IAAS Cloud Middleware

Midterm Project Report

Cloud Computing

To the

The University Of Texas at Dallas

Submitted by:

Anisha Nainani

Harsh Desai

Kanav Kaul

Sunish Sheth

Team Label : A11

The University of Texas at Dallas

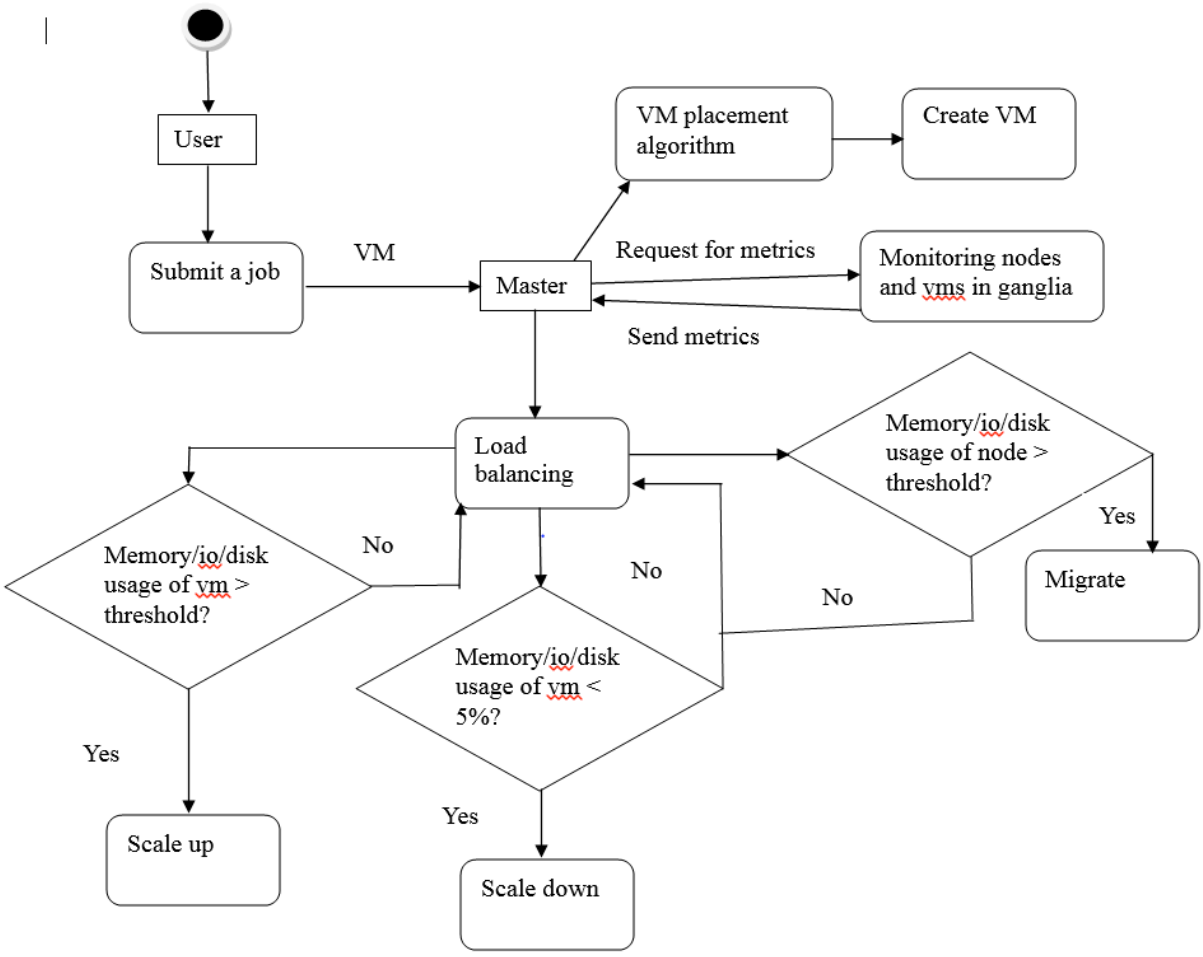
Dallas – 75252

December, 2015

Introduction

The IAAS model we propose creates a cloud environment with one master node and 5 shared slave nodes allowing users to submit their jobs, migrate their jobs and scale their jobs. An active load balancing algorithm is running constantly in the back end and is monitoring the load of all the slaves as well as itself and performing migration and scaling. A simple resource management solution is determined for placing the Virtual Machine at what events. A simple cloud environment is set up where KVM is used and interface for job submission and scaling is defined. Libvirt is explored to achieve Virtual Machine creation, scaling and migration. Virtual Machine Placement and Migration decision algorithms are designed. Ganglia is chosen as the monitoring system to observe various behaviors of the system to continuously display memory load, CPU load and IO load, etc. We do this with the help of many Virtual Machine Placement papers.

Activity Diagram



Architecture Diagram

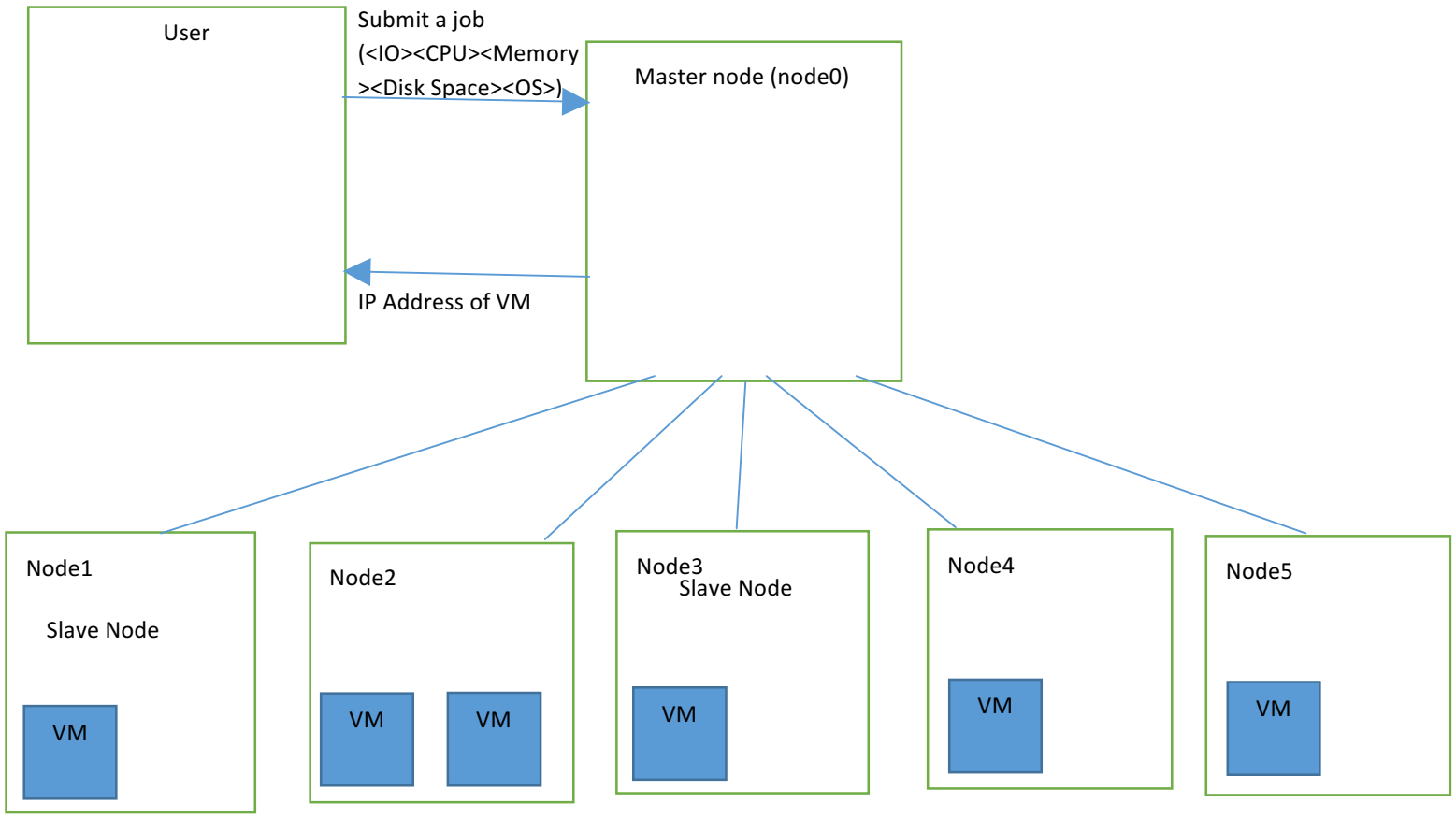


Fig1.1: Contains the architecture of the system.

Description of nodes in the architecture

There are 6 nodes in our cluster each having 32 GB RAM and 1 TB Hard disk. Each node consists of a static ip address and a bridge connection between node's Ethernet interface and vm's Ethernet interface. The master node makes decision for load balancing, migration, scaling up and down by continuously monitoring the nodes in the cluster. Following are the nodes in our cluster:

Node	IP Address
Node0 (master)	192.168.1.10
Node1	192.168.1.20
Node2	192.168.1.30
Node4	192.168.1.40
Node5	192.168.1.50

Detailed design

User Request:

User can request to create a VM by passing in their requirements. Requirements include IO, CPU, Memory, Disk Space required and OS type. Once the User request is received by the Master, it runs the load balancing algorithm to check which physical machine is best suited to place the VM. Once that decision is made, a createvm command is executed and the VM is placed in some specified node. The IP address of the VM is send back as the response to the user.

