CS 6343 – Cloud Computing

Final report

December 07th, 2014

VM placement and scaling

Submitted to:

Dr. I-Ling Yen

Professor,

Department of Computer Science

The University of Texas at Dallas,

Richardson, TX-75080

Submitted by:

Team B1

virtdc 1.0.0

[Virtual Datacenter]

[Source code - https://github.com/dcsolvere/virtdc]

Team Members

Ruiyu Wang

Qinghao Dai

Dinesh Appavoo

Rahul Nair

Haan Mo Johng

Revision History

Date	Version	Changes	Editor
10/23/2014	0.1.0	Initial File Creation	Haan Mo Johng
10/24/2014	0.2.0	Make contents.	Ruiyu Wang Qinghao Dai Dinesh Appavoo Rahul Nair Haan Mo Johng
10/26/2014	0.3.0	Architecture diagram, Framework workflow and architecture details, Placement manager workflow details, Simulation	Ruiyu Wang Dinesh Appavoo Rahul Nair Qinghao Dai Haan Mo Johng
12/07/2014	1.0.0	TroubleShooting, Automation, Monitoring, Decision Maker, Future Work	Ruiyu Wang Dinesh Appavoo Rahul Nair Qinghao Dai Haan Mo Johng

1. INTRODUCTION

1.1. Purpose

Assume that the workloads of the VMs are known when submitted, statically place and schedule and dynamically scale and migrate the VMs to make best resource utilization while never letting users feel short of resources.

1.2. Milestones

1. First review
   A. Set up KVM
   B. Implement VM_submitJob to allow actual job submission and execution
   C. Know exactly how VM migration and VM CPU scaling works
   D. Take one Google workload, generate one simulated job to match the workload, actually submit the job (workload does not have to be very accurate yet)

2. Midterm report
   A. Complete some VM migration or scaling functions
   B. Complete a sequence of job submissions
   C. Workload does not have to be very accurate yet, but should have improvements
   D. Has a simple decision algorithm in placement manager, but placement manager should be able to work with other parts of the system
   E. Some workload information about the VMs should be available, even if it does not go to placement manager

3. Third review
   A. VM migration and scaling function fully functional
   B. VM monitoring agent and host listener fully functional
   C. VM termination API fully functional
   D. Real-Time monitoring graphing functional
   E. Infrastructure setting up automated by scripting
   F. Optional load-balancing and consolidating will be finished before final demo

2. VIRTDC ARCHITECTURE AND FRAMEWORK

2.1 VIRTDC – High Level Architecture Diagram

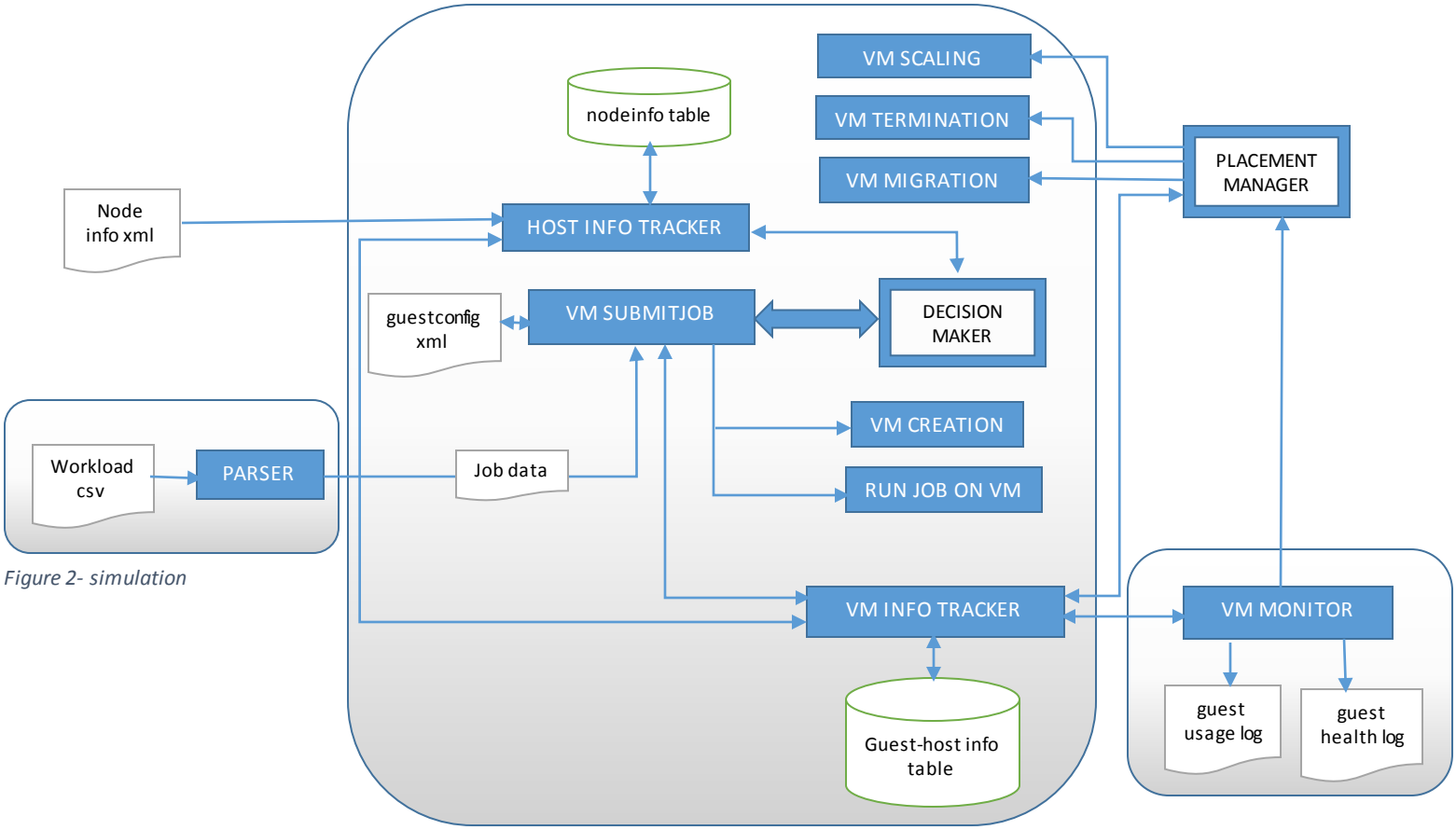


Figure 2- simulation

Figure 1 - framework

2.2 VIRTDC – Framework [libvirt, python, shell script, c]

Virtdc is an API for virtual machine placement and scaling which provides an environment to create, manage and monitor virtual machines effectively. Virtdc – framework provides API's to create virtual machines, maintain information about guest and host, terminate virtual machine and to handle static virtual machine placement. Internally it uses libvirt management API to accomplish this.

