



San José State
UNIVERSITY

CMPE 283 -VIRTUALISATION

INDIVIDUAL PROJECT

Advisor: **Dr. Simon Shim**

Submitted by:

Anuradha Vakil

anuradhavakil@yahoo.com

ID: 008016216

INDEX OF CONTENTS:

1. Introduction.....	3
2. Background.....	4
3. Requirements.....	5
4. Design.....	6
5. Implementation.....	10
6. Answers to the questions.....	16
7. Discussion.....	17
8. Conclusion.....	19
9. References.....	20

Introduction

Goal:

The goal of the project is to write availability manager using ESXi's Java API (WS API) which can be used as disaster recovery monitor for datacenter.

Objective:

The objective for designing availability manager is to gain hands on experience with hypervisor and learn how to manage them through VI APIs. This exercise would also give us an opportunity to apply concepts discussed in real world scenarios and learn issues that arise while applying the same.

While doing this exercise we need to recover VM or the host from complete failure by monitoring their liveliness. If the VM fails then host is checked, if the later is alive then VM is restored using earlier snapshot. If the host is not alive, then VM is moved to new host.

Need for Availability Manager:

In today's time when organizations of all sizes and catering to wide variety of services in all walks of life continuously depend on their IT systems, any down time may affect the organizations or public in horrific way. Duty of availability manager is to monitor and take a corrective action as soon as it senses that VM or the Hosts, hosting clusters of VMs is not responding. It has to sense in intelligent way whether the system under monitoring is powered off or is not able to respond due to some kind of hardware, software or network failure. Previously, when systems were batch oriented mainframes the down time used to be in days and would damage the organization significantly. Disaster recovery / availability manager is responsible for bring the systems in running state in case of failure.

Background:

Unlike mid 1970's when the systems were batch oriented mainframes that could be down for number of days, significantly damaging image of the organization, today's organizations which are heavily dependent on IT systems cannot afford to have any down time. With objective of 99.999% availability for critical systems, disaster recovery plays very crucial role.

Traditional disaster recovery approaches like daily or weekly data back ups and assuming that network computing can be readily rebuilt following a disaster has been proved inadequate. The real time processing makes it more difficult to recover the transactions which have occurred in-between of last data back up and disaster.

This leads to the desire of achieving more automatic recovery that can establish all the transactions before disaster. By using an availability manager which continuously checks/monitors health of VMs and in case of failure can replicate the running virtual machine to remote site and can boot that instantaneously. This also benefits the organization by creating lot of saving in hardware and maintenance of datacenters, real estate, power and staffing.

The proposed availability manager will be good example of cost effective disaster recovery system that can be implemented with minimal manual effort and in less time.

Requirements:

Functional Requirements:

1. To develop an availability manager using ESXi's Java API (WS API).
2. It will interface with virtual infrastructure using APIs.
3. The virtual infrastructure would comprise of – a host running on VMWare ESXi that is managed by VMWare vCenter server. The host would have VM running on it.
4. The availability monitor needs to take snapshot of the VM every 10 minutes.
5. The availability manager needs to monitor health of the VMs by checking its heartbeat. In case of failure it needs to check whether the vhost is healthy or not. Two scenarios can arise:
 - i) VM dead & vhost alive-revert the snapshot on the dead VM
 - ii) VM dead & vhost dead- Try to revive vhost, if effort does not succeed then remove the dead vhost from network and add new vhost and start the VM on new vhost- cold migrate.
6. Alarm-The availability manager also needs to sense whether VM has failed or has been powered off, in later case it need not take any action.
7. Java threads need to perform above mentioned tasks.

Non-Functional Requirements:

1. Availability: The proposed availability manager should be highly available and should be able to sense the problem at an early stage and take corrective action fast.
2. Load balancing: Irrespective of numbers of VMs running on the vhost, availability manager should be able to perform its tasks.
3. System recoverability: It should assure that in case of any kind of failure availability manager will be able to revert to current state of VM on same vhost or replicate and boot the VM in question on new vhost. Under any circumstances recovery of system is assured.
4. Simulation of disaster: In order to check whether the manager is performing upto mark or not, disaster has to be simulated. In our case, network failure would be simulated.

5. Fault tolerance: Manager has to ensure that system is fault tolerant that is under event of failure, instantaneously new VM would be created and restored to the point where it failed, thus preventing data loss and zero down time.

Design:

Architecture:

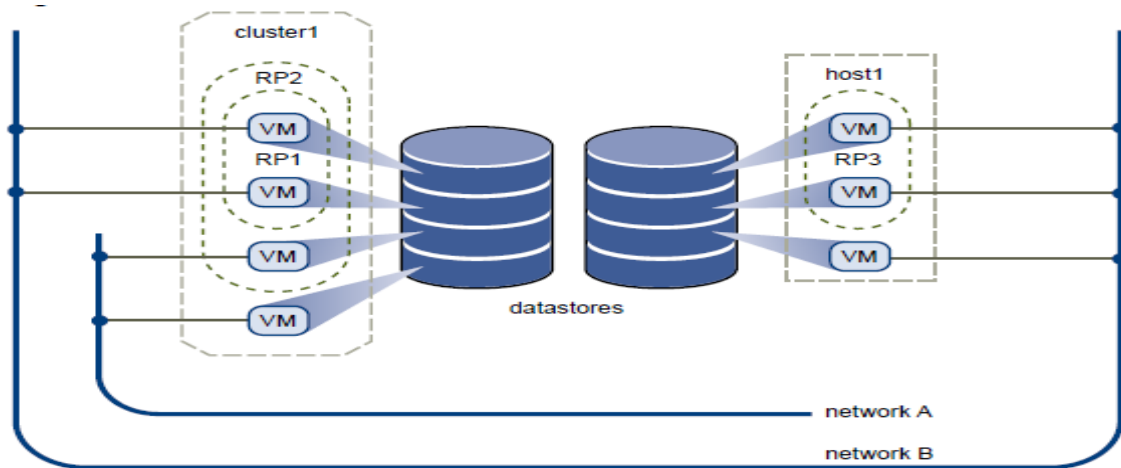


Figure 4.1: Virtual Datacenter Architecture in VMWare VSphere

VMware Infrastructure is a set of virtual elements used to build a virtual data center. For this exercise we would be starting with one host which has one VM. When disaster is simulated and network goes down, we would restart the VM by reverting to saved snapshot. Later on we would simulate vhost's failure too and would remove the dead vhost and add a new vhost. We would boot the old VM on this new vhost.

