‘N’ Client Simulation Using User Mode Linux

Suhas Aggarwal
Dept. of Computer Science & Engineering
Indian Institute of Technology,Guwahati
Assam 781039
suhasagg@gmail.com

Abstract
User mode linux (UML) is a open source software that allows to run Linux in a “virtual machine” on top of a physical Linux box. UML allows one to run Linux kernels as user mode processes under a host Linux kernel, giving a simple way to run several independent virtual machines on a single piece of physical hardware. So, with the help of UML, we can simulate large no. of computers (PCs). In this paper we give a methodology to simulate large no. of PCs using UML. These PCs are actually virtual machines running on top of a host. We studied maximum no. of PCs that can be simulated on a host with given RAM.We configure various functions on these PCs such as access to the internet, executing commands on these PCs simultaneously. This simulation is useful in many scenarios especially for experimentation purposes. One application of software will be in testing MPP, a product developed at HSR,Switzerland. It enables users to buy access to the internet at Hotspots. But right now,there is a problem when no. of clients trying to connect simultaneously to a server, exceed a given limit.So, a software to simulate this environment has to be designed to avoid use of large no. of physical computers which is expensive and infeasible.

1.Introduction
User Mode Linux (UML) allows one to run Linux kernels as user mode processes under a host Linux kernel, giving a simple way to run several independent virtual machines on a single piece of physical hardware. Under UML, each of the virtual machines can run its own selection of software, including different distributions of Linux and different kernels. This gives one the ability to have completely customizable virtual machines that are isolated from each other, and from the host machine.

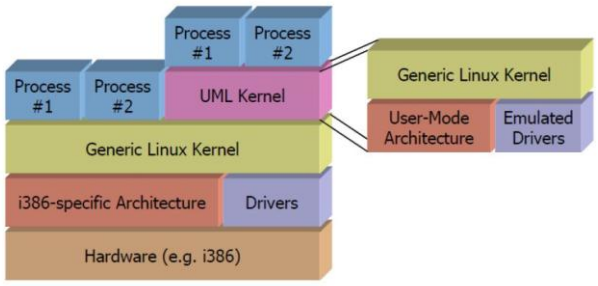


Figure1:User mode linux architecture

A virtual host running with a User-Mode-Linux kernel is just another user process according to figure1, so that even if the virtual machine crashes, the kernel of the host system is not affected.

1.1.UML Setup
Host System Configuration
OS – Fedora Core 10 (32 bit operating system)
Memory (RAM) – 3GB

1.2.Installation
Download the UML kernel and root filesystem to same working directory.Uncompress them:

```
host% bunzip2 linux-2.6.24-rc7.bz2 FedoraCore5-x86-root_fs.bz23.
```

1.3.Initializing a virtual machine using UML

Run UML as follows:

```
host% chmod 755 ./linux-2.6.24-rc7 host% ./linux-2.6.24- rc7 ubda=FedoraCore4-x86-root_fs mem=128M
```

The `mem` parameter simply specifies the amount of RAM that the virtual machine will have. The `ubda` parameter in this command is giving the UML kernel the name of a file to use to create the virtual machine's `/dev/ubda` virtual block device, which will be its root filesystem. The `/dev/ubda` virtual block device is the first block device

of the virtual machine and is analogous to the `/dev/hda` physical block device of the host Linux system.

1.4. Starting multiple UML instances with different root filesystem images

```
host% chmod 755 ./linux-2.6.24-rc7
host%./linux-2.6.24-rc7 ubda=FedoraCore4-x86-root_fs mem=128M
host%./linux-2.6.24-rc7 ubda=Debian-3.1-x86-root_fs mem=128
```

Two virtual machines are initialized, one running Fedora core 4 root file system and another using Debian root file system .

1.5. Starting multiple UML instances with same filesystem image – Copy on write

```
host% chmod 755 ./linux-2.6.24-rc7
host%./linux-2.6.24-rc7 ubda=machine1.cow,FedoraCore4-x86-root_fs mem=128M
host%./linux-2.6.24-rc7 ubda=machine2.cow,FedoraCore4-x86-root_fs mem=128M
```

Imagine that there are several very similar virtual machines. One would like for them to all use the same root file system image, as this would save a lot of space. This is not a good idea, as all of the machines could write to this shared image and cause problems for the other machines. So one would like each virtual machine to have its own file system that it can write to, but this would waste a lot of disk space, since one'd be making separate copies of the file for each machine. UML solves this problem with a feature called Copy-on-write (COW). The idea is that one can create a virtual block device from two files: one that is read-only and contains all of the shared data, and another that is read-write and stores all of the private changes. By specifying `ubdb=cowfile,sharedfile` when invoking the UML kernel, one is creating a `/dev/ubdb` device that writes changes to a file called cowfile, while using the file sharedfile for the much larger shared read-only data.

