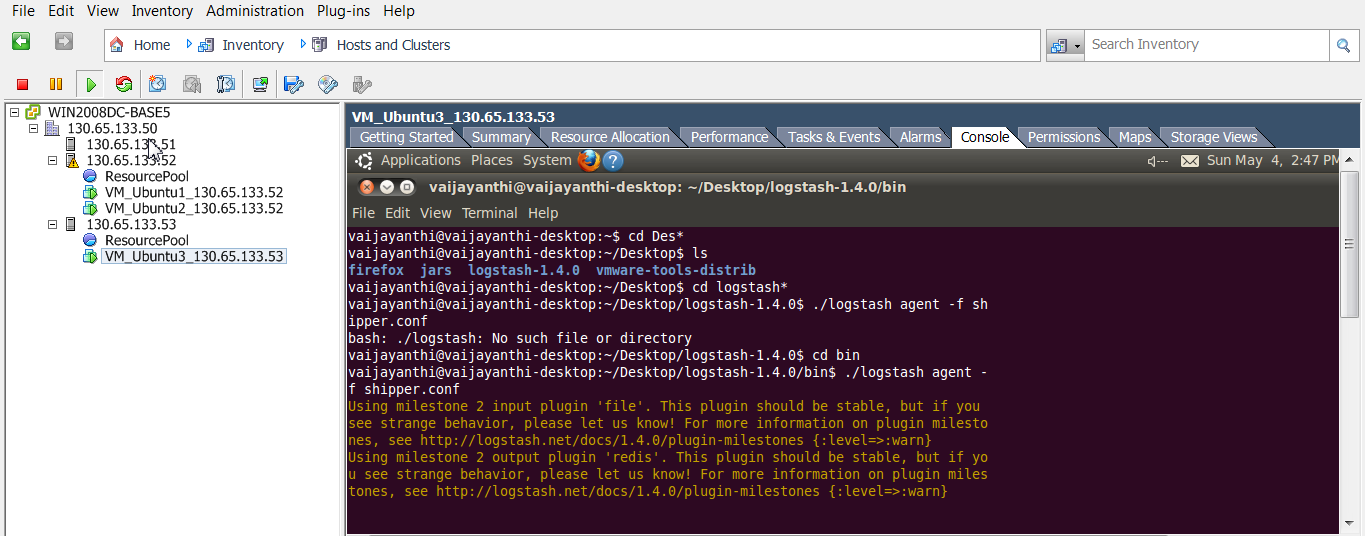
****

**Deploying the template:**

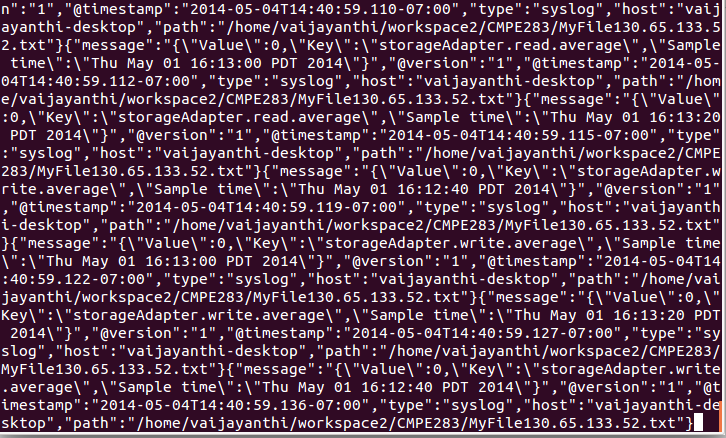
****

**Loading Data into MONGO DB using Logstash:**

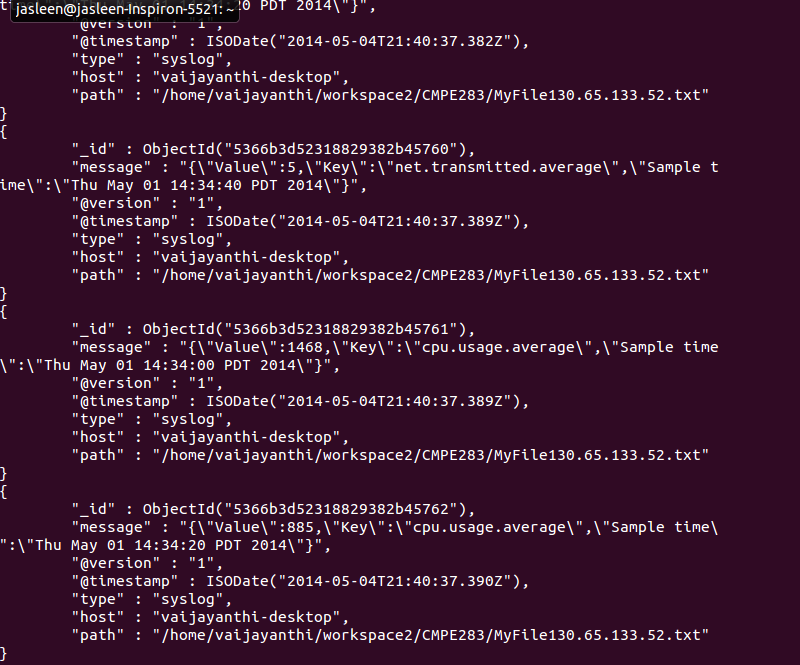
**Running shipper.conf to load data into redis server:**