Once a new VM is created a hashmap is updated which contains all the details of a particular user and the hashmap is written to a file. HashMap<key, value>, where key is the username of the user and value is an ArrayList of all the configurations and the details of the user. For example, each row for the user will contain, vm_name, IP Address, node, Current CPU, Memory, Disk space, I/O configurations, Snapshots names and its location.

API created for this component includes the following scripts:

creatvm.sh

createexistingvm.sh

startvm.sh

addHost.sh

network_config.sh

getips.sh

Command :

- python main.py createvm username vm_name memory disk os cpu io ip_address
- python main.py createexistingvm username vm_name memory disk os cpu io

  ip_address img_file job_type

Load Balancing:

Nodes	I/O	CPU	Memory	Disk
Node 1	High	Low	Medium	High
Node 2	Low	High	High	Medium
Node 3	Medium	High	Low	Low
Node 4	Low	High	Medium	Low
Node 5	Medium	Low	Low	Low

Table 1.1: Contains the usage of different nodes.

Explanation of Table: Example, Disk of node 1 is high. This means that the Disk usage of node 1 is high. This implies it is not good to put more load on disk of node 1.

Decision for categorizing for High, Medium and Low is as follows:
Usage <= 30% -> Low
30> Usage <=70% -> Low
70> Usage <= 95% -> Low
Above 95%, the Node can no more handle any requests, try for migrations or scale out methods to reduce node load.
Now when a new request arrives for performing load balancing, which includes the following parameters: I/O, CPU, Memory and Disk Space. We then first get the current status of each node in the system using the monitoring system (Ganglia). Once we get the data from the monitoring system, we update the load balancing table (Table 1.1).

Later on, using a smart algorithm, we can decide whether which node is best suited for the new request. Once the decision is made, the VM is placed in the specified physical machine.

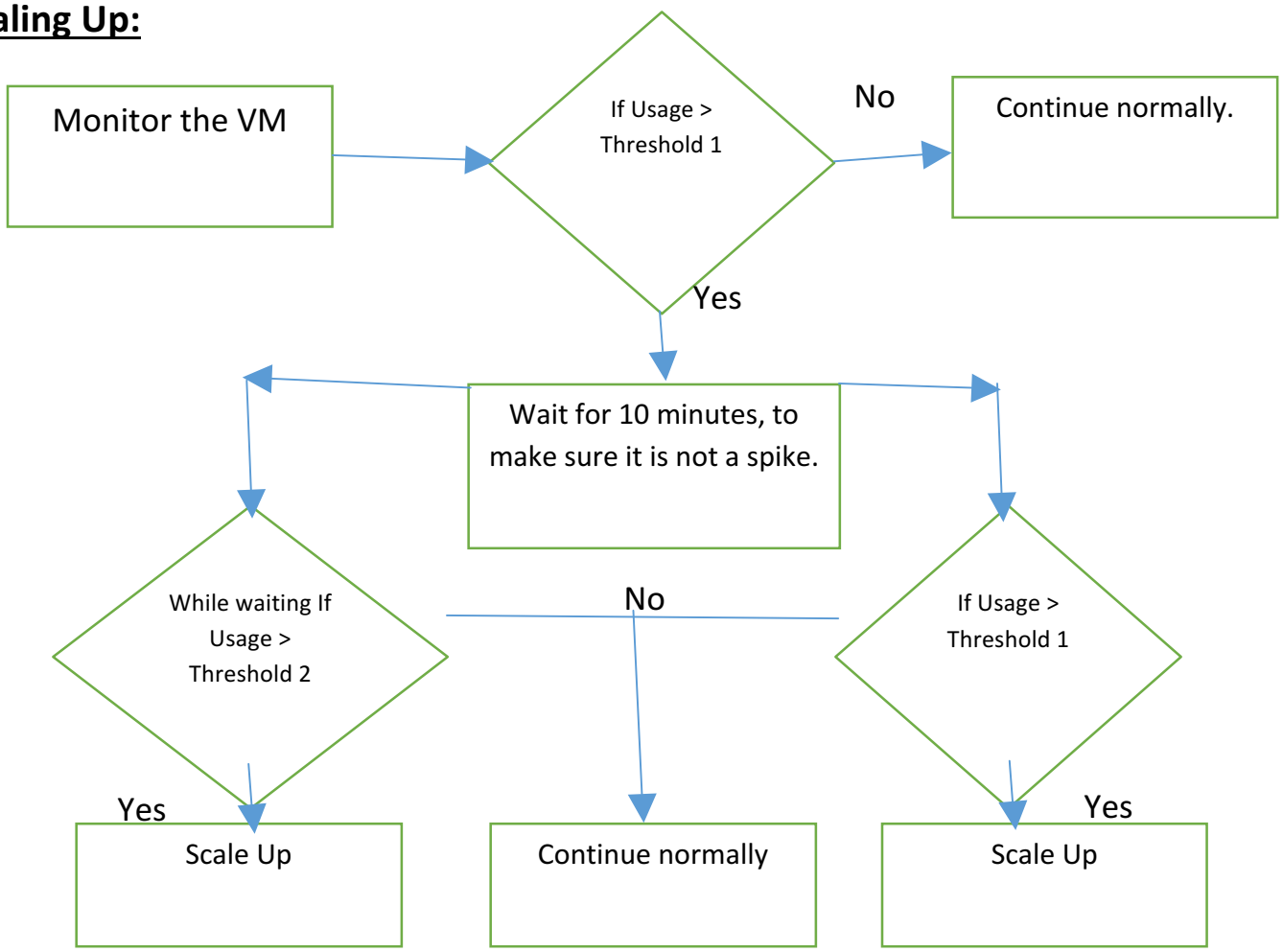
Algorithm:

1. Get the current status of nodes.
2. Get the VM requirements.
3. Add the VM requirements with the current load of all the nodes.
4. Place all the nodes into a priority list.
5. The node with the highest priority will get the VM.

API created for this component includes the following command:

- python main.py loadbalance vm_name i/o memory disk cpu

Scaling Up:

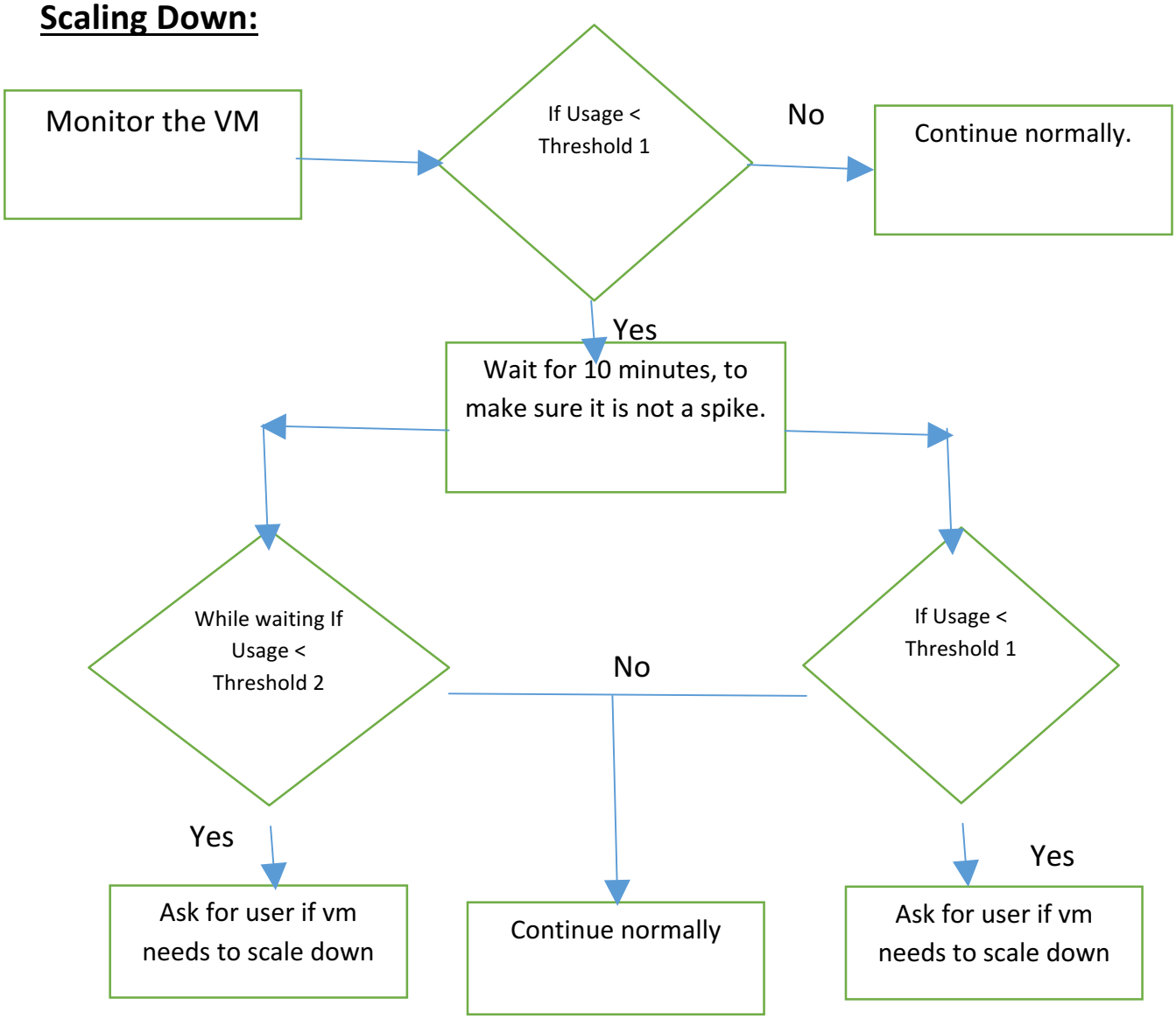


While scaling we monitor the VM and check if its usage has not increased above a certain level. For example, if the VM is allotted, 500GB disk size. If it is continuously using nearly 450GB then we might assume that the VM requires more disk space and it is thus, time to scale up if it can or else migrate and scale up. For this process, we take into consideration 2 thresholds. If any of the usage limit crosses a Threshold Mark 1(80% mark), then we wait for 10 minutes just to be sure if it is not a spike. If the usage still is high, then it's time to scale up the memory, disk or CPU. If while waiting if the usage still peaks up to threshold 2 (90% mark), then we scale up immediately. If none of these things happen then we function normally. We should ask the user before we start the process of scaling up to ask the limit of scale up.