One host acts as the master node to create the virtual machines on any slave nodes. Guests can also be placed in master node.

Host Information:

Virtdc API maintains a table to store the host information. In the initial setup of virtdc API this table is updated from the nodeinfo.xml. This xml will be collected from the user. Host Info Tracker module will update the table based on the nodeinfo.xml. Whenever there is a new resource/node update, host info tracker module has to be executed. This module will tweak the table.

host	ip_adddress	max_cpu	max_memory	max_io	avail_cpu	avail_memory	avail_io
node1	192.168.1.11	8	32689796	1.07E+09	8	32689796	1.07E+09
node2	192.168.1.12	8	32689796	1.07E+09	8	32689796	1.07E+09
node3	192.168.1.13	8	32689796	1.07E+09	8	32689796	1.07E+09
node4	192.168.1.14	8	32689796	1.07E+09	8	32689796	1.07E+09

Figure 3

Guest Information:

Virtdc API maintains a table to store the guest information along with the host. This dictionary is accessible from all API's in virtdc but can be updated only by the master node. By default it provides disk/io size of 4GB for each guest.

host	vmid	current_cpu	max_cpu	current_memory	max_memory	io
node1	vm1	1	3	4194304	5242880	4194304
	vm2	2	3	4194304	5242880	4194304
	vm3	1	3	4194304	5242880	4194304
node2	vm4	2	5	4194304	5242880	4194304
	vm5	2	6	4194304	5242880	4194304
	vm6	3	4	4194304	5242880	4194304

Figure 4

2.3 Update Host Information

Data centers can add new resources to the existing system. To achieve that 'Host Info Tracker' module provides an API to update the host information dynamically to the host info table. This module retrieves the information from nodeinfo.xml. So whenever there is an update in the nodeinfo xml, this module has to be executed in the master node to update the host info table.

Nodeinfo xml as follows,

```
<node_info>
                        <nodes>
                                <node>
                                <hostname>node1</hostname>
                                <ipv4address>192.168.1.11</ipv4address>
                                <max_capacity>
                                        <cpu_core>8</cpu_core>
                                        <memory unit="KiB">32689796</memory>
                                        <io unit="KiB">1073741824</io>
                                </max_capacity>
                         <available_capacity>
                                        <cpu_core>8</cpu_core>
                                        <memory unit="KiB">32689796</memory>
                                        <io unit="KiB">1073741824</io>
                                </available_capacity>
                        </node>
                        <node>
                …
                …

</node_info>
```

Figure 5

2.4 Update Guest Information

Guest info table will be updated whenever a new guest is created on any host. Guest information will be added along with the information about the host on which the guest is to be created. VM_Info_Updater will be responsible for add/update of guest information.

3. Virtdc Command Line Interface

Virtdc is a wrapper CLI for virtual machine placement and scaling which provides a way to create, manage and monitor virtual machines effectively. Virtdc framework enables user to create virtual machines, maintain information about guest and host, terminate virtual machine and to handle static virtual machine placement.

The following are the core functionalities of the virtdc CLI:

usage: virtdc [-h] [-v]
{create,terminate,list,dominfo,hostinfo,force-
migrate,removehost,loadbalance,consolidate,addhost,getip,monitorgraph}

Option	Specification
create	create new domain from the base image
terminate	terminate running domain
list	list running domain
dominfo	domain information
hostinfo	host information
force-migrate	migrate domain from source host to dest host
removehost	removehost
loadbalance	loadbalance host
consolidate	consolidate host
addhost	add new host
getip	get domain ip
monitorgraph	monitor domain usage

for detailed usage, please refer to http://www.utdallas.edu/~dxa132330/virtdc.html

4. GUEST CREATION

Guests can be created from XML configuration files. Guest configuration can be copied from existing XML from previously created guests or use the dumpxml option. To create a guest with virsh from an XML file:

virsh create configuration_file.xml

Creating a virtual machine XML dump(configuration file)

The following libvirt API gives the configuration XML from the existing guest,

virsh dumpxml [domain-name]

virsh dumpxml base_guest > guestconfig.xml

guestconfig.xml will contain the configuration for the base guest. Virtdc uses this xml as the base guest to create new guest on host by tweaking the guestconfig xml based on the requirement.

Guestconfig.xml looks similar to the following,

```
<domain
type='kvm'id='2'>
              <name>vm_name</name>
              <uuid>vm_uuid</uuid>
              <memory unit='KiB'>max_memory</memory>
              <currentMemory unit='KiB'>current_memory</currentMemory>
              <vcpu placement='static' current='current_cpu'>max_cpu</vcpu>
              <resource>
                <partition>/machine</partition>
              </resource>
            …
            …
</domain>
```

Figure 6

vm_name, vm_uuid, max_memory, current_memory variables will be replaced based on the new guest requirement.

vm_submitjob() creates guest from the configuration xml and updates the information about the guest in virtdc API's

```
vm_submitjob(vmid,cpu,memory,io)
```

This API takes the vmid, cpu, memory, io as the parameters to create the guest machine.

Virtdc uses the base image and clones new guest from the base using the configuration file. It uses virsh tool to create guest,

```
virsh --connect qemu+ssh://"+host+"/system create new_guestconfig.xml
```

Host Identification for placement:

Host will be identified using VM decision maker. Virtdc provides an API to verify whether the guest can be created in any host.

```
is_space_available_for_vm(cpu, mem, io)
```

This API checks the availability of cpu, memory and io from the guest and host information table [node_dict].