**Loading Data into MONGO DB from Redis Server:**

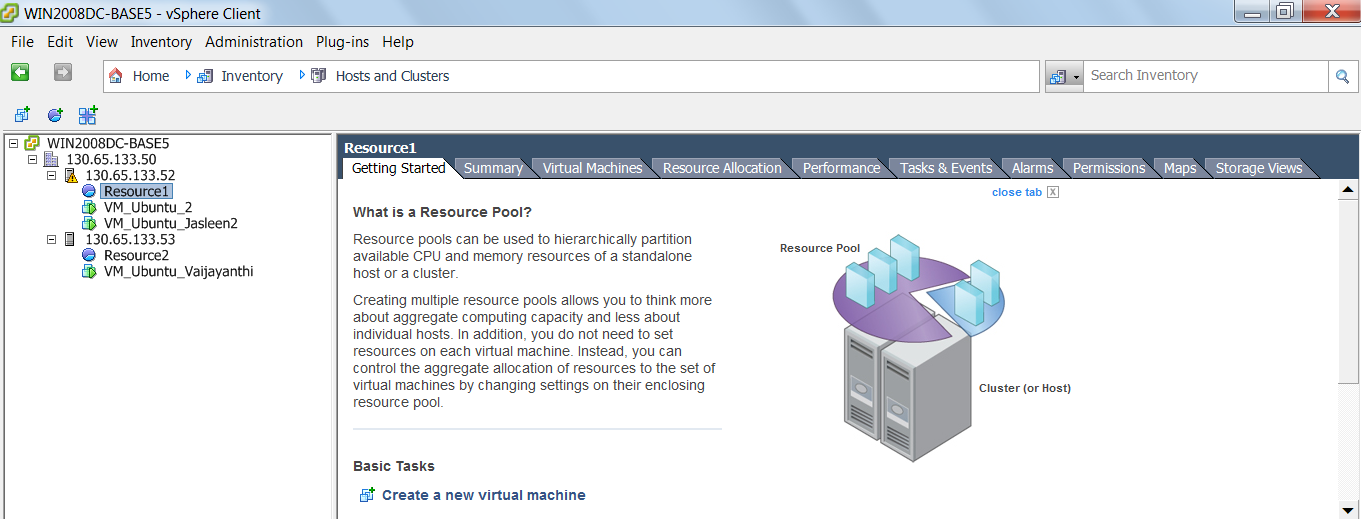
****

**Data in MONGO DB:**

****

**Distributed Resource Scheduler:**

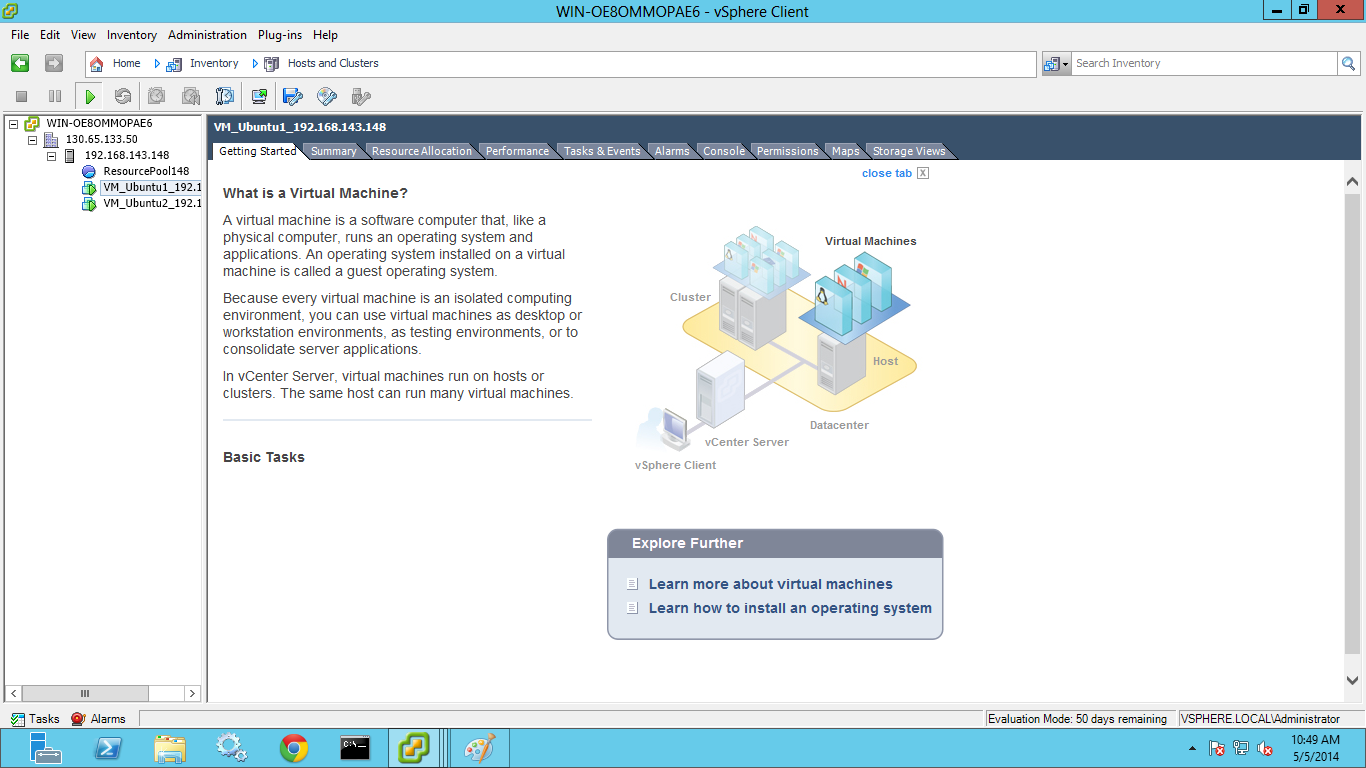
**Initial State before adding a new VM:**