API created for this component includes the following command:

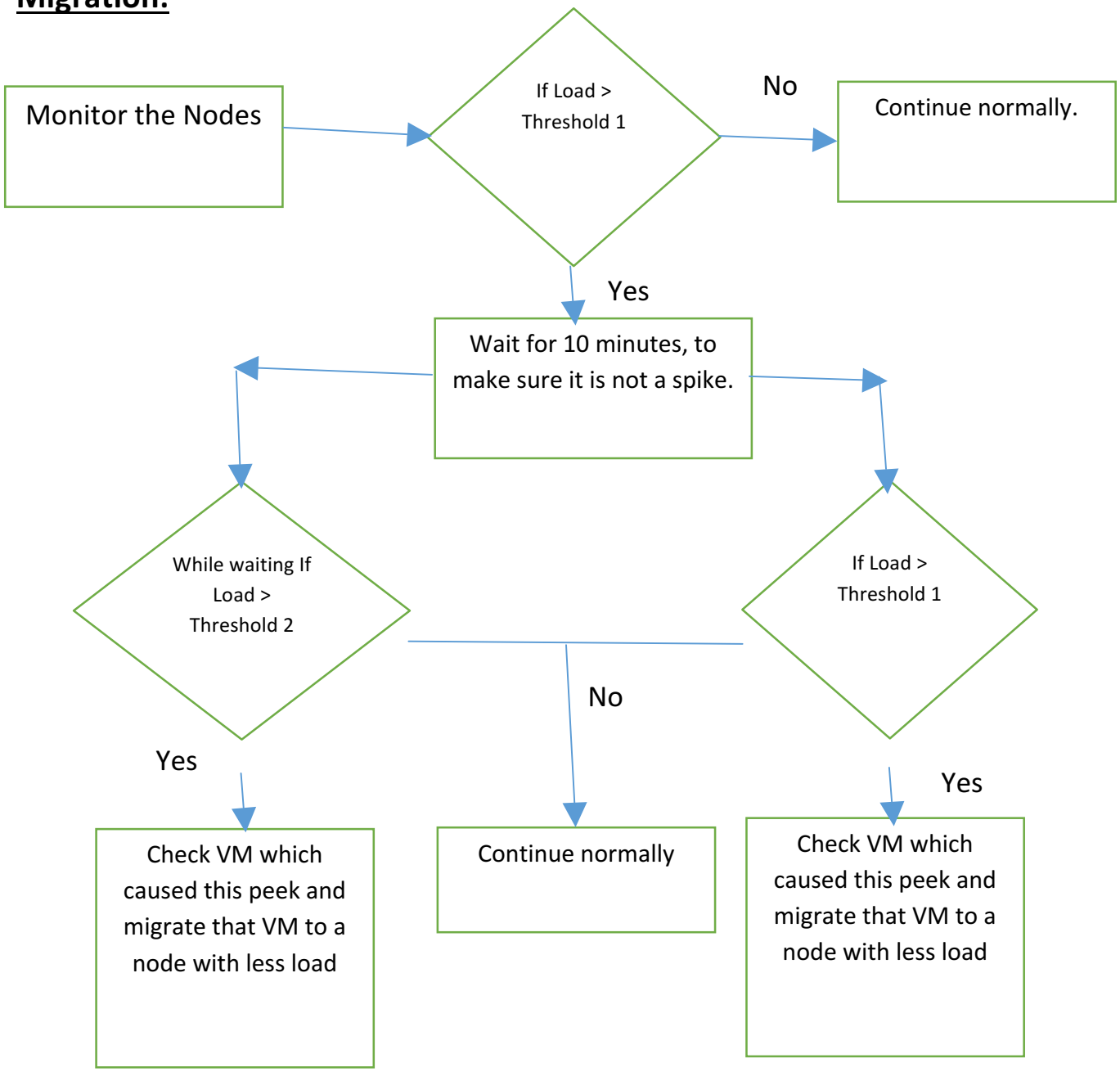
- python main.py scaleup vm_name size

Scaling Down:



While scaling we monitor the VM and check if its usage has not decreased below a certain level. For example, if the VM is allotted, 500GB disk size. If it is continuously using nearly 50GB then we might assume that the VM requires less disk space and it is thus, time to scale down. For this process, we take into consideration 2 thresholds. If any of the usage limit reduces to a Threshold Mark 1(10% mark), then we wait for 10 minutes just to be sure if it is not a spike. If the usage still is low, then its time to scale down the memory, disk or CPU. If while waiting if the usage still peaks up to threshold 2 (0% mark), then we scale down immediately. If none of these things happen then we function normally. We should ask the user before we start the process of scaling down to ask the limit of scale down.

Migration:



While migrating we monitor the node and check if its usage has not increased above a certain level. For example, if the node has allotted, 32GB memory. If it is continuously using nearly 30GB then we might assume that the VMs in the node might require more memory and it is thus, time to migrate a VM to a new Node using the Load Balancing algorithm. For this process, we take into consideration 2 thresholds. If any of the usage limit crosses a Threshold Mark 1(80% mark), then we wait for 10 minutes just to be sure if it is not a spike. If the usage still is high, then its time to migrate the VM. If while waiting if the usage still peaks up to threshold 2

(90% mark), then we migrate a VM immediately. If none of these things happen then we function normally. We should ask the user before we start the process of migration.

To migrate we need to check which VM caused the load to peak. We can do this by monitoring the VM data. Once the VM has been identified, we migrate the VM to another node with less load.

API created for this component includes the following script:

migrate.sh

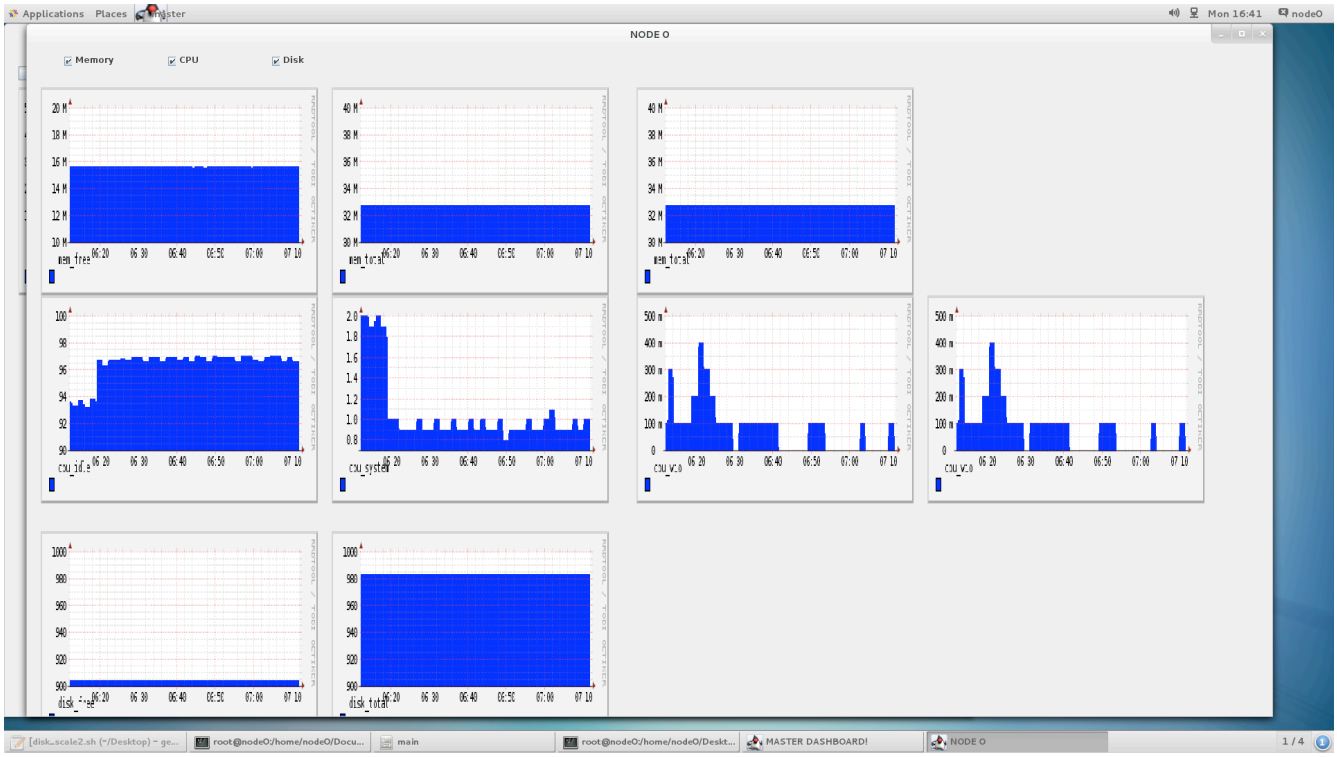
Command : python main.py migrate from_node to_node vm_name