5. GUEST MIGRATION

Guest can be migrated from one host to another. Placement manager will make the decision to migrate the guest. The following diagram illustrates the guest migration. Client host [figure 7] is the master node where the libvirt and virtdc API's are running.

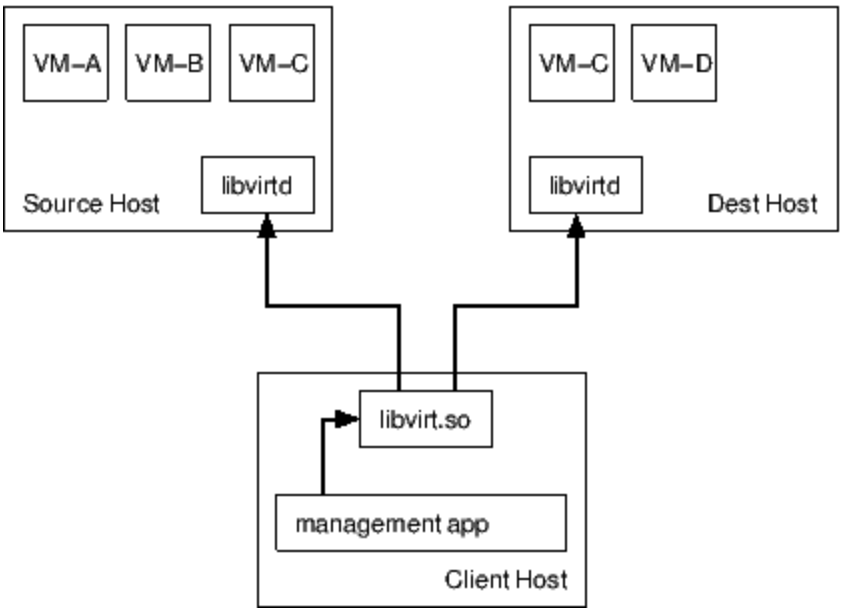


Figure 7 – ref: libvirt portal

All disk images of the guest will be stored in the network shared folder. Libvirt API migrates the currently running CPU processes and memory operations but the disk image of the guest will not be migrated. The guest images will be stored in the network file system (NFS). Even after the migration the guest will be accessing the image from source host.

Guest can be migrated under following scenarios —

1. Load balancing
2. Consolidation
3. Host removal

Migration can be achieved through libvirt API,

```
ssh -q -o StrictHostKeyChecking=no root@source_host "virsh migrate vmid
qemu+ssh://dest_host/system"
```

virtdc API for migration:

The following API in virtdc uses the vmid, source_host, dest_host to migrate the guest from the source host to the destination host. This uses the libvirt API to achieve the migration (as mentioned above).

```
initiateLiveMigration(vmid,source_host,dest_host)
```

6. GUEST SCALING

Virtdc provides API for cpu scaling and memory scaling. Scaling is not an user request process. Scaling decision will be retrieved from the placement manager and the client SLA configuration.

CPU scaling:

Virtdc API for cpu scaling is as follows,

```
initiateVMCPUScaleUp(hostname, vmid, cpu)
```

virtdc API internally uses libvirt to accomplish the cpu scaling,

```
ssh -q -o StrictHostKeyChecking=no root@sourcenode "virsh migrate vm_id
qemu+ssh://dest_node/system"
```

Memory Scaling:

Virtdc API for memory scaling,

```
initiateMemScaleUpOrDown(hostName, vmID, memorySize)
```

libvirt API for memory scaling,

```
ssh -q -o StrictHostKeyChecking=no root@'+hostName+' virsh setmem '+vmID+' '+memorySize
```

Source code for virtdc:

https://github.com/dcsolvere/virtdc

7. GOOGLE DATA SIMULATION

Parsing and scaling 40Gb size data

We parsed the Google cloud data, and scaled it down to a reasonable size. We scaled the lifetime of guest vm down to 5 minutes so that it fit our demo.

Introducing randomness in vm's lifetime

We also introduced some randomness in vm's lifetime, such as vm's has 1% chance to reboot at some point of time of its lifetime. This is done by simulator.

Google workload is retrieved from Google cloud storage and parsed using csv parser. Each task is considered as a single guest and a job may have multiple tasks/multiple guests.

Figure 8 – Overview

Workflow diagram for Simulation:

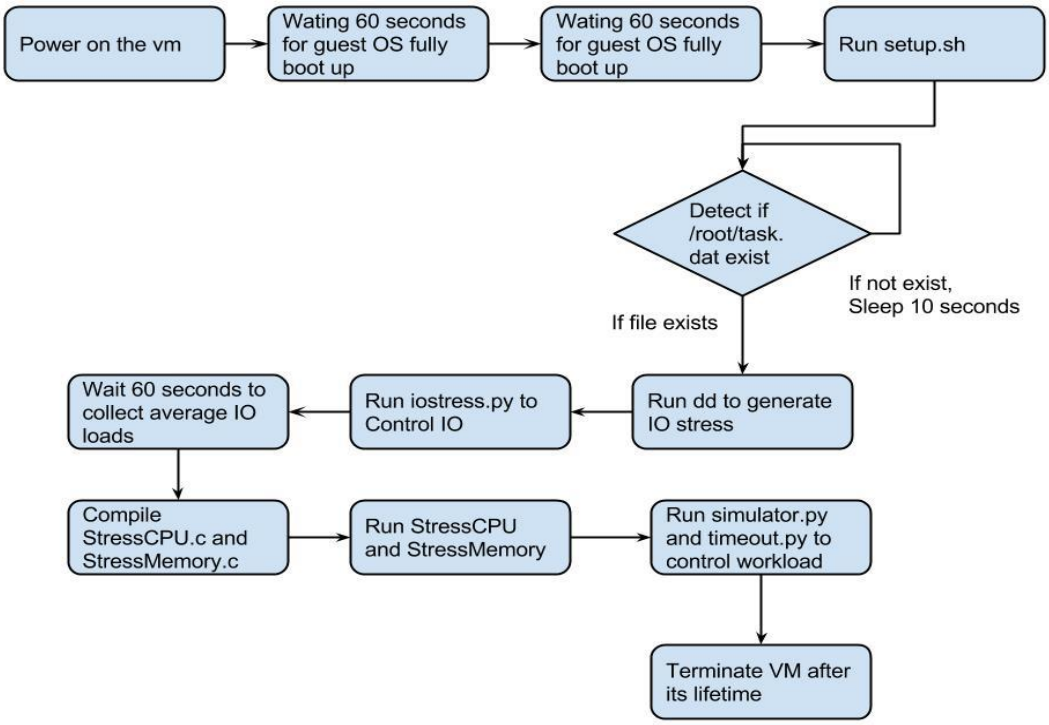


Figure 9 – Simulation workflow

The Simulation module initiates the following processes to simulate the workload parsed from Google data-

1. StressCPU : Keeps the CPU busy.
2. StressMemory : Keeps the memory busy by performing writes on it.
3. IOStress : Keeps the IO busy by continuously writing/reading disk.