Contents:

Datastores - Datastores are the storage resources which represents combinations of underlying physical storage resources in the data center.

Networking resources - Networks in the virtual environment connect the infrastructure like virtual machines, hosts, vcenter to each other or to the physical network outside of the virtual data center.

Virtual Machines - Virtual machines are designated to a particular Host, Cluster or Resource Pool and a Datastore when they are created. They run individual Operating system and behave as a physical computer/resource.

Computing and memory resources called Hosts, Clusters and Resource Pools -When one or more machines are grouped together to work and to be managed as a whole, the aggregate computing and memory resources form a Cluster. Machines can be dynamically added or removed from a Cluster. Computing and memory resources from Hosts and Clusters can be finely partitioned into a hierarchy of Resource Pools. Every host is designated to particular vCenter and every VM is designated to a particular host.

Availability Manager

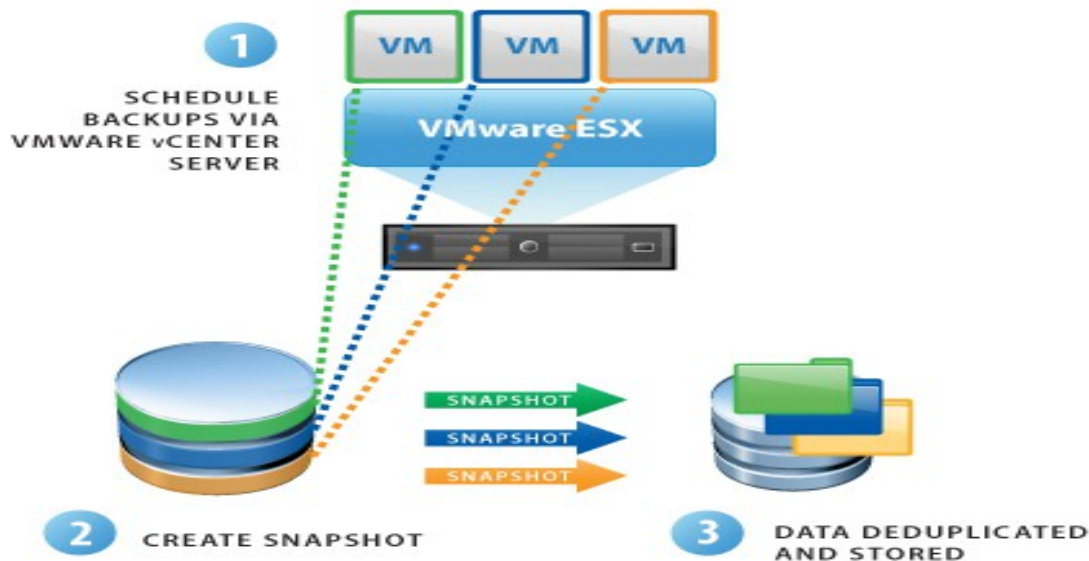


Figure 4.2: Process of disaster recovery done by Availability manager

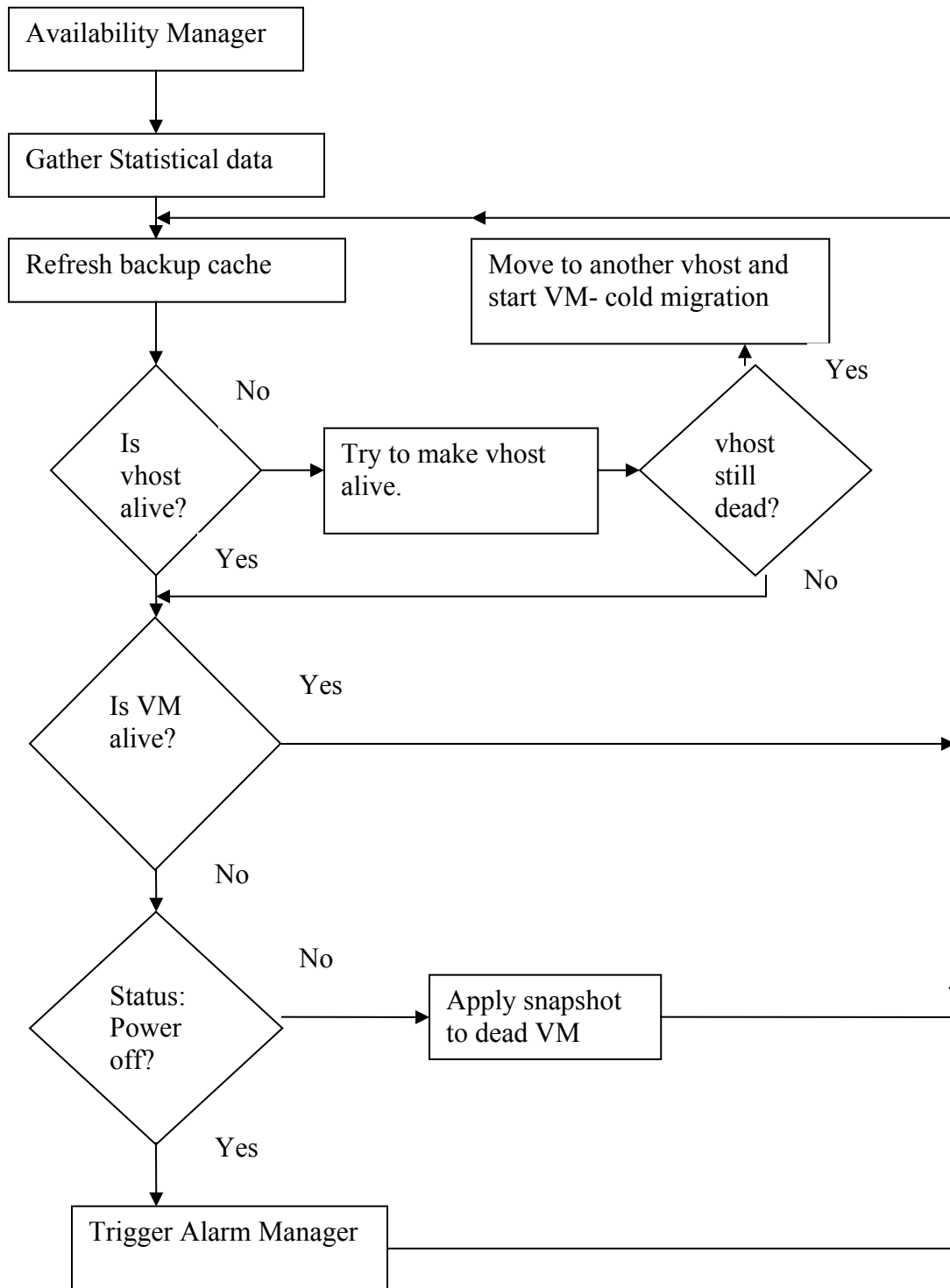
Availability Manager is designed as shown in the above picture. It refreshes the backup cache of the Virtual machine every 10 minutes of time and maintains a fresh instance of the current state of the machine in the form of snapshots. When a disaster occurs or manually simulated, the availability manager comes into place and clones the very recent snapshot of the virtual machine and restarts it on another safe, reachable host.