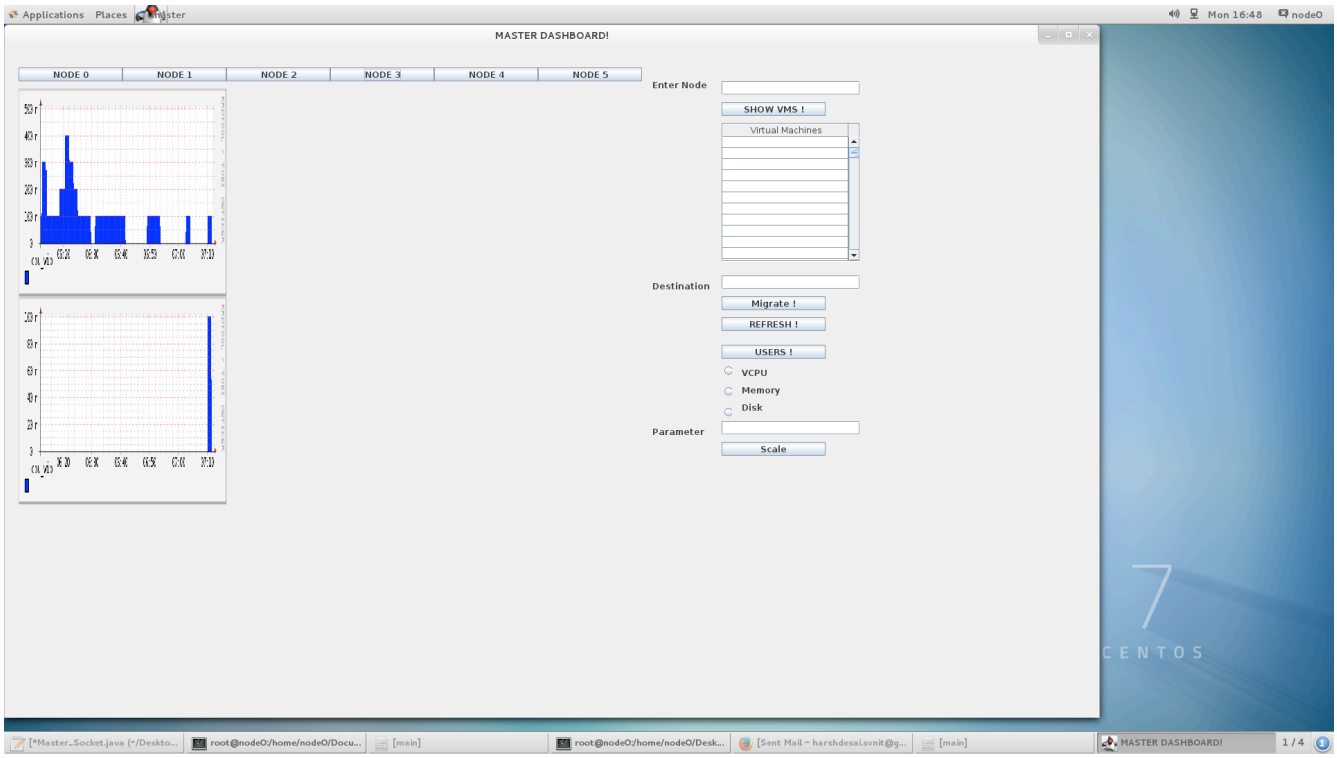
User Interface Design

Master User Interface

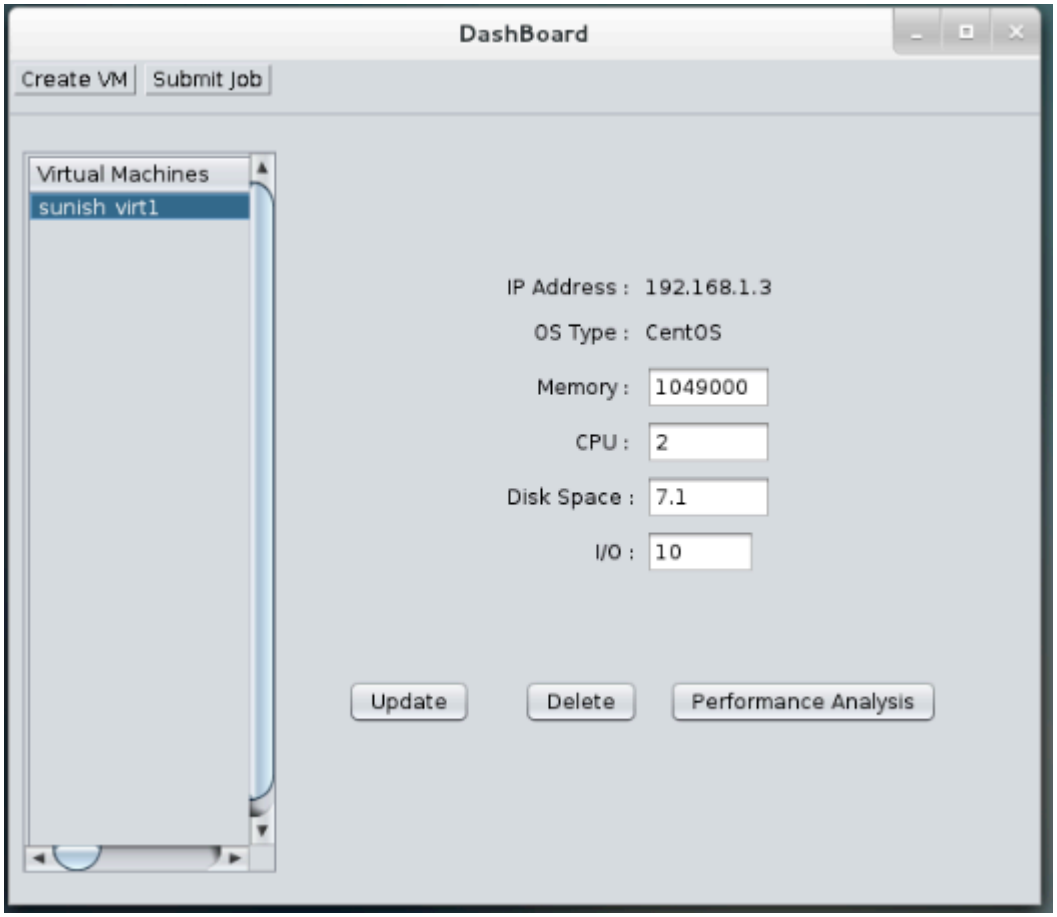
The User Interface is implemented in Java. Using the User Interface, we can migrate any virtual machine from one physical host to another. We can also scale the disk, number of vcpus and memory of each virtual machine from the master UI. The load of each physical host can be shown on the master UI when you click on any node or virtual machine.

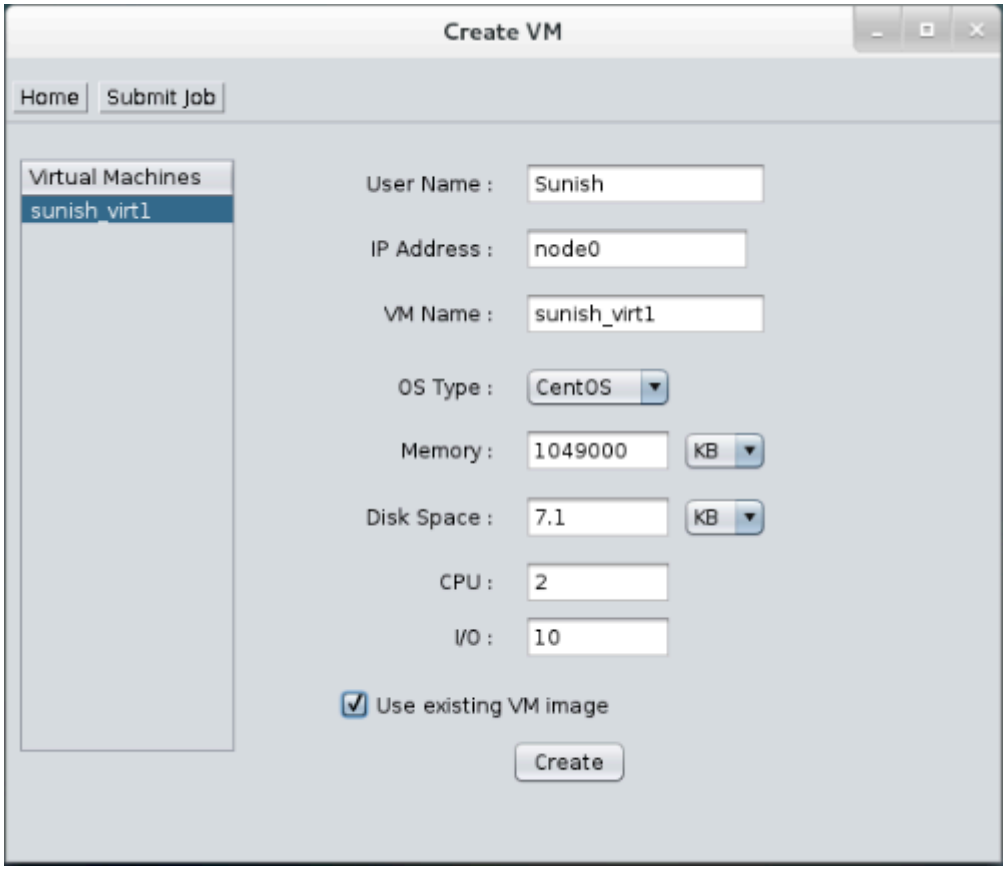
The following are the screenshots of the master UI :