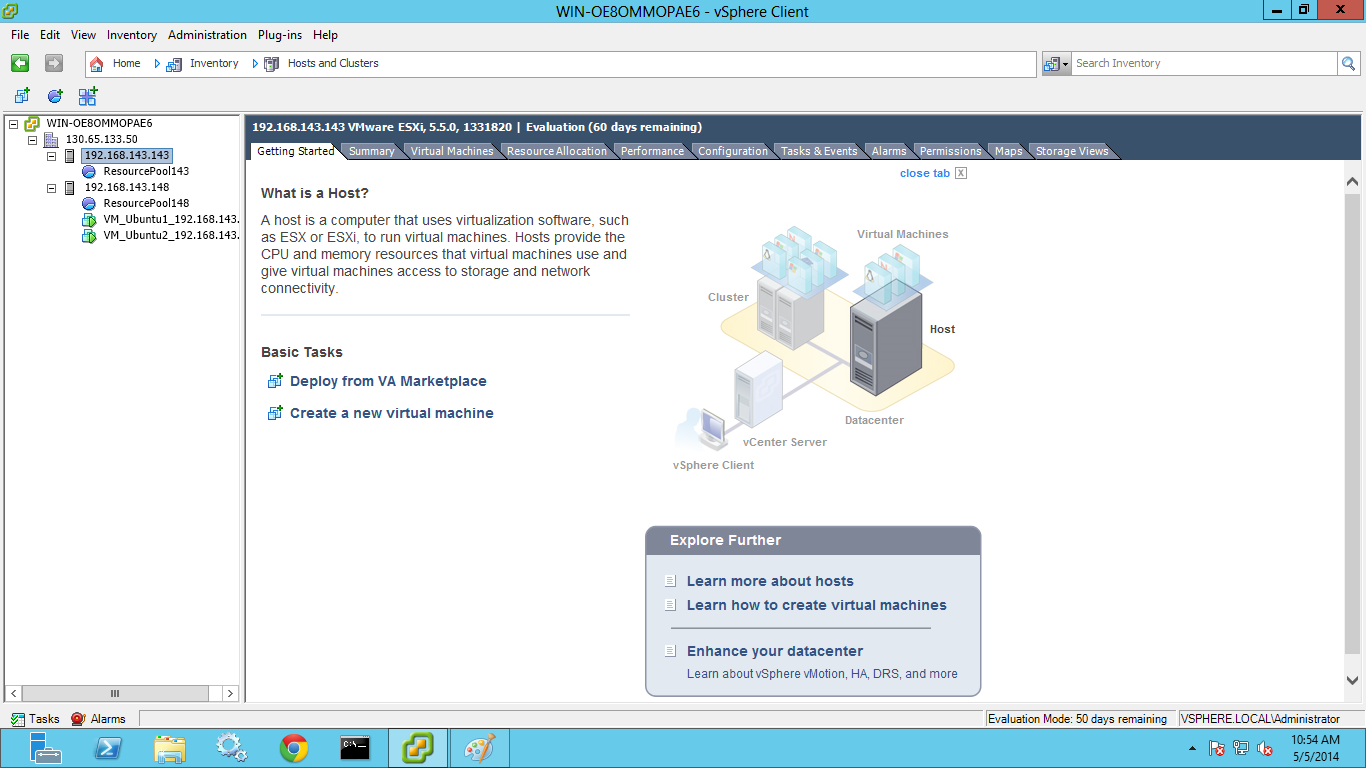
**Placing the VM in the host that has least CPU Usage:**