The virtual machine should support the feature of multiple snapshots. The designing part of the project is done after considering and close observation of the in-built features of VSphere like

³⁵₁₇ High Availability (HA): Provides easy-to-use, cost effective high availability for applications running in virtual machines. In the event of server failure, affected virtual machines are automatically restarted on other production servers that have spare capacity.

³⁵₁₇ VMware Distributed Resource Scheduler (DRS) – Intelligently allocates and balances computing capacity dynamically across collections of hardware resources for virtual machines

DATAFLOW:



IMPLEMENTATION:

Environment:

The implementation part of this project makes use of VSphere Vclient environment that contains:

³⁵₁₇ Hosts, clusters, resource pools virtual machines, datastores and networks.

³⁵₁₇ Virtual Infrastructure JAVA API by VMWare Infrastructure.

Prior to the development of disaster recovery system, the following steps are prerequisites:

³⁵₁₇ Creation of two VHosts which are VMware ESXi servers.

³⁵₁₇ Creating a Vcenter with Windows 2008 server installed, adding the hosts to the Vcenter and creates virtual machines on one of the hosts.

Implementation of Java API, for automating the process of disaster recovery through availability manager has following steps:

³⁵₁₇ Check if the host is alive by pinging and if so then ping to check if VM is alive.

³⁵₁₇ If the VM is alive a snapshot is created otherwise its checked whether VM is manually powered off or not, if its not then VM is reverted back from snapshot taken earlier.

³⁵₁₇ If it is manually powered off then alarm is triggered.

³⁵₁₇ If host is not alive then cold migration takes place.

³⁵₁₇ Pinging of the new host to check its status.

³⁵₁₇ If the destination host is live , then the cloned VM is migrated to it in a shared datastore. Else, the program exits.

Tools:

Eclipse IDE – for integrating the Java API to implement the disaster recovery .

VSphere Client- For managing VMs and hosts.

The JAVA Classes used in the project are:

1. HighAvailabilityManager – Main class that invokes instances of all the other classes, displays statistical data, initialize VM and Ping host.
 - a. VMManager (Inner Class) - Thread class managing VM availability
2. VMSnapshotGenerator (Inner Class) – Thread class generating snapshot and restoreFromSnapShot
3. VMPowerStateAlarm – trigger alarm if the VM is manually powered off
4. MigrateVM – implements code to migrate VM

Screenshots:

Screenshot 1: VM displaying statistical data:

```

if(vmsnap!=null) {
    Task task = vmsnap.revertToSnapshot Task(null);
    if(task.waitForMe()==Task.SUCCESS) {
        System.out.println("Reverted to snapshot: " );
    }
}
}

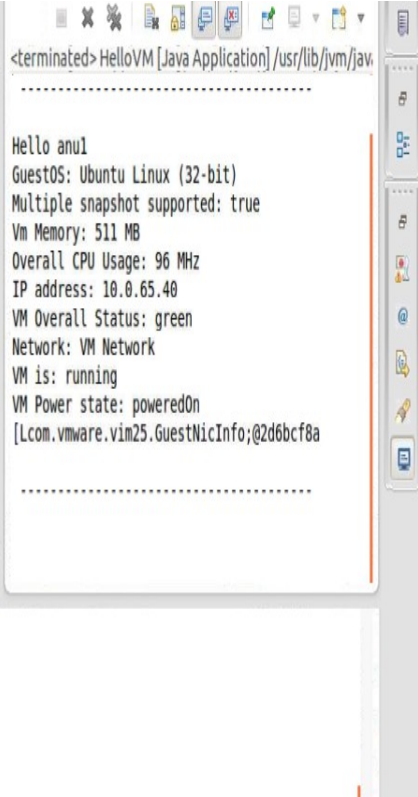
public static void displayStatisticsData(VirtualMachine vm) throws Exception{

    vm.getResourcePool();
    System.out.println("\n ----- \n" );
    System.out.println("Hello " + vm.getName());
    System.out.println("GuestOS: " + vm.getConfig().getGuestFullName());
    System.out.println("Multiple snapshot supported: " + vm.getCapability().isMultipleSnapshotsSupported());

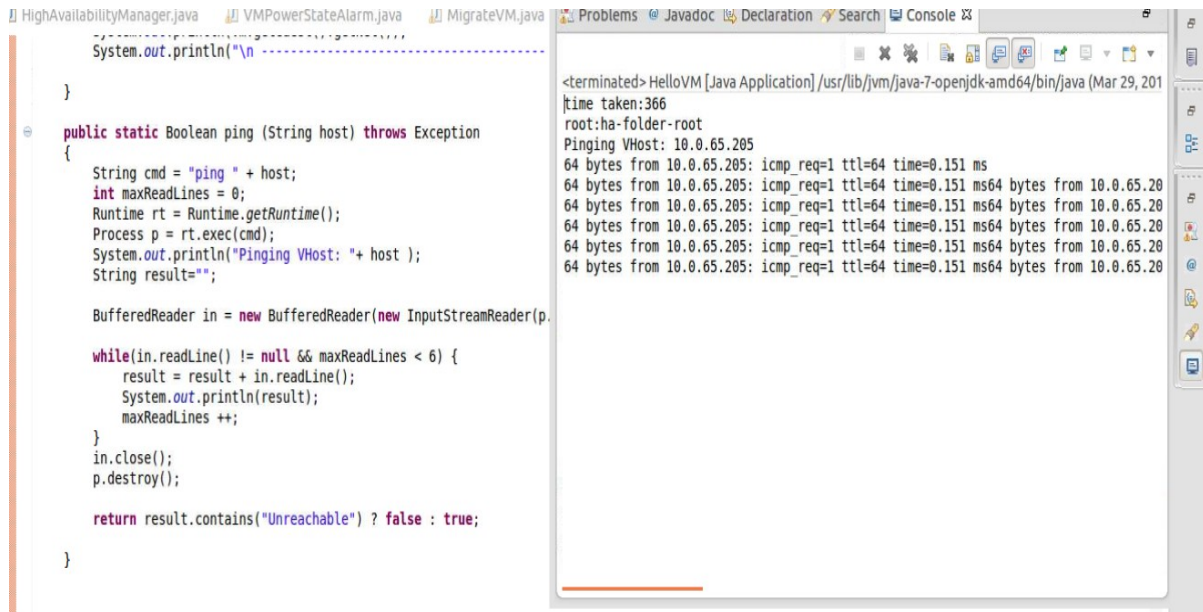
    System.out.println("Vm Memory: " + vm.getSummary().quickStats.getHostMemoryUsage() + " MB");
    System.out.println("Overall CPU Usage: " + vm.getSummary().quickStats.getOverallCpuUsage() + " MHz");
    System.out.println("IP address: " + vm.getGuest().getIpAddress());
    System.out.println("VM Overall Status: " + vm.getOverallStatus().toString());
    System.out.println("Network: " + vm.getNetworks()[0].getName());
    System.out.println("VM is: " + vm.getGuest().getGuestState());
    System.out.println("VM Power state: " + vm.getRuntime().getPowerState().toString());
    System.out.println(vm.getGuest().getNet());
    System.out.println("\n ----- \n" );

}

```



Screenshot 2: Pinging host- Successful ping



The screenshot shows an IDE with two panes. The left pane displays the source code of a Java application, and the right pane shows the console output.

```
System.out.println("\n\n");

}

public static Boolean ping (String host) throws Exception
{
    String cmd = "ping " + host;
    int maxReadLines = 0;
    Runtime rt = Runtime.getRuntime();
    Process p = rt.exec(cmd);
    System.out.println("Pinging VHost: " + host );
    String result="";

    BufferedReader in = new BufferedReader(new InputStreamReader(p.

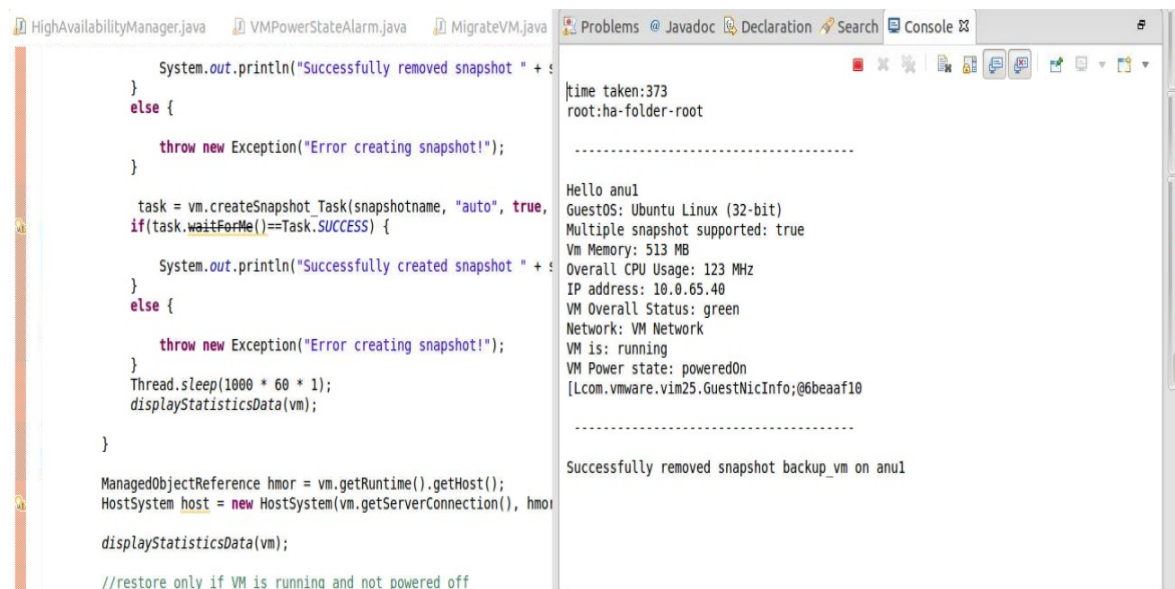
    while(in.readLine() != null && maxReadLines < 6) {
        result = result + in.readLine();
        System.out.println(result);
        maxReadLines ++;
    }
    in.close();
    p.destroy();

    return result.contains("Unreachable") ? false : true;
}

}
```

```
<terminated> HelloVM [Java Application] /usr/lib/jvm/java-7-openjdk-amd64/bin/java (Mar 29, 201
time taken:366
root:ha-folder-root
Pinging VHost: 10.0.65.205
64 bytes from 10.0.65.205: icmp_req=1 ttl=64 time=0.151 ms
64 bytes from 10.0.65.205: icmp_req=1 ttl=64 time=0.151 ms64 bytes from 10.0.65.20
64 bytes from 10.0.65.205: icmp_req=1 ttl=64 time=0.151 ms64 bytes from 10.0.65.20
64 bytes from 10.0.65.205: icmp_req=1 ttl=64 time=0.151 ms64 bytes from 10.0.65.20
64 bytes from 10.0.65.205: icmp_req=1 ttl=64 time=0.151 ms64 bytes from 10.0.65.20
64 bytes from 10.0.65.205: icmp_req=1 ttl=64 time=0.151 ms64 bytes from 10.0.65.20
```