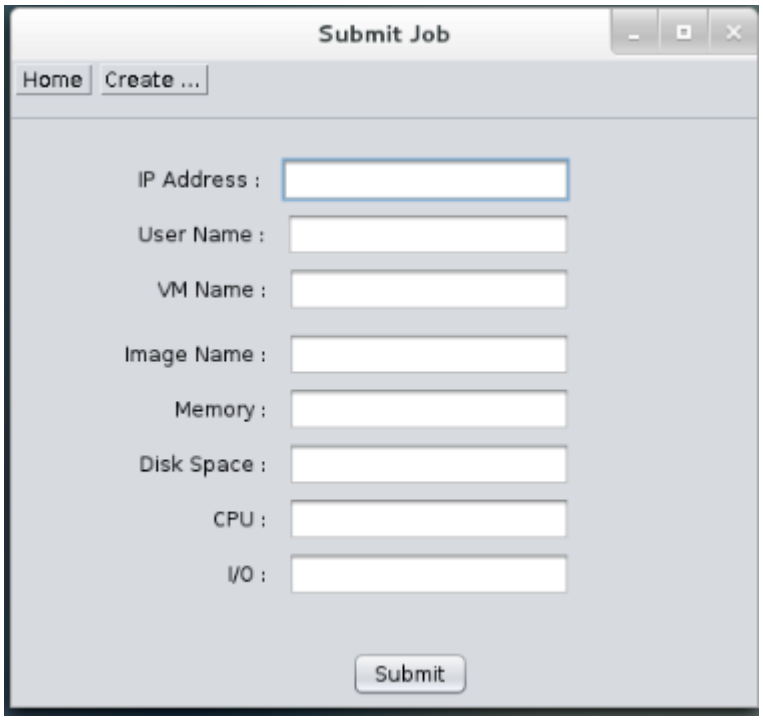



Applications
Places
master
Mon 16:48
nodeO
MASTER DASHBOARD!
NODE 0
NODE 1
NODE 2
NODE 3
NODE 4
NODE 5
Enter Node
SHOW VMS !
Virtual Machines
Destination
Migrate !
REFRESH !
USERS !
VCPU
Memory
Disk
Parameter
Scale
CENTOS
[*Master_Socket.java (~/Deskto...
root@nodeO:/home/nodeO/Docu...
[main]
root@nodeO:/home/nodeO/Desk...
[Sent Mail - harshdesai.svnit@g...
[main]
MASTER DASHBOARD!
1 / 4

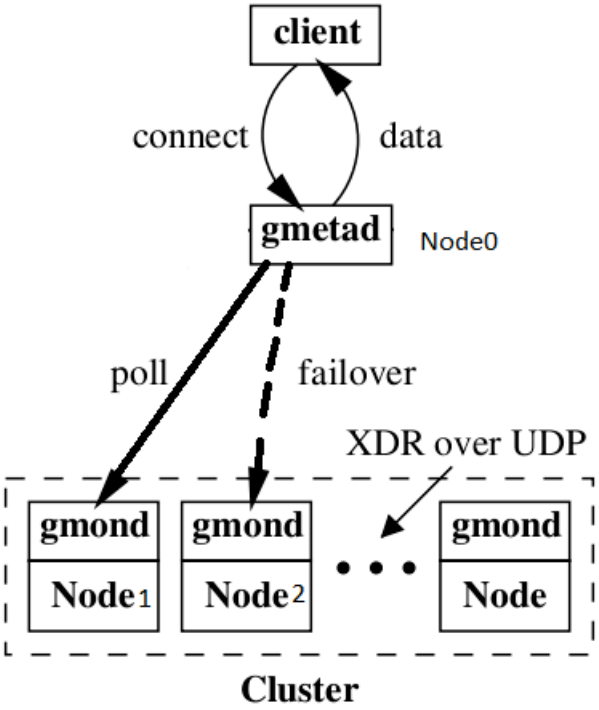
Client User Interface




Create VM
Home
Submit Job
Virtual Machines
sunish_virt1
User Name :
Sunish
IP Address :
node0
VM Name :
sunish_virt1
OS Type :
CentOS
Memory :
1049000
KB
Disk Space :
7.1
KB
CPU :
2
I/O :
10
Use existing VM image
Create


Submit Job
Home
Create ...
IP Address :
User Name :
VM Name :
Image Name :
Memory :
Disk Space :
CPU :
I/O :
Submit

Monitoring system



Ganglia monitoring system monitors all the hosts in the cluster. Gmetad service which is running on the master node0 polls data from gmond service running on all the other nodes (node1, node2, node3, node4, node5) of the cluster every 10 seconds. Master node performs load balancing, scale up, scale down and migration using metrics reported by the gmond daemon running on all the nodes. These metrics are stored in /var/lib/ganglia/rrds folder on the master.

Graphs corresponding to each metrics are created and stored in /var/lib/ganglia/graphs/cloud1 folder every 10 seconds to display real time graphs on the administrator User interface for every node.

API used for this component is rrdtool and gstat. The script monitor.py is created using these APIs. Description of the script is given in Implementation Details section.

Script generate_graphs.py will run in background which will continuously generate graphs for each node and store it in /var/lib/ganglia/graphs folder.

Socket Programming

Socket Programming is primarily used to give an extensionaly to the load balancing algorithm. In the load balancing algorithm, the client or the admin is asked about the decision if the algorithm finds out that the load is unbalanced.

When the master UI is started, the socket program and load balancing algorithm is started too. When a new client wants to use the service, it will contact the master ip address directly. The master updates its list of clients and accepts the job. Once the master finds out that the load is being unbalanced, only then it will throw out a prompt to the respective client. The client will enter the value and the the decision if they want to scale or migrate their virtual machine. The response will be updated at the master side and load balancing will be performed accordingly.

IMPLEMENTATION DETAILS

Iaas System Script

The master script **main.py** provides the following functionalities using their corresponding scripts:

Create Vm Script Name:
- createexistingvm.sh
- createvm.sh

Change Number of Cpus of VM:
- changecpuvm.sh

Change Memory of VM:
- changememoryvm.sh

Migrate the VM
- migrate.sh

Scale up the Disk of VM:
- scale.py

Static Network Setup in VM:
- addhost.sh
- network_config.sh

Delete Vm:
- deletevm.sh

Destroy Vm:
- destroyvm.sh

Shut Down Vm:
- shutdownvm.sh

Pause Vm:
- pausevm.sh

Start Vm:
- startvm.sh

Reboot Vm:
- rebootvm.sh

Get free ips:
- getips.sh

SSH login script:
- sshLogin.pyĵ

- python main.py loadbalance vm_name i/o memory disk cpu

- The above code is used to place any VM in the cluster with the parameter of I/O, Memory, Disk and CPU.

- python main.py migrate from_node to_node vm_name
  - The above code can be used to migrate a vm from one node to another
- python main.py scaleup vm_name size
  - The above code is used to scale-up the disk of a particular vm to a size mentioned
- python main.py createvm username vm_name memory disk os cpu io ip_address
  - The above code is used to create a vm mentioning the parameters mentioned
- ./scale.sh vm_name size

  To be clear, we can only scale up the disk by the amount of memory we specify. However, we cannot scale down the disk by a particular size because centos uses xfs file system. It does not use lv file system. If we were using ubuntu as the operating system, we could have scaled down the disk to any size we specify.

  In order to increase the size of the disk of a virtual machine, we need to shut down the virtual machine first.

  >>Virsh destroy <vm_name>

  After shutting down the virtual machine, we need to supply the following command :

  >>qemu-img resize /home/images/<vm_name>.qcow2 +<size>G

  We start the virtual machine back by supplying the comand :

  >>virsh start <vm_name>

  We need to wait for the virtual machine to start back and thus we supply a command :

  >>sleep 60

After it starts back normally,the following commands are supplied to which explanations are given :

>>fdisk /dev/vda /*We are now using fdisk to create a new partition*/

>>n /*‘n’ was selected for adding a new partition*/

>>p /*‘p’ is then selected as we are making a primary partition*/

>>3 /*As we already have /dev/vda1 and /dev/vda2 as shown in previous images, we have gone with using ‘3’ for this new partition which will be created as /dev/vda3*/

>><enter>

>><enter> /*We just press enter twice above as by default the first and last cylinders of the unallocated space should be correct. After this the partition is then ready.*/

>>t /*‘t’ is selected to change to a partition’s system ID, in this case we change to ‘3’ which is the one we just created.*/

>>3

>>8e /*The hex code ‘8e’ was entered as this is the code for a Linux LVM which is what we want this partition to be, as we will be joining it with the original /dev/vda5 Linux LVM.*/

>>w /*‘w’ is used to write the table to disk and exit, basically all the changes that have been done will be saved and then you will be exited from fdisk.*/

>>reboot /*To reboot the machine so that changes are saved */

wait for 60 seconds, hence

>>sleep 60

>>ssh <vm_name>

>>pvcreate /dev/vda3 /*We use the pvcreate command which creates a physical volume for later use by the logical volume manager (LVM).*/

>>vgextend centos /dev/vda3 /*Now we extend the ‘centos’ volume group by adding in the physical volume of /dev/vda3 which we created using the pvcreate command earlie*/

>>lvextend /dev/centos/root /dev/vda3 /*he logical volume is then extended using the lvextend command.*/

>>xfs_growfs /dev/centos/root /*There is then one final step which is to resize the file system so that it can take advantage of this additional space, this is done using the xfs_growfs command for XFS based file systems.*/

- python migrate.py nodename src_name dest_name vmname

  The following command is the only command to perform live migration :

  It uses the pre-copy and along with stop-and-copy technique to perform live migration.

  >>virsh migrate --live vm_name qemu+ssh://node_name/system --unsafe

  Once the master User Interface opens, in order to perform live migration from the User Interface itself, Just Type any node name which has any virtual machine in the source textbox and any node name which has low load currently for the virtual machine to migrate to in the destination text box.

  Click on “SHOW VMS !”
  Now, click on the Virtual Machine Name you want to migrate and then click on perform migration.
  Wait

  The migration may take some time depending on load and the size of the guest virtual machine. virsh only reports errors. The guest virtual machine continues to run on the source host physical machine until fully migrated.