**Initial State Before adding new Host:**

****

**After adding New Host:**

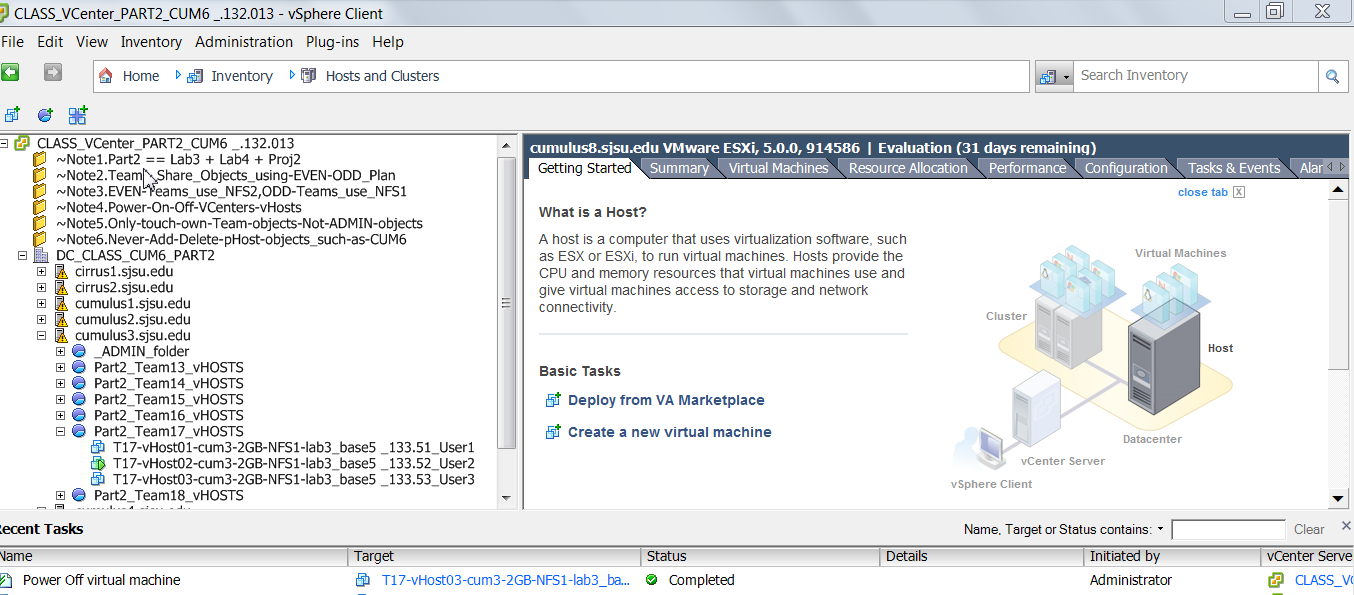
****

**Distributed Power Management:**

**Migrate Virtual Machines from the least used hosts to other hosts:**

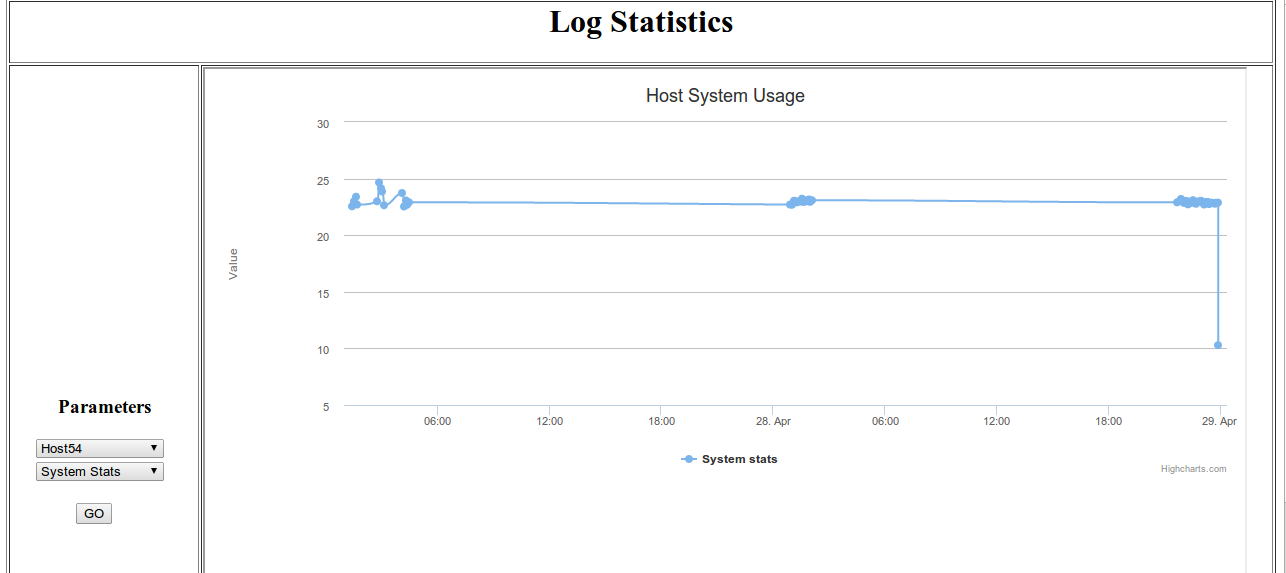


**Powering off the unused hosts after migrating the virtual machines:**

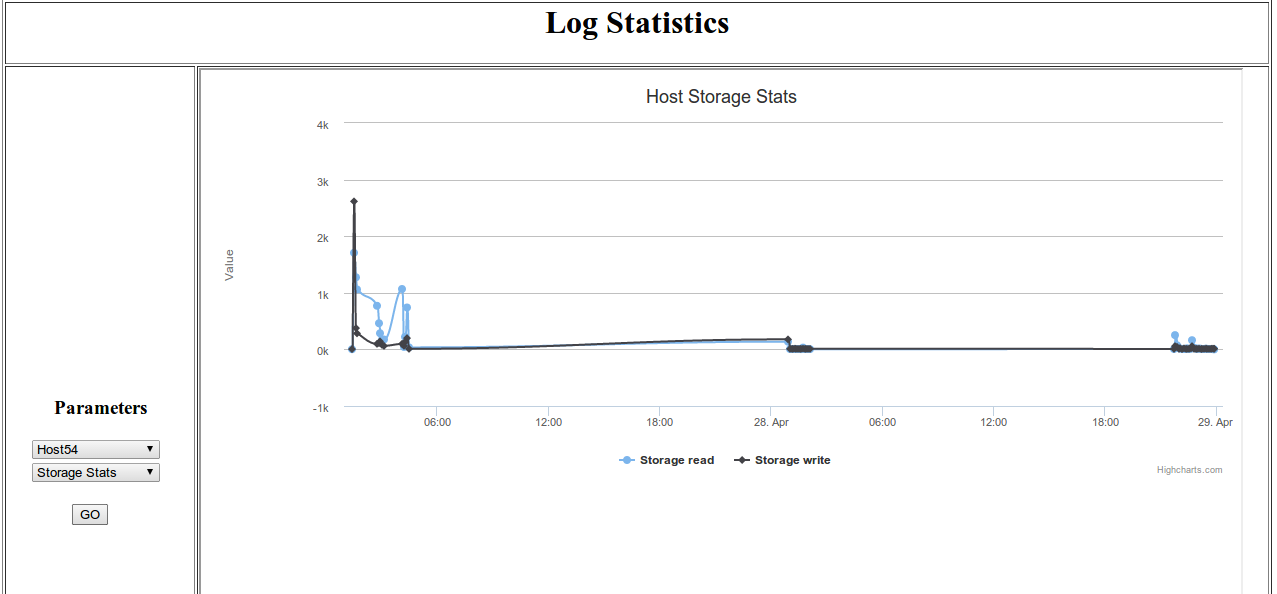


**Visualization Metrics Screenshots:**

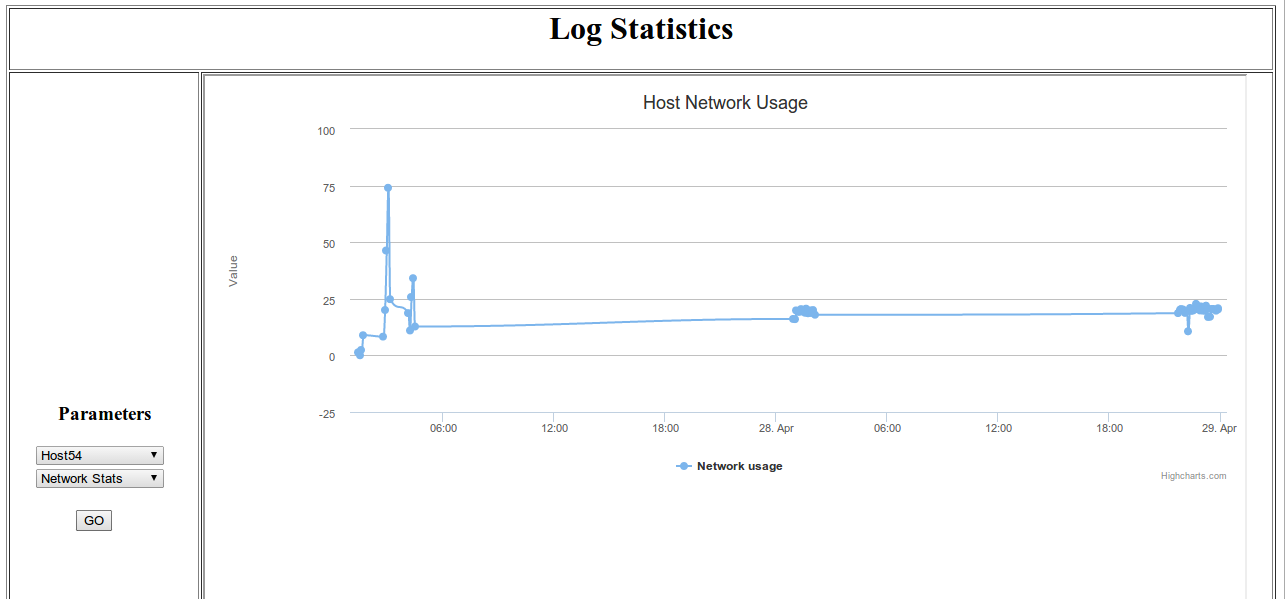
**Host System Usage:**



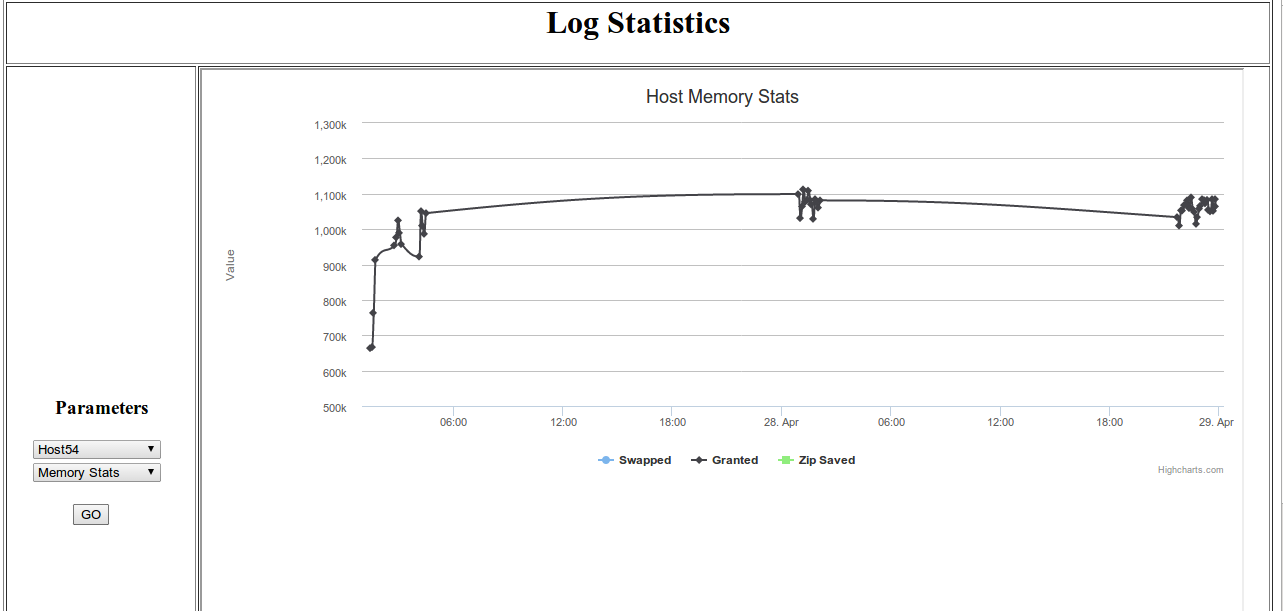