Screenshot 3: Remove previous snapshots- before creating new one. This is done in order to save space and keep updated copy.



The screenshot shows an IDE with two panes. The left pane displays the source code of a Java application, and the right pane shows the console output.

```
System.out.println("Successfully removed snapshot " + s
}
else {
    throw new Exception("Error creating snapshot!");
}

task = vm.createSnapshot_Task(snapshotname, "auto", true,
if(task.waitForMe()==Task.SUCCESS) {

    System.out.println("Successfully created snapshot " + s
}
else {
    throw new Exception("Error creating snapshot!");
}
Thread.sleep(1000 * 60 * 1);
displayStatisticsData(vm);

}

ManagedObjectReference hmor = vm.getRuntime().getHost();
HostSystem host = new HostSystem(vm.getServerConnection(), hmor

displayStatisticsData(vm);

//restore only if VM is running and not powered off
```

```
time taken:373
root:ha-folder-root

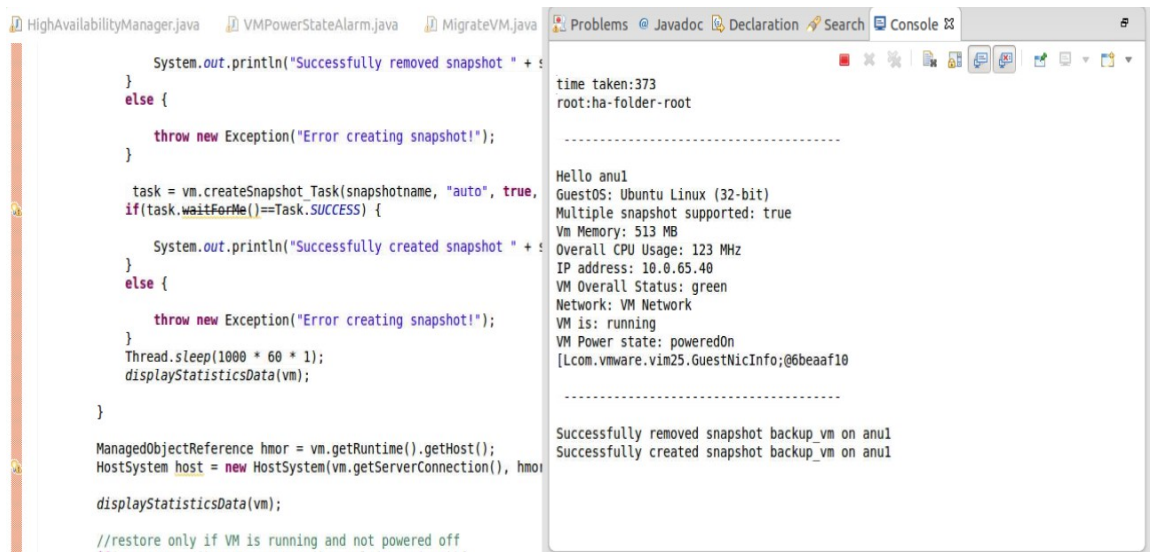
-----

Hello anu1
GuestOS: Ubuntu Linux (32-bit)
Multiple snapshot supported: true
Vm Memory: 513 MB
Overall CPU Usage: 123 MHz
IP address: 10.0.65.40
VM Overall Status: green
Network: VM Network
VM is: running
VM Power state: poweredOn
[Lcom.vmware.vim25.GuestNicInfo;@6beaaf10

-----

Successfully removed snapshot backup_vm on anu1
```

Screenshot 4: Create new snapshot- activity carried out every 10 min – API



The screenshot shows a Java IDE with a file named `HighAvailabilityManager.java`. The code implements a loop that creates and removes snapshots of a virtual machine named `anul` every 10 minutes. The console output displays the results of these operations, including system statistics and success messages.

```
System.out.println("Successfully removed snapshot " + snapshotname);
}
else {
    throw new Exception("Error creating snapshot!");
}

task = vm.createSnapshot_Task(snapshotname, "auto", true,
if(task.waitForMe()==Task.SUCCEEDED) {
    System.out.println("Successfully created snapshot " + snapshotname);
}
else {
    throw new Exception("Error creating snapshot!");
}
Thread.sleep(1000 * 60 * 1);
displayStatisticsData(vm);
}

ManagedObjectReference hmor = vm.getRuntime().getHost();
HostSystem host = new HostSystem(vm.getServerConnection(), hmor);

displayStatisticsData(vm);

//restore only if VM is running and not powered off
```