All the above processes once started will be continuously monitored to check if it is overshooting the workload. Based on this data, the execution of the process will be controlled and made to simulate the given workload.

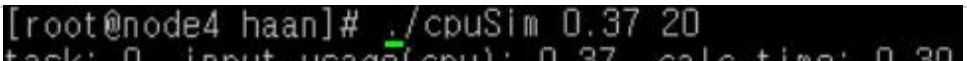
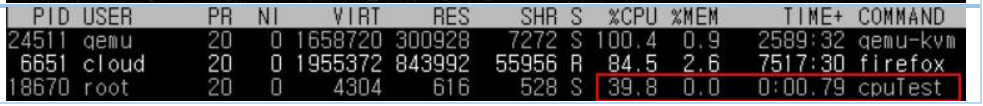
CPU simulation

We use c program to simulate CPU workloads. It takes two parameters. The first parameter is a CPU utilization and the second parameter is a time period. The input CPU utilization is from Google Trace Data that is parsed from parser.

This simulator goes one step further than other simulators that are using load average. This program can generate current CPU utilization while other simulators just run and sleep the program according to load average; Our simulator can simulate input usage that time period has less than 1 minute(minimum average time for load average), but other simulators based on load average are not able to the input usage. Moreover, for this reason, we can monitor current CPU utilization of each VM correctly.

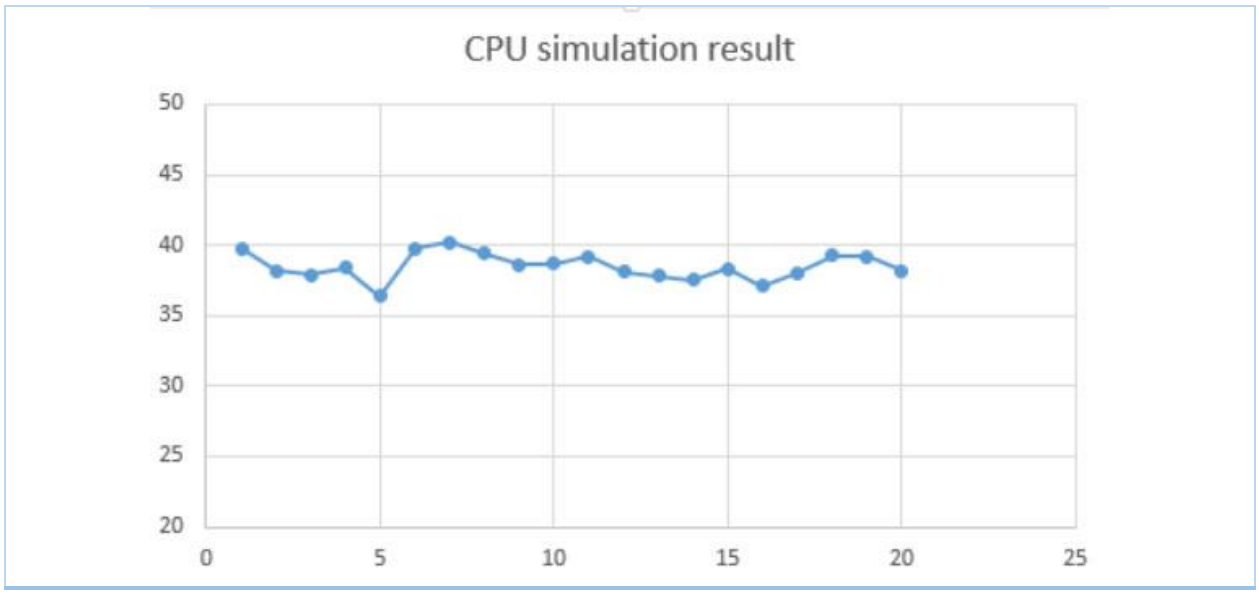
During the time period, the program simulates CPU utilization according to the input utilization. If workloads exceeds more than one processor, the simulator fork new process to enforce the workloads. Each process will terminate after some timeout.

1.Example of simulator

Input example. Utilization: 0.37	
Result on "top -d 1"	

The value of CPU utilization for example is 0.37. We can see that the program simulates CPU utilization according to the input and it is fluctuating near the input value 0.37.

2.Graph of the result

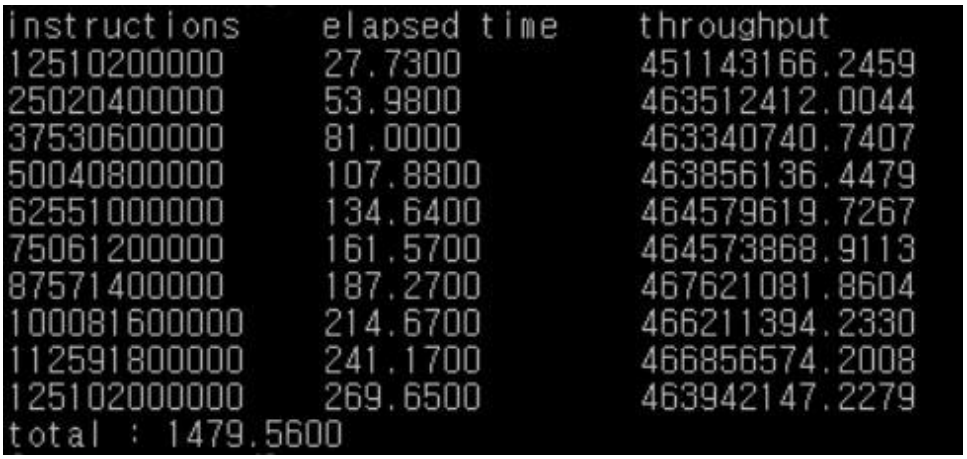


Simulation results during the time period is fluctuating and it is similar with real world program. We can also adjust the degree of fluctuating based on the input CPU usage to make it more realistic.