**Host Storage Statistics:**



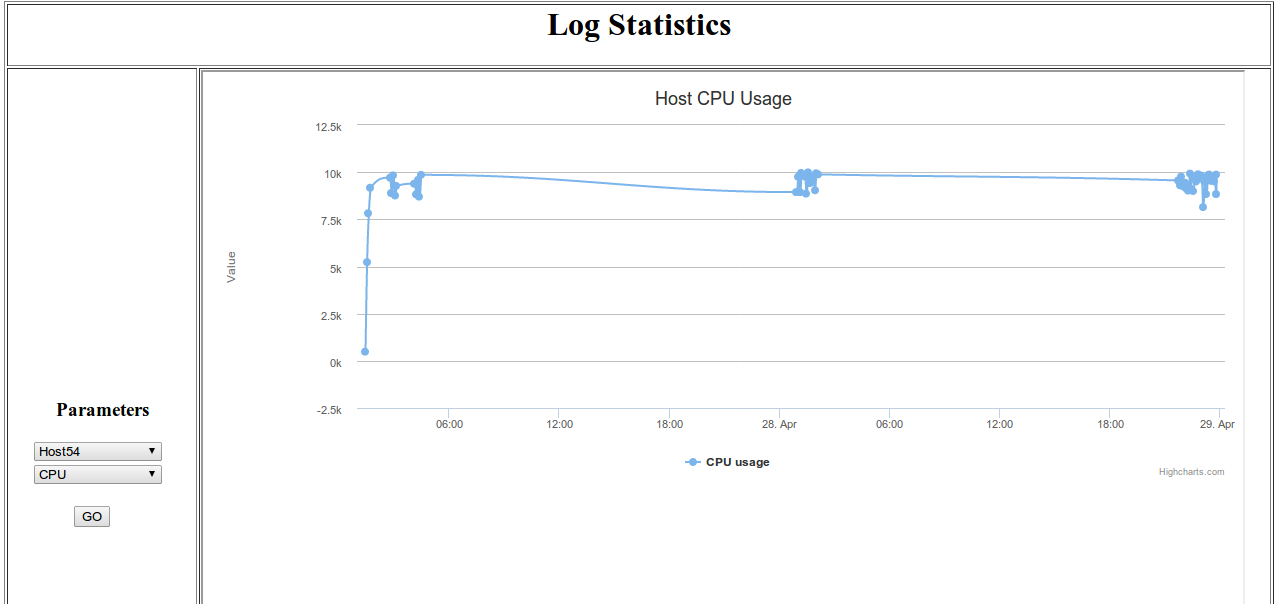
**Host Network Statistics:**



**Host Memory Statistics:**



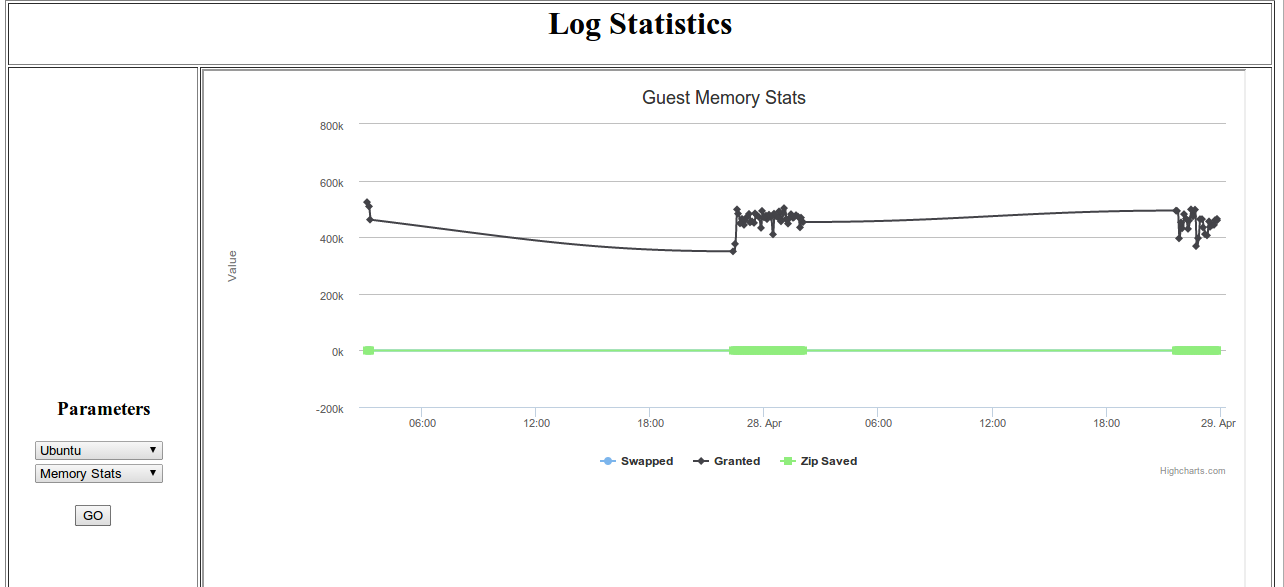
**Host CPU Usage:**



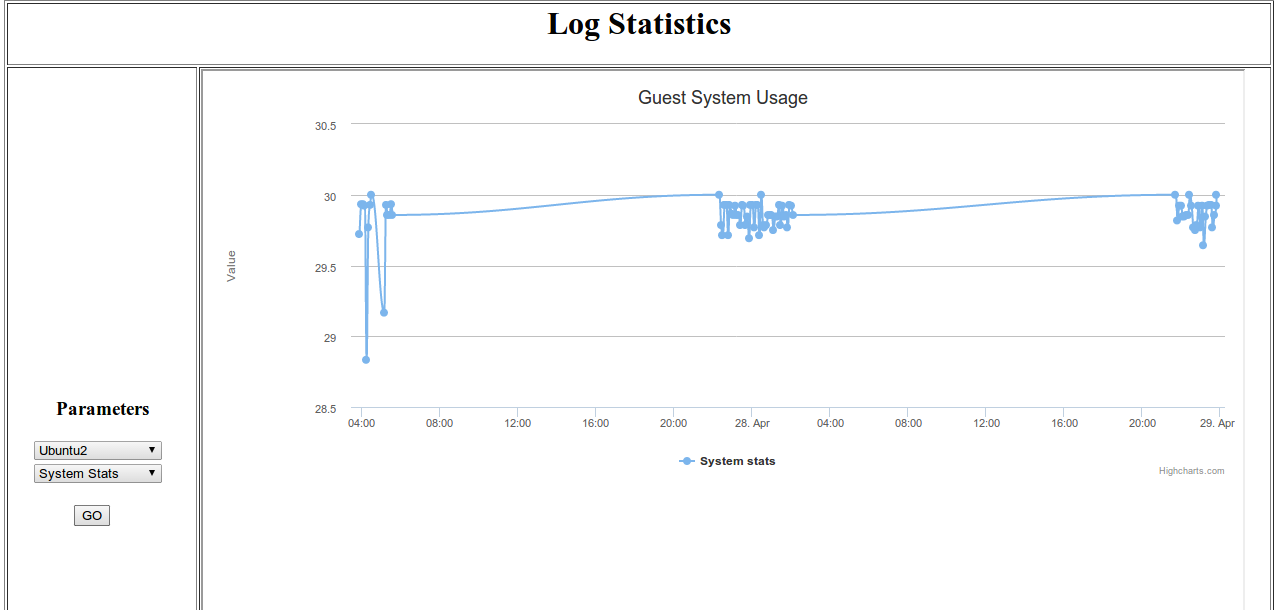
**Virtual Machine CPU Usage:**



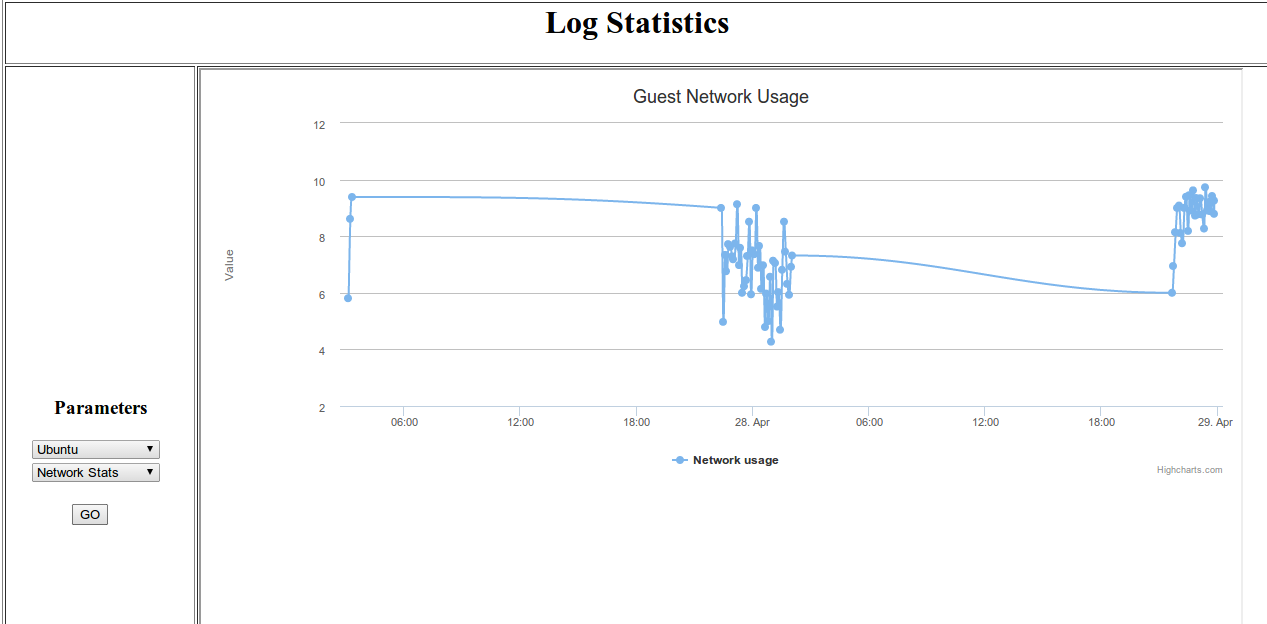
**Virtual Machine Memory Statistics:**



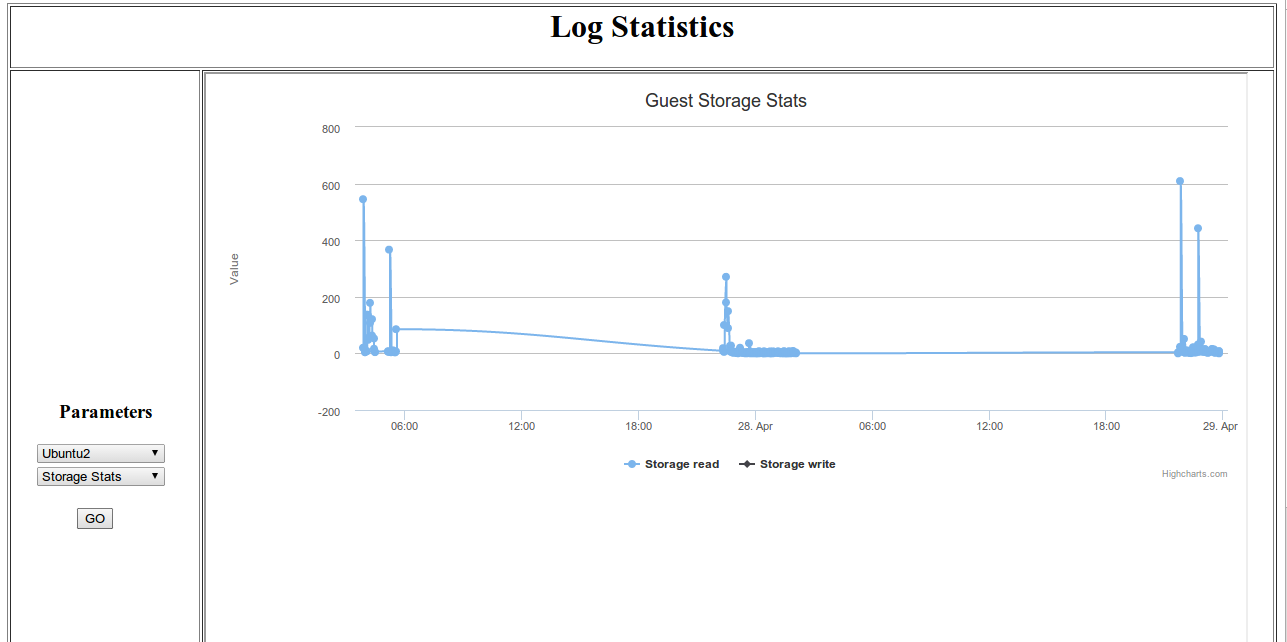
**Virtual Machine System Usage:**



**Virtual Machine Network Usage:**



**Virtual Machine Storage Statistics:**



**13. Challenges:**