1.6.Using hostfs to access files on the physical host from the virtual machine

UML provides a few ways for virtual machines to access files on the physical host. The simplest method is to use hostfs. Running the following command within the guest host makes the entire host file system available as /host in the virtual machine:

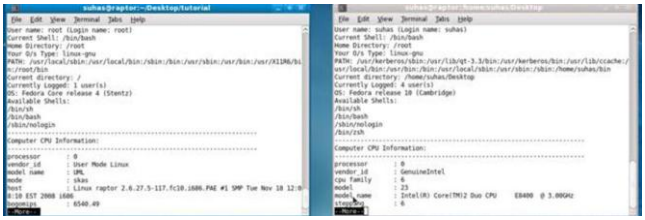
```
mount -t hostfs none /host
```

One example that shows how to mount just one particular directory on the host file system.

```
mount -t hostfs none /home/suhas -o /home/suhas
```

1.7. Running a script inside UML virtual machine

Tested scripts –
1)Script to display 'Hello world'
2)Script to display system configuration



1.8. UML Networking

Bridging

With bridging,the TUN/TAP device used by the UML instance is combined with the host's physical Ethernet device into a sort of virtual switch.The bridge interface forwards Ethernet frames from one interface to another based on their destination MAC addresses.This effectively merges the broadcast domains associated with the bridged interfaces.

1.8.1 Bridging in Practice

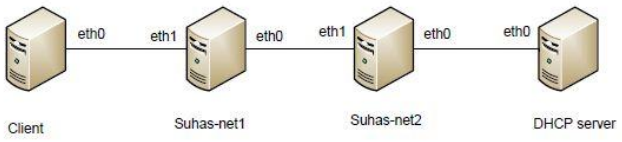


Figure2:Subnet

1.8.2 Bridging

Suhas net -1

Interfaces eth0 and eth1 attached to the bridge.

```
1) ip link set eth0 up
2) ip link set eth1 up
3) brctl addbr br0
4) ip link set br0 up
5) brctl addif br0 eth0
6) brctl addif br0 eth1
```

Suhas net -2

Interfaces eth0 and eth1 attached to the bridge

```
1) ip link set eth0 up
2) ip link set eth1 up
3) brctl addbr br0
4) ip link set br0 up
5) brctl addif br0 eth0
6) brctl addif br0 eth1
```

1.8.3 Configuration details

1) `ip link set eth0 up, ip link set eth1 up` : Interfaces eth0 and eth1 are activated.
2) `brctl addbr br0` :Then, we make bridge and call it `br0`. This creates a bridge with nothing connected to it.
3) `ip link set br0 up` : Bridge is activated.
4) `brctl addif br0 eth0, brctl addif br0 eth1` : Finally,we add eth0 and eth1 to the bridge.