  Finally, run the command at the destination :
  >>virsh list

- python main.py createexistingvm username vm_name memory disk os cpu io

  ip_address img_file job_type

  - The above code is used to create a vm from an existing vm. It uses the same parameters as above and plus 2 more parameters.
  - img_file is the image used to create a vm using a particular image.
  - Job_type can be

        - deleteclone - delete the original vm and create a new one
        - create - create a new vm using the image and then delete it.
        - default - will create a copy of the original VM
- python main.py destroyvm vm_name
    - The above code will shutdown the VM immediately.
- python main.py deletevm vm_name
    - The above code will delete the VM permanently.
- python main.py pausevm vm_name
    - The above code will pause the VM.
- python main.py startvm vm_name
    - The above code will start the VM.
- python main.py shutdownvm vm_name
    - The above code will shutdown process of the VM in a normal manner.
- python main.py rebootvm vm_name
    - The above code will reboot the VM.
- python main.py changememoryvm vm_name size
    - The above code is used to change the memory of the vm using the size mentioned above.
- python main.py changecpuvm vm_name size
    - The above code will change the number of vCPUs of the vm. It uses hyperthreading to allocate 8 threads per core and thus we can utilize the core efficiently by assigning cores to the vm in terms of threads.

- python main.py getuserdetails
    - The above code can be used to retrieve the information of all the users and its corresponding VMs and its configuration.
- python main.py getnodedetails
    - The above code can be used to retrieve the information of all the nodes and its corresponding VMs and its configuration.
- python main.py getvmdetails
    - The above code can be used to retrieve the information of all the VMs and its configuration.
- python main.py modifyuserdetails user_name vm_name parameter new_value
    - The above code will modify the vm according to the parameter send to it.

      Parameters can be:
        - cpu : vCPUs can be updated by the user so corresponding changes are updated in the permanent storage and also updating the vm
        - memory : memory can be updated by the user so corresponding changes are updated in the permanent storage and also updating the vm
        - disk : disk can be updated by the user so corresponding changes are updated in the permanent storage and also updating the vm
        - io : vCPUs can be updated by the user so corresponding changes are updated in the permanent storage.
        - node: node of a vm can change on migration so that needs to be updated in the permanent storage.

- python main.py deleteuserdetails user_name vm_name
    - The above code will delete the vm and remove its entry from a permanent storage.
- python main.py adduserdetails user_name vm_name IP_address node_name CPU memory disk io snapshot
    - The above code will add a user detail into a permanent storage (i.e. a file: userdetails.txt) so that all the records of each vm, its user, which node and the vm configuration is stored.
- python main.py getips number
    - The above code will return all the ips which are available for use and can be given to the user client for use in their VM.
- python main.py start_monitor
    - The above code can be used to run the automatic load balancing and carrying out migration and scaling on a VM according to the decision algorithm.
    - The user is asked for inputs of whether it wants to scale or not. If the user says yes then the it will be asked the amount by which the user wants to scale.
    - If the vm crosses a 95% mark then the user is forced to scale up
    - The user is never forced to scale down
    - The admin is asked for inputs of whether it wants to migrate or not. If the admin says yes then the migration process begins
    - If the node crosses a 95% mark then the migration The user is never forced to scale down

- python sshLogin.py vm_name
    - Automatically logins to the VM with accepting the authorization key, passwords and starts interacting with the VM.

Monitoring system scripts

- Status of the cluster : python monitor.py --status –all
  The above command displays the status of all the nodes in the cluster (Cpu, memory, disk, io)
- Status of a node : python monitor.py --status nodename
  The above command displays the status of a particular node
- Particular Metric of a node: python monitor.py --metric metricname nodename
  The above command gives the value of a particular metric of a node stored by ganglia
- Live nodes in cluster python monitor.py --live-nodes
  The above command shows all the live nodes in the cluster
- Dead Nodes in cluster : monitor --dead-nodes
  The above command shows all the dead nodes in the cluster
- Generate graph: monitor --graph time[--second seconds | --hour hours | --day days | --week weeks] metricname nodename
  The above command creates a graph of data fetched by ganglia in rrd files for a time period specified in time tag
- Generate real time graphs: python generate_graphs.py
  It generates the graphs of all the nodes and vms and updates every 10 seconds in the /var/lib/ganglia/graphs folder.

Snapshots

The reason why snapshots are important are to manage fault tolerance.

The Virtual Machine Manager takes periodic snapshots of it's virtual machines regularly. The procedure to take snapshots would be using libvirt.

Libvirt has its own command for taking snapshots.

Once a snapshot is taken, it is stored in the Operating Systems file directory.

After about 2 hours, the snapshot is deleted.

That is when a new snapshot is taken.

The VM can restart using the snapshots.

User Interface

There will be two interfaces : 1 for the master and 1 for the slave.

The reason for the user interfaces would be to provide a smoother transition for the user and the master administrator to handle virtual machines smoothly.

Master GUI Functionalities : Can track all it's virtual machines. The following would be displayed on the master dashboard :

1. GUI of each slave. -

    a. Virtual Machine memory

    b. CPU Time c. I/O Buffer

2. GUI for each USER :

    a. The details of each user.

    b. Details involve amount of virtual machine memory allocated to each user.

    c. Amount of CPU Time for each user.

    d. Amount of I/O required for each user.

PROBLEMS ENCOUNTERED

Initial understanding of the entire architecture and approach needed to build an IAAS infrastructure took quite our time. We brainstormed few architectures and approaches to tackle this problem. We read many online blogs and research papers in order to get the best solution. After many iterations and modifying our understanding we finally got a good architecture which we can proceed with.
We had never worked at the Kernel level and worked with VMs. Understanding Libvirt and learning its capabilities was important.

Creating a VM: We started by actually creating VMs from scratch using the image file of CentOS. We found that this took lot of time to create a VM using this process and plus it can not be created without user intervention. We finally decided to use cloning of a base vanilla image in order to create a vm which was easy and fast.

Network:
We needed to provide a static ip address to each node and vm. Initially, we used dynamic ip addresses but every time when we used to restart the system and vm, a new ip address was assigned. For this we changed the configuration of /etc/sysconfig/network-scripts/ifcfg-em1 on nodes and vms.

But then we realized that the configuration of a vm cant be changed just after its created, so we searched on internet and found virt-sysprep command to run a script in a vm without running it and changed the ntwork configuration at the time of createvm.
After assigning the static ip addresses to the vms, we were not able to do ssh to the vm. So, we had to add a rule in iptables, so that the request to access port 22 of ssh can be granted.

We needed to maintain a permanent storage of all the User Details, VM configuration and Node information in the master node. This information was necessary to map all the information so in the UI all the information changes dynamically. We needed to know which VM is on which node at that time in order to migrate, scale the VM. We needed to know the VM and its configuration to load balance the VM dynamically. We needed to modify the details in order to keep it up to date so all the details are displayed on to the UI. We needed to change the node when we migrate, change the configuration on scaling. So integration of all this was tedious.

Hyperthreading the vCPUs with 8 threads and configuring caused a lot of issues.

Integrating each components i.e. master script of Iaas system, admin UI, User UI and monitoring scripts was time consuming. Taking real time load metrics from ganglia in master script of Iaas system and displaying real time graphs on the admin and user UI was a major task.

After creating a load balancing algorithm, testing was important. Creating suitable workloads which tests all the parameters and condition was really important. After running lot of stress programs, we found the hadoop job best tested our infrastructure.

Monitoring System: We needed to create a cluster on ganglia such that gmetad service running on master node can pull the metrics gmond service running on all the other nodes which needed to be monitored. These gmond services were not able to pass information about the metrics to the gmetad service on master initially. Then we configured the gmond.conf file on each node by specifying the ip address and port on which the gmetad service is listening.

Taking snapshots:

1. Libvirt being a fairly new tool, it took some time to understand how things work.

2. Creating a snapshot, finding out where it is stored and extracting snapshots from the stored area took some days.

Performing Migration without Network File System :

a. First of all, when performing migration the approach being used was to copy the image file of the virtual machine to the slave destination before performing migration. Only then when, we supply the command, we get the virtual machine up and running at the destination host. However, there were discrepancies when achieving such a migration.

Case 1: If the pages are dirtied while performing migration with this technique then the changes will not be saved at the destination host and this may cause inconsistency between the the source machine and the destination machine.

Case 2 : Copying virtual machine image from one machine to another machine causes a lot of overhead when we need to perform migration frequently. This causes the system to slow down and sometimes even stop.

Case 3 : Also the “Broken Pipe” error is encountered in case we keep a connection “ON” for a long time.

Thus, we decided to use a shared network where the virtual machine image files are shared between the physical hosts. For this, the Network File System had to be set up. When the Network File System is set up, then we get rid of the case where we need to copy the image files before performing migration. Also, no broken pipe error is occurred. System never slows down and migration is performed reliably.

b. Second of all , when performing migration, we get the following error :

libvirtError: unsupported configuration: Unable to find security driver for label selinux
This error means that the securty driver is switched on to permissive mode and hence, it wont be able to perform the necessary migration we desire. Howver, the only way to tackle this error was to disable selinux and restart the physical host. Also, the SELINUXTYPE is set to Targeted.

c. Also since, when migration is being performed, a lot of routine security checks are made. This throws the following error :

libvirtError: Unsafe migration: Migration may lead to data corruption if disks

TO tackle this problem, the following solution is proposed : the migration command is appended by – unsafe. This causes the migration to be performed. However it may cause a security threat. But since, we are ruling out the possibility of a security threat, we use the unsafe option.

Setting up the Network File System:

When setting up migration, the Network File System. Mounting a filesystem on an existing filesystem with files existing in the current file system on the destination host will throw the following error at the destination host when perfoming the mount.

Mount.nfs : existing files in te current directory.