Console Output:

```
time taken:373
root:ha-folder-root

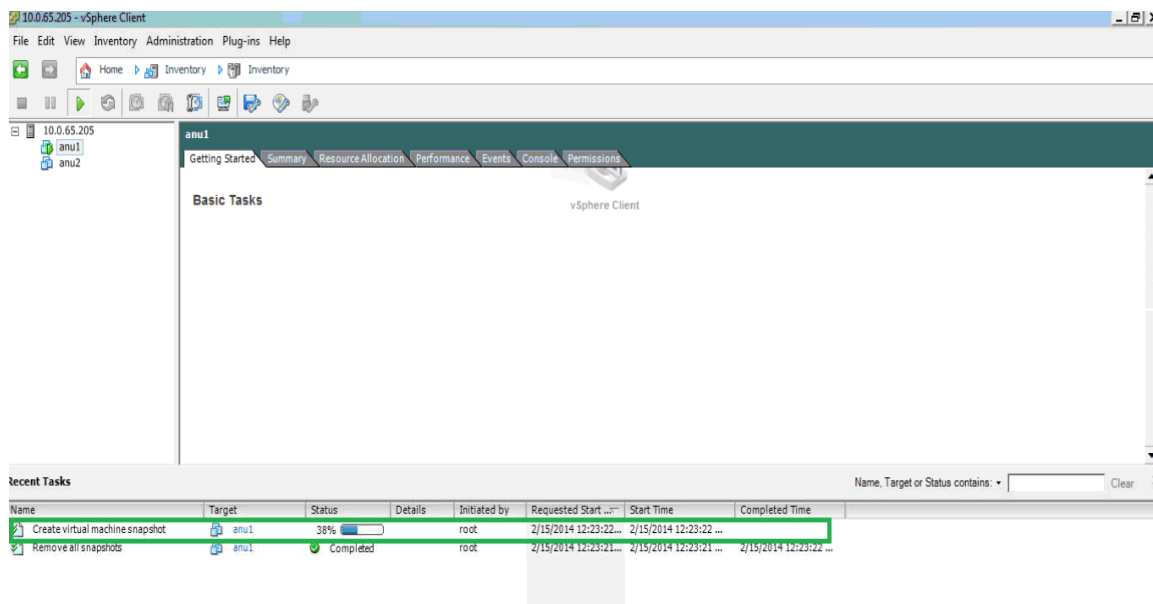
-----

Hello anul
GuestOS: Ubuntu Linux (32-bit)
Multiple snapshot supported: true
Vm Memory: 513 MB
Overall CPU Usage: 123 MHz
IP address: 10.0.65.40
VM Overall Status: green
Network: VM Network
VM is: running
VM Power state: poweredOn
[Com.vmware.vim25.GuestNicInfo:@6beaaf10

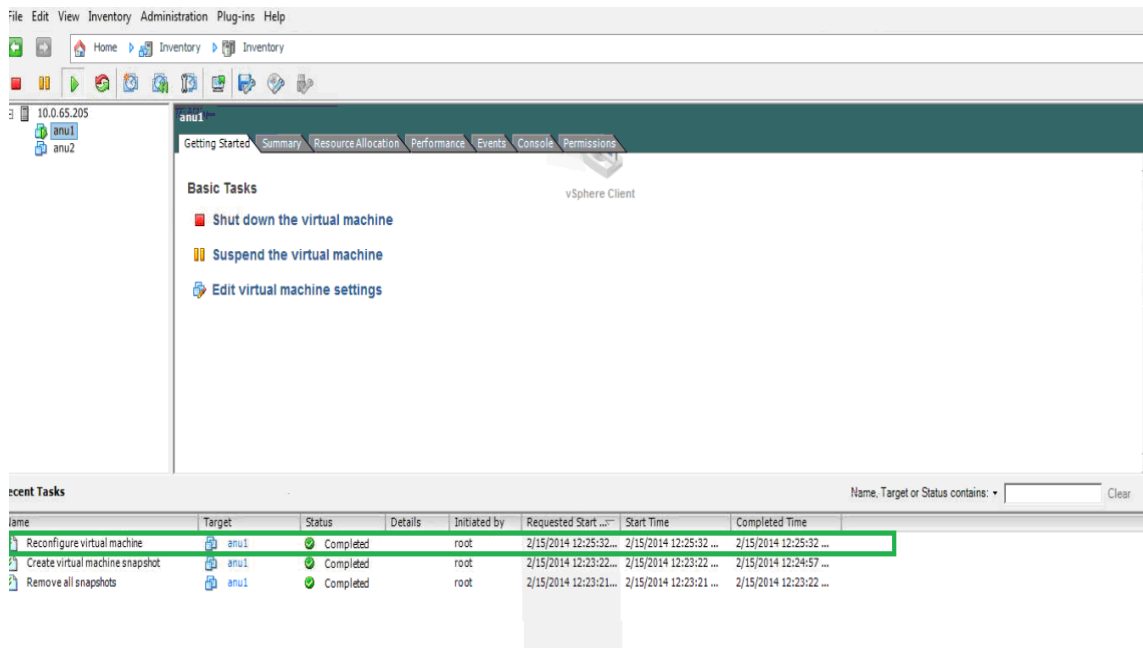
-----

Successfully removed snapshot backup_vm on anul
Successfully created snapshot backup_vm on anul
```