1.8.4 UML Network set up

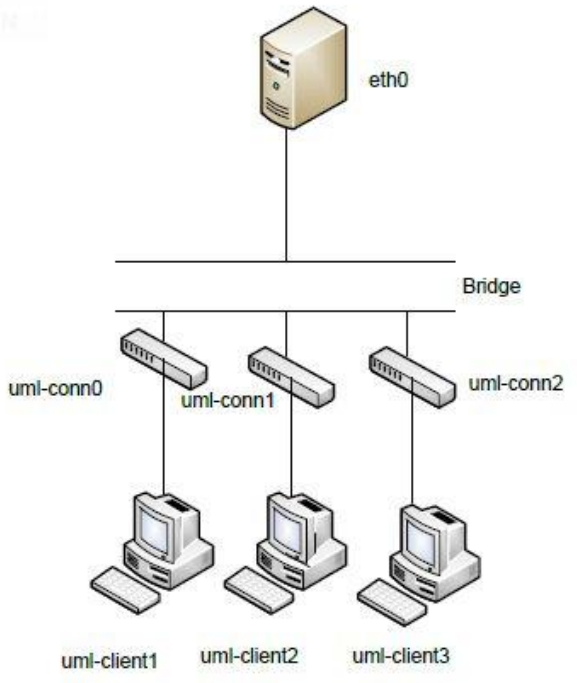


Figure3:UML Network setup

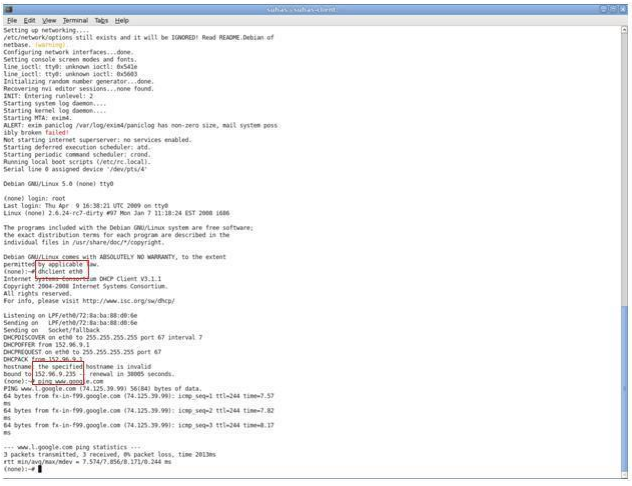
1.8.5 Configuration

1)`ip link set eth0 up`: Interfaces eth0 is activated.
2)`brctl addbr uml-bridge`: We make a bridge called uml-bridge.
3) `brctl setfd uml-bridge 0`: Set the forwarding delay to 0 seconds.
4) `brctl sethello uml-bridge 0`: How often STP "hello" packets with information on the bridged topology are sent.
5) `brctl stp uml-bridge off`: Turn off spanning tree protocol.
6) `ip link set uml-bridge up`: Bridge interface is activated.
7) `brctl addif uml-bridge eth0`: eth0 is added to the bridge.
8) `tunctl -t uml-conn0`: tun device ,uml-conn0 corresponding to uml client1 is created.
9) `ip link set uml-conn0 up`: uml-conn0 interface is activated.
10) `brctl addif uml-bridge uml-conn0`: uml-conn0 is added to the bridge.
11) `tunctl -t uml-conn1`: tun device ,uml-conn1 corresponding to uml client2 is created
12) `ip link set uml-conn1 up`: uml-conn1 interface is activated.
13) `brctl addif uml-bridge uml-conn1`: uml-conn1 is added to the bridge.
14) `tunctl -t uml-conn2`: tun device ,uml-conn2 corresponding to uml client3 is created
15) `ip link set uml-conn2 up`: uml-conn2 interface is activated.
16) `brctl addif uml-bridge uml-conn2`: uml-conn2 is added to the bridge.

Example
Host hosting 3 uml clients.

Host is running Debian Linux 5.0.
UML clients are using Debian 4.0 x86 root filesystem.
UML clients are configured by DHCP server on the network.

UML Client



UML client gets ip address 152.96.9.235 from DCHP server and is able to access internet

2. UML Automation

Scripts -

1)UML start scripts -

```
#!/bin/sh
# start N UML instances on Debian 5.0

UML_EXEC="/home/suhas/Desktop/uml"
UML_DIR="/home/suhas/Desktop/uml/Debian-4.0-x86-root_fs"
MEMORY="16MB"

cd $UML_EXEC

#start UML instance in a separate screen started by screen command
#UMLs are accessible by screen -R H$i

start_instance() {
screen -S H$i -d -m ./linux-2.6.24-rc7 ubd0=$i.cow,$UML_DIR mem=$MEMORY eth0=tuntap,uml-conn$i &
}

start_uml() {

          i=1
          while [ $i -lt 50 ]
          do
          start_instance $i
          i=`expr $i + 1`
          done


}

start_uml
```

2)login script -

```
#!/bin/sh
# Login to UML instances

login() {

        i=1
        while [ $i -lt 50 ]
        do
        screen -S H$i -p 0 -X eval 'stuff root\012'
        i=`expr $i + 1`
        done
}

login
```

3)Network_configuration script

```
#!/bin/sh
# Configure Network interface for UML


ip link set eth0 up
brctl addbr uml-bridge
brctl setfd uml-bridge 0
brctl sethello uml-bridge 0
brctl stp uml-bridge off
ip link set uml-bridge up
brctl addif uml-bridge eth0

i=1
          while [ $i -lt 50 ]
          do
          tunctl -t uml-conn$i
          ip link set uml-conn$i up
          brctl addif uml-bridge uml-conn$i
          i=`expr $i + 1`
          done
```

4)start_network script

```
#!/bin/sh
#UML clients get ip address from DHCP server

start_network() {

                i=1
                while [ $i -lt 50 ]
                do
                screen -S H$i -p 0 -X eval 'stuff dhclient\012'
                i=`expr $i + 1`
                done

}
```

```
start_network
```