3.Approach

Our approach for CPU simulation is based on the number of instructions. We have figured out the number of instructions that our CPU can handle within in one second and have applied the number to our simulator. For example, if our CPU can handle 500,000,000 instructions and input utilization is 50%, we execute 250,000,000 instructions per second.

We decompose the type of instructions into four parts, CPU intensive, Memory intensive, Write intensive, and Read intensive. We have done experiments to figure out the number of instructions that our CPU can handle within one second. To figure out the number, we use matrix multiplication. For example, we have 10 rounds of experiments and we increase the size of matrix for every round to increase the number of instructions. Then, we have checked elapsed time for each round. By doing this, we can get throughput (the number of instructions / elapsed time), which means the number of instructions that our CPU can handle.



instructions	elapsed time	throughput
12510200000	27.7300	451143166.2459
25020400000	53.9800	463512412.0044
37530600000	81.0000	463340740.7407
50040800000	107.8800	463856136.4479
62551000000	134.6400	464579619.7267
75061200000	161.5700	464573868.9113
87571400000	187.2700	467621081.8604
100081600000	214.6700	466211394.2330
112591800000	241.1700	466856574.2008
125102000000	269.6500	463942147.2279
total : 1479.5600		

This picture is example of the experiments for CPU intensive. Column "instruction" shows the number of instruction of the round, "elapsed time" shows the elapsed time for the round, and the "throughput" shows the number of instructions divided by elapsed time. The throughput is the number that our CP U can handle within one second.

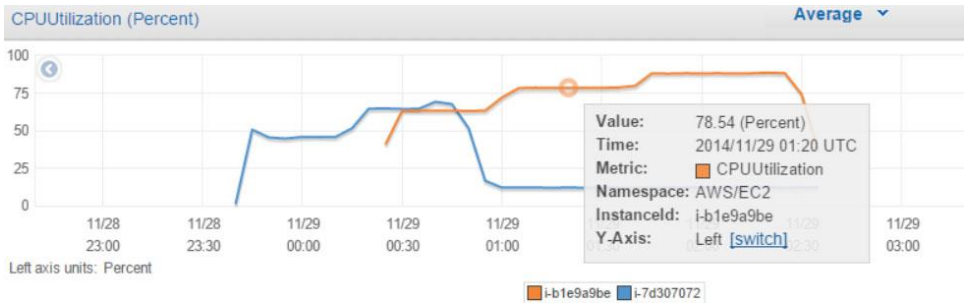
There are little fluctuations between each rounds and each types of instructions also has different number of instructions. We need to adjust the number of instructions based on our experiments. We have done it by heuristic way.

We made tax calculator that reads a file to get tax rates, calculates taxes, and write it to file. The program has all the four types of instructions, CPU, Memory, Read, and Write. After develop the program and we adjust the number of instructions to match it based on our experiments results by heuristic way.

4.Running our simulator on Amazon Web Service

Instance type	vCPUs	Memory(Gb)	Storage	Capacity
T2.Micro	1	1	EBS	2.5Ghz, Intel Xeon Family
M3.Medium	1	3.75	1x4 (SSD)	2.6Ghz, Intel Xeon E5-2670vs

We have tested our simulator on Amazon Web Service(AWS) using two EC2 instances. They are have different capacity, so they will have different number of instructions that they can handle in one second. We have executed same program on the two instances. The same number of instances are coded on the program. This is because we are expecting to see whether they are drawing similar curve.



This graph shows that our the number of instruction based approach for simulator is acceptable even in real Cloud environments. The two instances has different capacity, so absolute number for the graph should different, but curve should be same. The graph shows that they are drawing similar curve. Because of time constraint in this semester, we do not have time to figure out the exact number of instructions that each instances can handle within one second and it is out of range of our project purpose. Important thing the graph shows is our approach is acceptable and working on real Cloud environments.

Memory simulation

In memory simulation, the program keeps writing data into memory repeatedly. We write the process id into a pid file and use that file to keep track of process id.

IO simulation

In IO simulation, we use linux dommand 'dd' to write/read to/from disk. We use iostat to keep track of real-time io usage, and calculate the average usage during past one minute. We also store the process id of io simulation in a pid file.

Workload format

The workloads configuration file is written as the following format:

<time><cpu_usage> <memory_usage><io_usage>

And the very last two lines are the maximum cpu usage and memory usage, with all data are being properly ceil or floor.

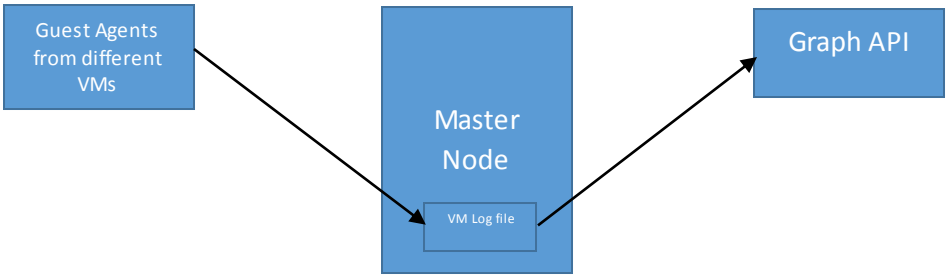
8. MONITORING – VMONERE API

The monitoring setup involves creating and installing an agent in the Guest, which reports to the Master node about its current status through socket communication.

For this purpose, a monitoring agent has been placed in the base image. Due to this, every Guest that is created has an agent running when they boot up.