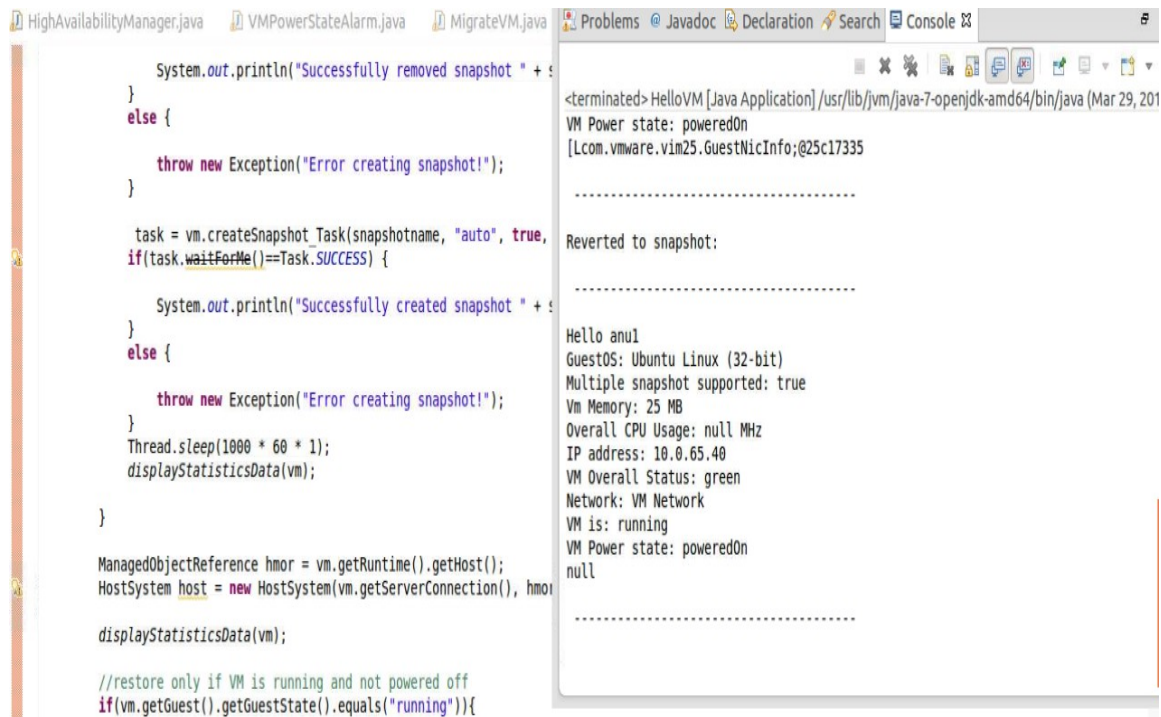
Screenshot 5: Create new snapshot- activity carried out every 10 min – VM



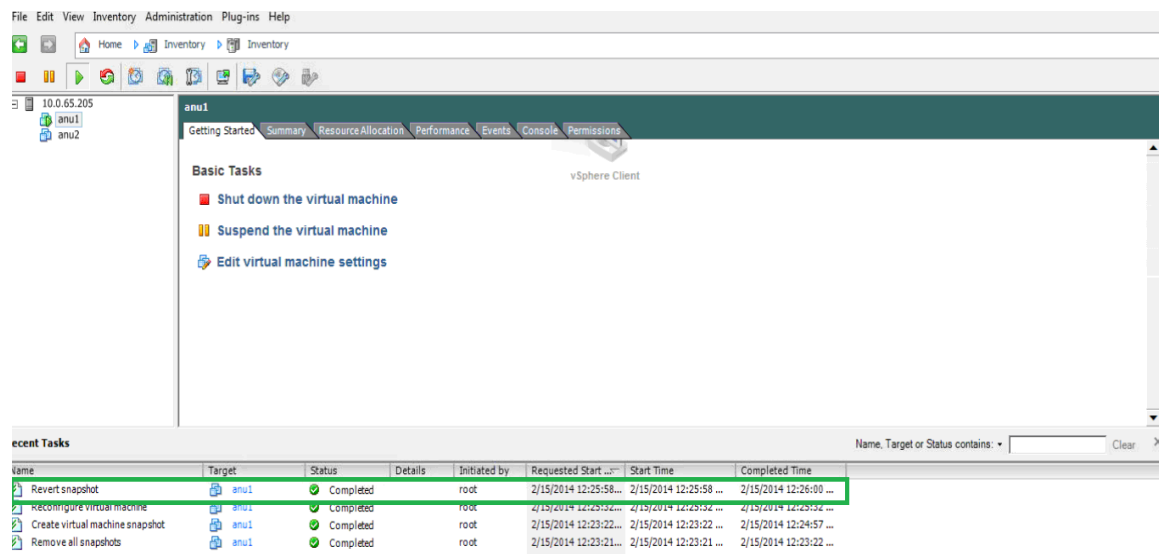
Screenshot 6: Reconfigure network setting- Disconnected- Simulating VM failure.



Screenshot 7: Revert to snapshot API



Screenshot 8: Reversion completed-VM



Screenshot 9: Alarm Manager was triggered when VM was manually powered off.

```
.....  
Hello anu1  
GuestOS: Ubuntu Linux (32-bit)  
Multiple snapshot supported: true  
Vm Memory: null MB  
Overall CPU Usage: null MHz  
IP address: null  
VM Overall Status: green  
Network: VM Network  
VM is: notRunning  
VM Power state: poweredOff  
null
```

```
.....  
Pinging VHost: https://10.0.65.205/  
Triggering Alarm since VM is Powere Off
```

Answers to Questions:

Q 1: In case of failure, what is a good approach during Disaster Management of Virtual Machines-

- o Check the Host first, then the Virtual Machine
 - o Check the Virtual Machine, then the Host
- Justify your answer with sufficient reasons

Ans: Every VM is assigned to a particular host. And one host can have more than one VMs. Now in case of failure either VM can go dead, or host or both!

According to me the good approach would be to check first whether host is alive or dead. The reasons for this are:

1. If the host is not alive then VM won't be alive.
2. In case of disaster recovery checking VM first and then host and finding both are dead, would waste a ping or time. Instead as soon as its known that host is dead, next action can be taken like migration.

Q 2: How many threads do you think are required for the optimal functioning of your Availability Manager? Support your answer with reasons.

Ans: For optimal functioning of Availability Manager in our scenario 2 threads would be sufficient. The reasons for this are:

1. One thread would be incorporated in main body of availability manager. This thread would ping host and VM every one minute to check the heartbeat and sleep otherwise. The same thread would invoke second thread to take snapshot at regular interval.

2. The second thread would be incorporated by snapshot. This thread would be responsible for taking the snapshot. Taking a snapshot is an independent task, hence after this thread is done it will die. Every time a new thread would be created.