Thus, the virtual machine images on the destination host need to be removed and the filesystem of node0 (master node) need to be mounted on the filesystem of the slave nodes(node1 and node2).

If the system is made to reboot, then the nfs filesystem is broken and hence, nfs needs to be built up again on both the master and the slave.

When the nfs service is not started and if we press the command :

>>showmount -e

we get the following error :

clnt_create: RPC : Unable to send.

When the filesystem is not properly exported, we get the following error at the destination host

nfs.mount : RPC unable to connect.

Scaling of disk :

When performing scaling of disk, if we are unable to ssh in the virtual machine, we will not be bale to perform scaling of disk.

Hence, when a virtual machine is created, the passwordless ssh has to be set up, so that we can ssh into the virtual machine and expand the logical volume size when we are in the virtual machine.

Since, we need to increase the virtual machine disk size, we need to deal with increasing size of the current disk after allocating it enough memory and then increase the appropriate disk by entering the virtual machine and using fdisk.

We update the existing partition and make it a linux filesystem. We write it to the physical machine disk. However there is a risk of data loss when performing increase in the disk size via partitioning. To solve this, a new partition is created. Once a new partition is created, we make it as linux file system and write it to the file. To make permanent changes to the file, we reboot the machine and then we do a pvcreate so that the change can be reflected int the physical partitioning. Now, we cannot use and other command to increase the size of the disk. Since this is an XFS filesystem, we use xfs_growfs. We can use this for growing the size of the partition. However, there is no provision to use it to decrease the size of the filesystem since, XFS does not support decreasing of filesystem. If it would be an ubuntu file system, then we would be able to decrease the size of the file system but then that would have been a result to data loss. Thus, we rule out the possibility of scaling down the disk of a virtual machine.

Socket Programming :

The load balancing algorithm works in such a fashion that it asks the user and administrator to make the decision once it detects that the load is being unbalanced. When it is for the master, a pop up block will be displayed at the master interface. When it is for the client, a pop up block should be displayed at the client interface. Howeverm in order for the client to receive the message from the load balancing algorithm, we need to pass the message via some mechanism and hence socket programming is used for such purpose. The main issue with socket

programming was that of the concept of peer to peer communication. Because when the client starts, it sends a message to the server of the master. However, when the master needs to send messages according to the load balancing algorithm, then it will directly contact the server of the client machine and then the client machine will respond to the server via a pop up box. In this way we use socket programming for efficient load balancing.

Master User Interface :

There were multiple considerations of the master User Interface. Since, the load balancing algorothm works in Python, the user interface was initially decided to to be in python. However, due to Python's limited libraries in terms of user Interface, we scrapped the idea and implemented theMaster UI in Java. There were multiple ways to display the load of a certain physical host. However, the most efficient way was to directly load graphs at the master user Interface and make them dynamic using Java's functionality of JfreeCharts. The following operations are performed at the master user interface : 1. Migration 2. Scalling 3. Show utilization of Virtual Machines. In order to perform migration from the user interface, problems related to streaming of commands are encountered and thus are tackled using the following Java library : Process.java

Thus the mount data is to be flushed and then the mount has to be performed all over again.

Client User Interface:

There were multiple challenges that were encountered while designing the Client-Side User Interface.

Since, all the backend system was written in Python, the interface was also decided to be written in python. However, due to Python's limited libraries in terms of graphical interface support, we decided to explore other technologies.
Due to time constraints we decided to develop the Client Side User Interface in Java Swing rather than creating a web service.
We faced a lot of issues to gather real time data from ganglia in order to plot a graph of the VM.
After researching, we decided to plot the graphs in ganglia and then refreshing the plot in the Client Side UI after a fixed interval of time.

Faced a lot of issues while communicating between the Master User Interface and the Client User Interface.
This was solved by using a ssh script

Designing the Architecture :

1. Detailed study of all the research papers and extracting useful details to implement the system.

2. Understanding the difference between centralized and distributed system.

3. Designing the algorithm to implement the IAAS middleware system.

INSTALLATION MANUAL

Installation Guide to CentOS.

1. Download the ISO Image.
2. Make a Bootable Drive.
3. Begin Installation.
4. Select Language and Keyboard. Select your preferred language as well as the Keyboard type you have. Take care not forget to choose the correct keyboard or else you will end up with a few scrambled keys.
5. Change The Installation Destination. By default the installer will choose automatic partitioning for your hard disk. Click on the Installation Destination icon to change this to custom partitioning.
6. Click on the hard drive you want to install CentOS 7 and under the Other Storage Options, choose I will configure partitioning then click done.
7. Select The Partitioning Scheme. Next select the partitioning scheme to use for the mount points. In this case choose Standard Partition.
8. Create a Swap Space. You can create a swap space from one of the partitions and set the desired capacity, which is dependent on the RAM you have. Choose the File System for swap space as swap, and click on Re-format, though reformatting is optional. You can also name your swap space to whatever name you like but a name like swap is more descriptive.
9. Create a Mount Point. The next step is to create a mount point where the root partition will be installed. Depending on your requirements you might need to put the boot, home and root partition on different mount points. For this case we shall have only one mount point /.
10. After this set the Label and Desired Capacity to whatever you wish. A rule of thumb is to use descriptive names for the Label especially if the computer is to be used by different system administrators.
    Choose the file-system as ext4 and click on re-format.
11. Accept Changes.
12. Set Date And Time

Setting up the network:

On all the nodes, configure /etc/sysconfig/network-scripts/ifcfg-em1 file.

```
TYPE="Ethernet"
BOOTPROTO=dhcp
DEFROUTE="yes"
IPV4_FAILURE_FATAL="no"
IPV6INIT="no"
NAME="em1"
UUID="fa1b6610-38a1-4473-b03a-335bdc2c26d1"
DEVICE="em1"
ONBOOT="yes"
PREFIX="24"
DNS1=8.8.8.8
DNS2=8.8.4.4
NM_CONTROLLED=no
bridge=br0
IPADDR=192.168.1.10
NETWORK=192.168.1.0
GATEWAY=192.168.1.1
NM_CONTROLLED=no
```

Setting up bridge connection

1. Add a bridge
   brctl addbr br0

2. Add interface em1 to the bridge
   brctl addif br0 em1

3. Assign a dynamic ip address to the bridge
   dhclient br0

4. Add a iptables entry for accepting requests at port 22 of ssh
   iptables -A FORWARD --in-interface em1 --out-interface br0 -p tcp -d 192.168.1.0/255.255.255.0 --destination-port ssh -j ACCEPT
   iptables-save

Graphical User Interface

User Manual on the Graphical User Interface - Master

1. Run the master dashboard by keying in java master

2. Check any of the 3 checkboxes to extract the detail of it's slave nodes.

3. Click on any button to get the details extracted from the checked nodes.

4. The graphs will be based on csv files extracted from the rrd files retrieved from Ganglia monitoring system.

5. How to perform Migration?

    a. Key in the source node .

    b. Click on SHOW VMS!

    c. The VMS are shown in the table.

    d. Click on any one of the VMS

    e. Key in the destination node details

    f. Click on MIgrate!

6. How to perform Scaling ?

    a.Click in Scale !

    b. A new form will load.

    c.On new form :

    Scale up => Increase the number of VMS by keying in the number of VMS

    Scale down => Decrease the number of VMS by keying in the number of VMS

    Scale out=> Scale out the VMS by keying in the source node and the destination node!

User Manual on Graphical User Interface - USER

1. Enter the Virtual Machine Details by clicking on VMSearch.

2. Key in the credentials of the Virtual Machine by typing in

    a. Memory

    b. Input / Output

    c. Number of Virtual Machines

Installation Guide for Ganglia Monitoring System

1. Download latest epel

```
$ sudo wget https://dl.fedoraproject.org/pub/epel/epel-release-latest-7.noarch.rpm
$ sudo rpm -Uvh epel-release-latest-7.noarch.rpm
```

2. Install ganglia and its dependencies

```
$ sudo yum -y install apr-devel apr-util check-devel cairo-devel pango-devel libxml2-devel rpmbuild glib2-devel dbus-devel freetype-devel fontconfig-devel gcc-c++ expat-devel python-devel libXrender-devel
$ sudo yum install  rrdtool ganglia ganglia-gmetad ganglia-gmond ganglia-web httpd php apr apr-util
```

5. Configure Meta Node on master (192.168.1.10)

    Edit the /etc/ganglia/gmetad.conf file and specify the cluster name and host of the meta node (master).

```
$ sudo vi /etc/ganglia/gmetad.conf
```

```
data_source "cloud1" 192.168.1.10:8649
```

6. Configure Monitoring Nodes (host that you want to monitor)

   Edit the /etc/ganglia/gmond.conf file

```
$ sudo vi /etc/ganglia/gmond.conf


//Specify cluster name in the cluster section
cluster {
     name = "cloud1"
     owner = " unspecified "
     latlong = "unspecified"
     url = "unspecified"
}


//Specify the meta node host and port in udp_send_channel section
udp_send_channel {
          host = 192.168.1.10   //master ip
          port = 8649
          ttl = 1
}
```

7. Start the gmetad and httpd service on master node

```
$ sudo service gmetad start
$ sudo service httpd start
```

8. Start the gmond service in all the nodes

```
$ sudo service gmond start
```

9. Install ganglia-web package

```
$wget http://sourceforge.net/projects/ganglia/files/ganglia-web/3.5.10/ganglia-web 3.5.10.tar.gz

$ tar -xzvf ganglia-web-3.5.10.tar.gz
$ cd /etc/ganglia-web-3.5.10/
$ make install
```

Ganglia Web interface can be viewed at http://192.168.1.10:8649/ganglia

Setting up NFS

About NFS Mounts and why we use it ?