The sequence of actions can be listed as following-

1. Guest boots up and the monitoring agent starts.
2. The monitoring agent takes the CPU, Memory and IO status of the Guest.
3. A log file is generated in the Master node for the VM at /var/lib/virtdc/vmonere/monitor_logs/<VM ID>.log.
4. The monitoring agent appends the status report into the log.
5. The monitoring agent retrieves and appends the status every 5 seconds (configurable in code).
6. Every Guest has its own log file in the folder.



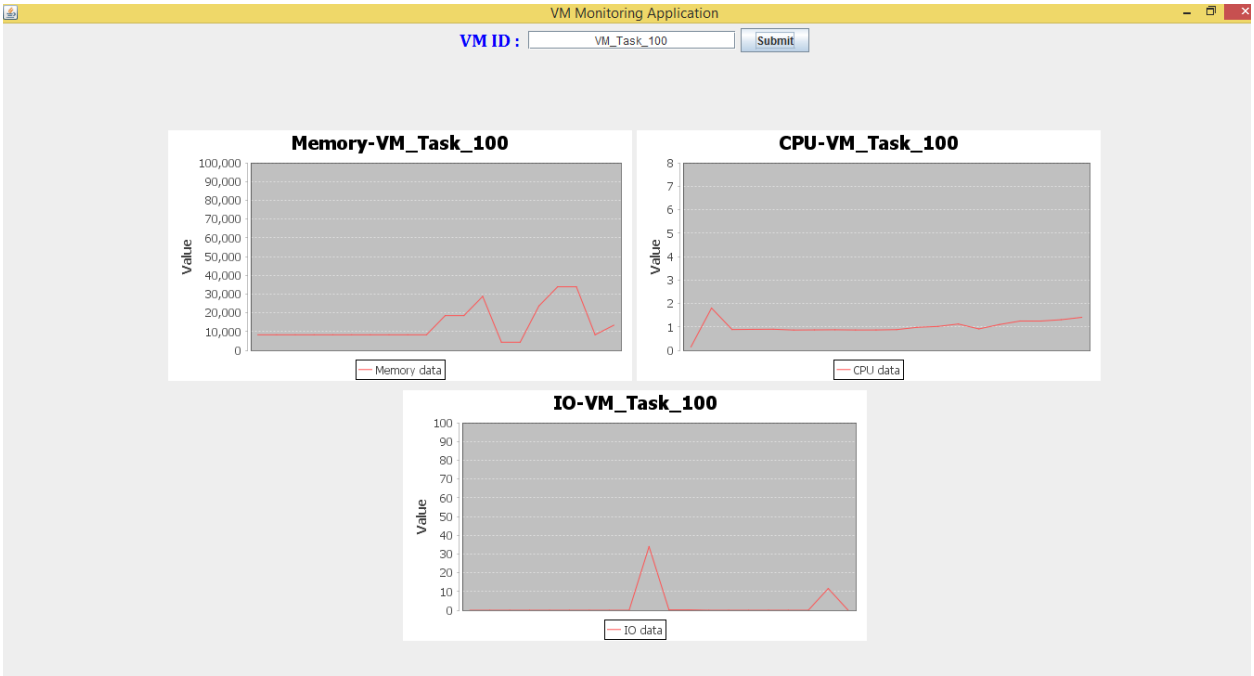
A sample of the guest information stored in the log file is given below-

```
      TIMESTAMP                    VM ID        CPU     OS        MEMORY    IO
2014-12-01 23:28:34.498171   | VM_Task_100  | 1.405| 160768.0 | 8260   | 0.00
2014-12-01 23:28:40.964723   | VM_Task_100  | 1.036| 163840.0 | 13380  | 20.59
2014-12-01 23:28:47.435615   | VM_Task_100  | 0.839| 149504.0 | 13380  | 29.89
2014-12-01 23:28:53.908183   | VM_Task_100  | 0.847| 149504.0 | 13380  | 100.10
```

Once the VM is terminated after the tasks are over, the log files are automatically deleted after 24 hours, to avoid any space issues and incorrect representation of the data.

As the data in log file may not give a clear picture of the load on the Guest, the monitoring API also includes a real-time graphical representation of the data that is written in log.

The figure below shows a sample of the graphical output for the VM with VM ID - VM_Task_100.



The graphical output is an API developed using JFreechart.

The VMonere graph application can be started with the following command:

```
virtdc monitorgraph
```

The panel displays a textfield for entering the VM ID, for which the status needs to be tracked. If the VM ID entered is incorrect or the log file for the VM is yet to be generated, then appropriate error message is displayed. Once the correct VM ID is submitted, the panel will show the real-time moving graphs for CPU, Memory and IO.

For real-time analysis, the graph will start with the latest data that is appended in the log file. The graph moves after every 5 seconds- this is in sync with the 5 second duration for retrieving and appending of status data in the log file.

The monitoring data is used by the Decision maker for getting the current status of the Guest. The decision maker keeps tab on the status of the guest and the host and based on the load, it makes a decision on whether the VM needs to be migrated or scaled up/down.

This action is performed by the decision maker job running in Master node.

Possible enhancements in Monitoring API:

- Represent host CPU, Memory and IO status as graphs.

- In graph, include the input task data for direct comparison with the current status of the VM.

- Make the graph API available as a real-time graph on a webpage for monitoring the status without connecting to the Master or any other nodes in the network.

9. Trouble Shooting

Booting up agents in guest

When we were trying to boot up the agents in guest OS, we found that the agent will exit after writing very first few lines of log. This is because there is an index-out-of-bound exception occurred on the remote end. This took us several days to debug.

10. Setup Node

When doing setup of a new node, like adding new node in cluster. What we need to do is to:

1. Configure network
2. Configure ssh with all other nodes in the cluster
3. Configure nfs

We use setup.py to setup the configuration on physical nodes and it passed all tests.