1. The lab infrastructure inhibited large storage hence, we had to make sure we purged the log files collected on our Virtual Machines so that the machines did not run out of space.
2. Configuring Logstash and using plug-ins for determining the output forms were difficult.
3. Shipping of the logs from client to server and filtering performance keys of requirement was taxing task.
4. In the initial phase of the lab, we were required to collect just a few statistics from the huge performance log file that was getting generated. To identify which resources were needed and which not took a considerable amount of time.
5. Dynamic generation of graphs with real-time updation of response was very slow.
6. Implementing DRS and DPM could not be done without an algorithm and the algorithm which fitted this model was Knapsack problem. To  comprehend the problem and to bring it in sync with the Knapsack problem statement was a challenge. Once identified, it was rather easy for us to proceed with the algorithm.
7. Since the initial few labs did not involve creating new Resource pools within our local vCenter, creating and managing Resource Pools wa a issue.
8. Migrations of VMs as a part of DPM and at the same time maintaining synchronization with the mysql table real-time weren’t easy.

**14. References:**

1. <http://logstash.net>
2. <http://openjdk.java.net/projects/jdk6/>
3. <http://www.slashroot.in/logstash-tutorial-linux-central-logging-server>
4. <https://help.ubuntu.com/12.04/serverguide/mysql.html>
5. <https://www.digitalocean.com/community/articles/how-to-install-and-use-redis>