NFS mounts work to share a directory between several servers. This has the advantage of saving disk space, as the home directory is only kept on one server, and others can connect to it over the network. When setting up mounts, NFS is most effective for permanent fixtures that should always be accessible.

Setting Up NFS at the Master Node

Step one : Download the required software

>>su -

Enter Password :

>>yum install nfs-utils nfs-utils-lib

>>chkconfig nfs on

>>service rpcbind

>>service nfs start

Step two : Export the shared directory

The next step is to decide which directory we want to share with the client server. The chosen directory should then be added to the /etc/exports file, which specifies both the directory to be shared and the details of how it is shared.

The directories shared in the current cloud system are :

1. /home/images (where the qcow2 image files are stored)

2. /home/qemu/images (where the xml files are stored)

>>vi /etc/exports

/home/images node1(rw,sync,no_root_squash,no_subtree_check)

/home/images node2(rw,sync,no_root_squash,no_subtree_check)

/home/qemu/images node1(rw,sync,no_root_squash,no_subtree_check)

/home/qemu/images node2(rw,sync,no_root_squash,no_subtree_check)

rw: This option allows the client server to both read and write within the shared directory

- sync: Sync confirms requests to the shared directory only once the changes have been committed.
- no_subtree_check: This option prevents the subtree checking. When a shared directory is the subdirectory of a larger filesystem, nfs performs scans of every directory above it, in order to verify its permissions and details. Disabling the subtree check may increase the reliability of NFS, but reduce security.
- no_root_squash: This phrase allows root to connect to the designated directory

>>exportfs -a
Setting up the NFS at the Slave Nodes

Step one : Download the required software

>>yum install nfs-utils nfs-utils-lib
Step two : Mount the directories

>>mkdir -p /home/images
>>mkdir -p /home/qemu/images
>>mount node0:/home/images /home/images
>>mount node0:/home/qemu/images /home/qemu/images

Just to check whether the directory is mounted or not,supply the following command to the slave node:

>>df -h


Applications Places Terminal
Mon 17:38 nodeO
home
disk_scale.java
Trash
disk_scale2.sh
anisha@10.40.0.11
disk_scale1.sh
main
cloudCrime.jar
Client_Socket.java
disk_scale.sh
abc.sh
login.py
sshLogin.py
root@nodeO:~
File Edit View Search Terminal Help
[root@node1 images]# df -h
Filesystem Size Used Avail Use% Mounted on
/dev/mapper/centos-root 50G 29G 22G 58% /
devtmpfs 16G 0 16G 0% /dev
tmpfs 16G 148K 16G 1% /dev/shm
tmpfs 16G 17M 16G 1% /run
tmpfs 16G 0 16G 0% /sys/fs/cgroup
/dev/sda1 494M 214M 281M 44% /boot
/dev/mapper/centos-home 865G 16G 850G 2% /home
node0:/home/images 865G 38G 827G 5% /home/images
node0:/home/qemu/images 865G 38G 827G 5% /home/qemu/images
[root@node1 images]# ssh node2
Last login: Mon Dec 7 17:34:06 2015 from node0
[root@node2 ~]# df -h
Filesystem Size Used Avail Use% Mounted on
/dev/mapper/centos-root 50G 5.3G 45G 11% /
devtmpfs 16G 0 16G 0% /dev
tmpfs 16G 2.6M 16G 1% /dev/shm
tmpfs 16G 17M 16G 1% /run
tmpfs 16G 0 16G 0% /sys/fs/cgroup
/dev/mapper/centos-home 865G 570M 865G 1% /home
/dev/sda2 494M 118M 376M 24% /boot
/dev/sda1 200M 9.8M 191M 5% /boot/efi
node0:/home/images 865G 38G 827G 5% /home/images
[root@node2 ~]# ssh node0
Last login: Mon Dec 7 17:31:47 2015 from node1
[root@node0 ~]#
7
CENTOS
[sp11_final.pdf - Mozilla Firefox]
root@nodeO:~
1 / 4

User Manual-Cloud Environment

The following are the commands that the user can use to access the cloud environment.

- Load Balancing: cloud1 loadbalance vm_name i/o memory disk cpu
- Migration Command: cloud1 migrate vm_name new_vm_name snapshot_name
- Scale Up Command: cloud1 scaleup vm_name size
- Scale Down Command: cloud1 scaledown vm_name size
- Create VM Command: cloud1 createvm vm_name memory disk os
- Destroy VM Command: cloud1 destroyvm vm_name
- Pause VM Command: cloud1 pausevm vm_name
- Start VM Command: cloud1 startvm vm_name
- Shutdown VM Command: cloud1 shutdownvm vm_name
- Reboot VM Command: cloud1 rebootvm vm_name
- Clone VM Command: cloud1 clonevm vm_name new_vm_name
- Snapshots: cloud1 createsnapshot
- Current Snap shot: cloud1 showsnapshot vm_name
- Restore Snap shot: cloud1 restoresnapshot vm_name snapshot_name
- Snapshots: cloud1 createsnapshot
- Current Snapshot:cloud1 showsnapshot vm_name
- Restore Snapshot: cloud1 restoresnapshot vm_name snapshot_name
- Migrate VM: python migrate.py src_name dest_name vm_name
- Scale up disk: ./scale.sh vm_name size
- Master UI: java master
- Socket program Server: java Master_Socket
- Socket Program Client: java Client_Socket
- Status of the cluster : python monitor.py --status –all
- Status of a node : python monitor.py --status nodename
- Particular Metric of a node: python monitor.py --metric metricname nodename
- Generate real time graphs: python generate_graphs.py
- Live nodes in cluster python monitor.py --live-nodes
- Dead Nodes in cluster : monitor --dead-nodes
- Generate graph: monitor --graph time[--second seconds | --hour hours | --day days | --week weeks] metricname nodename
- Client UI: java –jar ClientUI.jar

Work Distribution

1. Anisha Nainani:

Monitoring System:

- Deciding and studying about the monitoring system.
  - After evaluating zenoss and ganglia, ganglia is decided as our monitoring system.
  - Study architecture and working of ganglia
- Setting up the entire ganglia environment in the cluster and gathering the rrd files

RRDTool

- Understanding various rrdtool features and various commands which can be used in our project.
- Using RRDTool to fetch metrics and status of all the nodes in cluster to provide data to load balancing and vm placement decision algorithms running on master as well as master and client ui to display information about nodes and vms

Real time graphs

- Creating real time graphs to display metric information about nodes on Master UI
- Creating real time graphs of vms to display metric information about vms on Client UI

Networking:

- Setting up static ip address for each node
- Setting up the bridge connection between host interface and KVM
- Allocating the static ip address to each vm on createvm by virt-sysprep
- Enter the ip address to hostname mapping of all the hosts and vms in /etc/hosts.
- Enable password less SSH among the nodes and vms
- Setting up Port forwarding to enable access to the cluster from external network

Creating loads for testing:

- Creating Hadoop cluster and workloads for testing the system.

Integration and bug Fixes

2. Harsh Desai:

Migration:

- Performing migration based on pre-copy and stop-and-copy techniques.
- Creating the migrate.py script to perform migration.

NFS Setup

- Setting up the Network File System between master and slaves.
- Mounting the images directory of the master on all the slaves.

Scaling of Disk

- Scaling up of disk of any virtual machine to specified size of upto 10G.
- Creation of scale.sh to perform scale up of disks.
- Manipulating partitioning of disks to prevent loss of existing data.

Master User Interface

- Developing the User Interface for Master Node in Java to display all the nodes and virtual machine information and also, provide an interface to scale and migrate virtual machines from the master on button click.
- Displaying monitoring data from all virtual machines and nodes.

Socket Programming

- Developing Socket Programs for client and master for efficient working of the load balancing algorithm.
- Providing an interface for client to make decisions on performing migration and scaling based on the load balancing algorithm.

Integration and Bug Fixes

3. Kanav Kaul

- Installed CentOS and KVM on the machines.
- Researched about Libvirt and wrote the following python scripts:
  - Start the VM - used to start the VM.
  - Stop the VM - used to stop the VM.
  - Restart the VM – used to restart the VM.
- Designed the Client Side User Interface. Under the client side interface, I designed the following layouts:

  - CreateVM
    -User can create a new VM by either starting from scratch or by using the configuration of the existing VM's.
  - Submit Job
    -User can submit a job to the Master.
  - Dashboard
    -User can view the status of all the VM's here. He/she can also view the performance graphs of an individual VM.
    -User can also change the initial parameters of the VM and cause scaling. Scaling takes place one parameter at a time.
  - Performance Analysis
    -User can view the different loads of a particular VM. The different loads shown are:

CPU Information
I/O Information
Memory Information

- Integration and Bug Fixes

4. Sunish Sheth:

- Architecture and Approach:
  - Researching various concepts like load balancing, VM placement, migration, scaling and devising the architecture and approach to create a IAAS system.
- Libvirt:
  - Understanding various libvirt features and various commands which can be used in our project.
- Load Balancing (Decision Algorithm):
  - After learning about load balancing, devising an algorithm which can efficiently perform load balancing and make decision regarding migration of VM from one physical machine to another and scaling up or scaling down a VM depending on its demands.
- Cloud Environment:
  - Creating scripts for
    - CreateVM
    - Cloning a VM from the original VM in 3 different ways.
    - Scale up and down the number of vCPUs and using Hyperthreading
    - Scale up and the down the memory of the VM
    - Delete a VM from the cloud environment and also removing it from the permanent storage
    - Destroying the VM
    - Pause the VM
    - Get free IPs in our domain which can be used by our clients
    - Creating a permanent storage to store information related to Users, VMs and Node of the cloud environment.
  - Dynamically adjusting according to the number of live nodes and vms in the cluster.(Data obtained from monitoring system.)

- Created a SSH login script which takes as input username and password and automatically logs into the host system.
- Integration and bug Fixes.