11. Problems encountered

Issue	Details
Kernel bug in CentOS 7 versioned prior to 3.15 (ours are 3.10)	https://bugs.freedesktop.org/show_bug.cgi?id=62411 Workaround: - http://bugs.centos.org/view.php?id=7177
Gnome env has conflict with python pexpect module	not able to send carriage return
vm_migration needs shared storage	config NFS server on master node as shared storage pool
cpu hot unplug is not implemented in kvm	- https://access.redhat.com/documentation/en-US/Red_Hat_Enterprise_Linux/6/html/6.3_Release_Notes/virtualization.html
ssh connection stuck for newly created machine	Fix: - change guest creation template to honor MAC change - create local agent on guest, start on guest boot and report monitor data to host
iostat reports %util greater than 100%	https://bugzilla.redhat.com/show_bug.cgi?id=445676

Green - solved **Orange - no impact** **Red - still exist**

12. IMPACTED FILES

12.1. Simulation

No.	Description	Related files
1	Parse Google Cloud Workloads	parse.py
2	Running Initial Script	setup.sh
3	Generating IO workload	iostress.py
4	Compile cpuSim.c and StressMemory.c	cpuSim.c stressMemory.c
5	Running StressCPU, StressMemory, iostress. simulator.py is the main coordinator to control the CPU and memory load and io workload.	simulator.py timeout.py

12.2. VM framework

No.	Description	Related files
1	Setup environments	bridge_setup.py SSHPasswordless.sh nfs_master_setup.sh nfs_client_setup.sh SSHSetUp.py
2	Activate a monitoring system	vmonere_listener.py vmonere_start_monitor.py vmonere_agent.py vmonere_utility.py
3	Implement an API to accept VM placement change decisions (VM_change_placement)	VM_PlacementManager.py
4	(VM_submitJob) Extract job information for placement decision	VM_PlacementManager.py
5	(VM_submitJob) Invoke (place_newJob) to get a placement decision	VM_RunJob.py VM_Framework_Utility.py
6	(VM_submitJob) Create the VMs at the corresponding host	VM_submitJob.py virtdc.py
7	Implement a mechanism to notify the client and placement manager the (job_termination_event)	VM_terminationList.py
8	Implement an API to report the results of a job execution (VM_jobReport)	VMMemoryOverUsageInfo.py
9	Implement an API for VM workload data provisioning (VM_workload)	VM_RunJob.py
10	Keep logs to record the job execution characteristics, the placement decisions and changes, etc. for performance analysis of the VM placement algorithm	VM_Monitor.py VM_Monitor_Utility.py

12.3. VM placement manager

No.	Description	Related files
1	Implement the placement manager based on the VM placement, scaling, and migration decision algorithm	VM_PlacementManager.py
2	Invoke (VM_workload) periodically to obtain VM workload data for all VMs	vmonere_listener.py vmonere_start_monitor.py vmonere_agent.py vmonere_utility.py
3	Decide whether there should be VM scaling or VM migration	VM_PlacementManager.py
4	Invoke (VM_change_placement) to give the decision and activate changes	VM_PlacementManager.py
5	Implement an API to give placement decision for a new job (place_newJob)	VM_submitJob.py
6	Analyze and decide where to place the VMs	VM_decisionMaker.py
7	Return the placement decision to VM framework	VM_decisionMaker.py
8	(job_termination_event) When the event is triggered, decide whether there should be VM scaling or migration, if so, make the decision and invoke (VM_change_placement) to make placement changes	VM_terminationList.py VM_decision_algorithm.py

12.4. Monitoring Graph files

No.	Description	Related files
1	Main panel for graphs	MainPanel.java
2	Monitors CPU load from log file	CPUMonitoringGraph.java
3	Monitors Memory load from log file	MemoryMonitoringGraph.java
4	Monitors IO load from log file	IOMonitoringGraph.java

13.REFERENCES

[1] On Theory of VM Placement: Anomalies in Existing Methodologies and Their Mitigation Using a Novel Vector Based Approach Appendix I: Process Details

[2] On Resource Management for Cloud Users: A Generalized Kelly Mechanism Approach

[3] SLA-aware virtual resource management for cloud infrastructures

[4] VMware Distributed Resource Management: Design, Implementation, and Lessons Learned

[5] Q-Clouds: Managing Performance Interference Effects for QoS-Aware Clouds

14. Individual Contribution

Name	Work
Haan Mo Johng	1.Framkework Infrastructure setup and practice.(OS, KVM, bridge networking, Libvirt) Developed scripts to automate bridge-network setup for 4 nodes. Bug fixing for scripts for bridge-network automation. Investigate about SSH delaying. 2.VM placement Develop scripts for VM placement and communication between VM and Host using IP. 3.Monitoring Investigate about Monitoring tools. Develop monitoring scripts for CPU and Memory. Update monitoring scripts to add IO monitoring. Update monitoring scripts with timeout. Bug fixing for monitoring. Update monitoring method to get current usage instead of load average. Develop scripts to delete monitoring log periodically (24 hours). 4.Simulator Investigate about CPU monitoring tools. Design new method based on the number of instructions for CPU simulation to generate current CPU utilization. Develop and experiment the design in our machine. Develop simulation program running on Amazon Web Service using EC2 Test our CPU simulator on Amazon Web Service. Update the CPU simulator to include four types of instructions, CPU-intensive, Memory-Intensive, Read-intensive, and Write-Intensive. 5.Decision manager Reading a paper, Q-Clouds: Managing Performance Interference Effects for QoS-Aware Clouds Investigate about static decision algorithm using web-based service, the number of instruction based approach, and genetic algorithm. Investigate decision algorithm in terms of performance and cost using web-based service and genetic algorithm.
Ruiyu Wang	1. Infrastructure Setup • Project infrastructure and environment setup for all 4 nodes(OS, KVM, NFS, firewall, bridge networking, passwordless ssh, etc.) • Created and maintaining original guest base image that all guests would be cloned from. Tuning OS parameters for the base image so that all vm's can successfully boot and run without confliction. 2. Investigation • slow responding ssh connection to newly created guest