Q 3: How did you set and use the Alarms to aid your Availability Manager? Where else can you use the alarms? Explain.

Ans:

In this project alarm is used whenever the VM is powered off manually.

The Availability Manager calls the Alarm when it finds that VM has been manually powered off. It does so by checking the status of the VM which would be “powered off”.

When the alarm is triggered it looks for current state of the VM. If VM is “poweredoff” it changes the color of alarm from yellow to red. Then it powers on the VM for High Availability of the availability manager.

One can similarly use another alarm when snapshot needs to be reverted back on dead VM. This alarm would be invoked when availability manager finds out that host is alive but VM is not alive. The alarm can carry on its task of reverting back to the snapshot in cache.

DISCUSSION: How the features are implemented.

1. Host add/remove mechanism.

The method is:

- a. New object for newhost is created using ManagedEntity.
- b. New object of the type ComputeResource which points to the parent folder of the newhost is created. ComputeResource always has a root ResourcePool associated with it.
- c. String objects for comparison are created.
- d. Next object of type queryVMotionCompatibility is created which has return type as HostVMotionCompatibility. It specifies VMotion compatibility types for a host. QueryVMotionCompatibility investigates the general VMotion compatibility of VM with set of hosts. This would basically check whether the host can be added or not to the cluster.

- e. It then its checked for compatibility, if it fails then error messaged is displayed and if it passes then host is added to cluster and eventually cold migration takes place.

2. The approach used to configure failure detection for each VM.

The approach was:

- a. The thread in availability manager class would ping host every minute to see whether its alive or not. If found alive it would ping VM to see whether alive or not.
- b. On checking the condition whether IP address of VM null or not, indicates whether VM is in running state or not.
- c. If not running next thing checked is the status of the VM, whether manually powered off or not. If its powered off the Alarm is generated and if its not powered off then it indicates that there is some kind of failure. In this condition VM is reverted back to the last snapshot in cache.

3. How were host failures detected.

The thread in Availability Manager would ping the host every minute. If the host wouldn't reply in a minute, it would be declared dead.

4. The mechanism used to convert between the image formats used by the hypervisors.

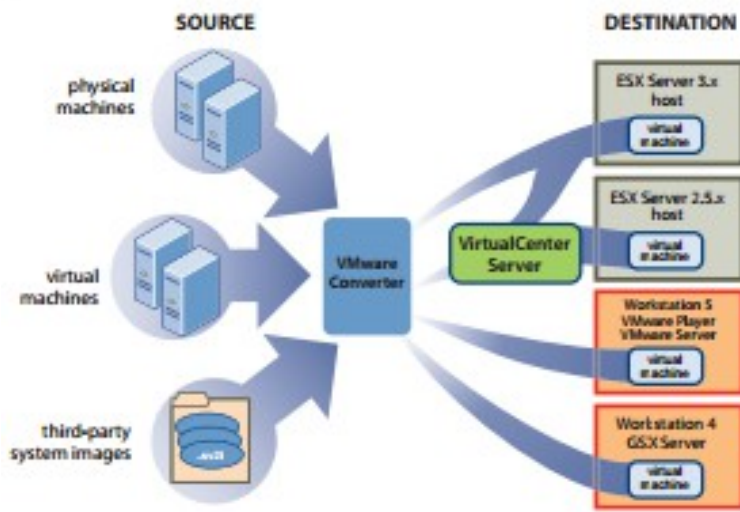
There are few converts available in market for conversion of images between hypervisors. They are:

- Qemu-img tool.
- VMware Converter.

We use VMware Converter which is used to convert between the image formats used by hypervisors. It can be managed through simple, task based user interface that enables conversion between physical machines or third party disk image formats or VMware consolidated Backup images of VM.

This is carried out in 3 steps:

1. Specify the source physical server, VM or third-party format to convert.



2. Specify destination format, VM name and location for new VM to be created.
3. Create/Convert to destination VM and configure it.

It achieves this through sector based copying. It first takes a snapshot of source machine before migrating the data, resulting in fewer failed conversions and no downtime on the source server. It communicates directly with guest OS running on the source machine for hot cloning and has no direct hardware level dependencies.

For cold cloning, the VMware Converter Boot CD provides necessary environment of Windows PE boot, which provides support for latest hardware and is thus able to recognize most physical server systems.

Conclusion:

Lessons learnt:

- The project gave us an opportunity to study issues arising in storing snapshots and migrating the VM on new host in detail.
- The availability manager we built for the lab and the java threads that we incorporated gave us a chance to learn about thread programming.
- Issues arising during cold migration were interesting and gave us lot of insight in functioning of datacenter.
- VI APIs provide an excellent automated way of managing VMware infrastructure and help maintaining their accessibility and maintainability.

Challenges:

- Handling of snapshots and keeping the latest copy in order to save space and keep it less confusing was bit challenging.
- The other challenge I faced was while restoring the snapshot it was difficult to get it up in powered on state and network connected.
- Thread concurrency resulted in lot of exceptions during development when VM would be busy completing previous tasks.

Reference:

Virtual Infrastructure VMware API reference documentation:

<http://www.vmware.com/support/developer/vc-sdk/visdk25pubs/ReferenceGuide/>

Disaster recovery:

http://en.wikipedia.org/wiki/Disaster_recovery#Importance_of_disaster_recovery_planning

VMware Converter:

http://www.vmware.com/pdf/converter_datasheet.pdf

Stretched Clusters and VMware vCenter Site Recovery Manager - Understanding the Options and Goals - T E C H N I C A L W H I T E P A P E R

VMware Infrastructure Architecture:

http://www.vmware.com/pdf/vi_architecture_wp.pdf

Programming in VI JAVA API helpful book:

VSphere Web services SDK Programming Guide

Other sites referenced:

Regarding

<http://docs.oracle.com/javase/tutorial/essential/concurrency/runthread.html>

http://pubs.vmware.com/vi3/sdk/ReferenceGuide/index-mo_types.html

threads-