	• CPU/MEM scaling, successfully did online CPU adding, MEM adding/removing. Found out CPU online deleting is not supported by libvirt yet • Initial Investigation of various monitoring tools and drafted an investigation report: i. https://docs.google.com/spreadsheets/d/1tozKsq1nyYWH-wM0lLvZvM8CXmF0hTeYHST1OqBiA0A/edit#gid=1225566833 3. Simulation • Write IO part of the simulator, which use linux 'dd' command to continuously read / write file on disk. • Work with Haan in creating a more accurate CPU simulation 4. Framework • Proposed ways to do monitoring for CPU, MEM & IO. For CPU read OS average load in /proc/loadavg. For MEM read simulator task mem usage in /proc/pid/status. For IO, use iostat command to fetch disk utilization • Contribute code in cpu scaling • Contribute code for cpu/men/io monitoring api • Contribute getGuestIP() function to fetch guest ip through virsh console(serial console) using Python pexpect module • Code review, testing and bug fixing of the whole project (~70%) 5. Decision Manager • work with Dinesh in implementing algorithm for vm load balancing and consolidation
Dinesh Appavoo	Virtdc - Simulation: 1. Developed an API to to parse the csv max_memory, max_cpu, max_io data and call the vm_submit_job API to create the domain. 2. Implemented vm_run_job API to send the workload data to the domain and start the simulation process once after the domain creation virtdc - Framework: 1. Developed vm_submit_job API by cloning new guest from the base guest. 1.1. In this maintaining a configuration for each guest based on CPU , Memory and IO parameters. 2. Developed host_info_tracker API which parses the nodeinfo.xml and stores the information in python dictionary. This XML can be updated anytime by the users on new host/resources addition. host_info_tracker will take care of this and the new resource will be available for future domains. 3. Developed guest_info_tracker API to maintain the domain information(ID, CPU, mem, io,start-time, host) along with a host in python dictionary. This will get updated on domain creation/scaling/migration/termination 4. Developed vm_decision_maker API by implemeting a static placement algorithm (Based on 'Planar hexagon paper ') to identify space for the guest 5. Designed a framework to maintain host information and this will get updated

	on guest creation. Maintaining tables to store the current host availability data 6. Developed vm_cpu_scaling API to scale-up/scale-down the CPU based on the initial allotted CPU and availability. Scaling up cannot be done above the maximum limit. And libvirt does not support CPU live scale down functionality 7. Developed vm_mem_scaling API to scale-up/scale-down memory until the max/min limit. 8. Implemented vm_placement manager framework to process the vm_monitor API reports. Developed simple algorithms to make the decision for CPU/Memory scaling considering the threshold given by the user in the SLA (SLA for threshold is hardcoded for now. Later it can be taken as XML input) 9.Developed vm_migration API to migrate the guest from the source host to the destination host. Disk image will still be in source host since the path is shared through NFS. 10. Developed mail api to send mails to support team and users on exception and abnormal usage 11. Created virtdc command line utility to produce the command line functionalities 12. Developed virtdc command using argparse to provide various command options Vmonere - Monitor: 1. Created vmonere_socket_sender api to report the usage from the domain to the host periodically. 2. Created vmonere_socket_listener api to recieve the domain usage information from the vmonere_socket_sender api from every domain and store information in json and text file format in master host. 3. Developed report_usage_on_hotspot api in vmonere-monitor to report the usage to placement manager after detecting hotspot in the domain virtdc - Manager: 1. Developed placement_manager api to perform action on hotspot detection and call the decision maker api. 2. Implemented an algorithm to process the usage and call any of the following api's after decision making i. CPU scaling api ii. Memory scaling api iii. vm_migraton api
Qinghao Dai	1. Framework · Parsed all 40Gb Google cloud data and analyzed the data's distribution. Scaled down the data to a reasonable size so that vm guest can be running for demo. (5 minutes) · Introduced randomness to workloads configuration file to simulate vm guest's crash. · Worked on the initial implementation of simulator and created a timeout utility for running any programs on base image. 2. Simulation · Developed the format of workload configuration file and apply it into

	simulator. · Coordinate with Dinesh on submitting new task job. · Integrate programs developed by Haan, Rahul, Ryan into simulator to simulate CPU, MEM, IO · Investigated bash_profile and crontab job to be running on base image 3. VM Placement Framework · Coordinate with Dinesh to make sure that agent can be running on base image properly. · Test Setup.py on physical node to make sure that new physical node can be easily configured when adding . 4. Decision Maker Trouble shooting on booting up agent on base image, so that agent can send monitoring data back to host properly
Rahul Nair	1. Framework • Infrastructure setup and practice.(OS, KVM, bridge networking, Libvirt) • Developed and automated SSH passwordless communication between hosts, so that introducing new hosts/resources will not require any manual work • Work with team to investigate framework issues like SSH delaying, connectivity issues etc. • Initial work on creating an automation script for setting up the whole environment with NFS, bridge networking and SSH communication. (Finished by Qinghao). 2. Simulation • Developed initial simulation framework for CPU stress. • Developed the simulation framework for Memory stress. • Based on Professor's feedback, investigated other better and possible ways to stress CPU. • Test the Memory stress simulator and improve the same. • Coordinate with Qinghao for the integration of all simulation programs. (Owner- Qinghao). 3. VM Placement Framework • Coordinate with Dinesh in implementation of vm_placement manager framework and helping in algorithm development. (Owner–Dinesh). • Coordinating and providing inputs in the implementation of dynamic VM placement algorithm. • Developed simulate_google_data API to send upcoming guest to vm_creation and maintain a queue for the remaining guests in the waiting list. • Created the driver program for the job processing to pass the parsed csv input data to the vm_submit_job. 4. Monitoring • Create agents for monitoring in Guest image which reports to master about the current status of CPU, Memory and IO.

	• Modify monitoring scripts to include log frequency and integration with the Guest monitoring agents. • Investigated and worked on configuring different monitoring tools like Munin etc. • Developed own VMonere API for generating real-time graphical representation of the monitoring data. • Improved the VMonere API based on Professor's feedback and currently working on enhancing it further to include the data for Hosts monitoring data. • Worked on web based real-time representation for the monitoring data for external monitoring. • Investigate bugs with monitoring logs and fix the same. 5. Decision Manager • Coordinate with Dinesh for the VM Decision maker algorithm and implementation, and providing help with the inclusion monitoring data into the decision making process. (Owner- Dinesh). • Implemented vm_termination API to handle Vm termination. This involved keeping track and identifying the VMs who have reached their lifetime and based on that destroy the VM if required and update the guest info table. • Coordinate with Dinesh in the development of vm_migration API to live migrate the VM from source host to destination host. (Owner- Dinesh)