5)Transferring file stored in host to UML Clients -

```
#!/bin/sh

change_directory() {

                i=1
                while [ $i -lt 50 ]
                do
                screen -S H$i -p 0 -X eval 'stuff
cd\040/\012'
                i=`expr $i + 1`
                done

}

create_directory1() {

                i=1
                while [ $i -lt 50 ]
                do
                screen -S H$i -p 0 -X eval 'stuff
mkdir\040host\012'
                i=`expr $i + 1`
                done

}

mount_directory() {

                i=1
                while [ $i -lt 50 ]
                do
                screen -S H$i -p 0 -X eval 'stuff
mount\040-t\040hostfs\040none\040/host\012'
                i=`expr $i + 1`
                done

}

change_directory

create_directory1

mount_directory
```

6)Script scheduling using crontab -

```
#!/bin/sh
#Using cron daemon to schedule script on UML
clients

set_crontab() {

                i=1
                while [ $i -lt 50 ]
                do
                screen -S H$i -p 0 -X eval 'stuff
crontab\040/host/home/suhas/Desktop/cron.txt\012'
                i=`expr $i + 1`
                done

}

set_crontab
```

7)Cron file – cron.txt

```
0 10 * * * /host/home/suhas/Desktop/script
```

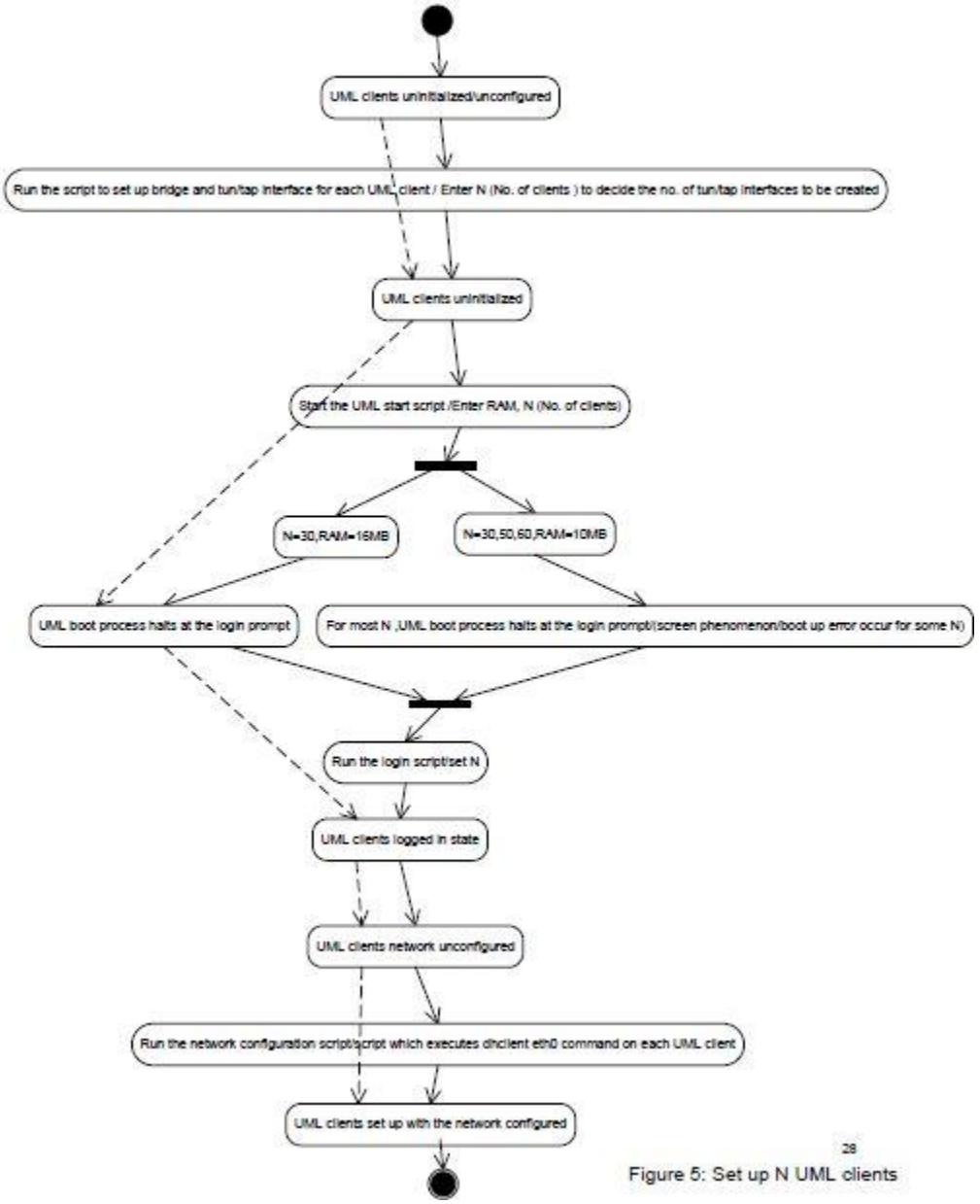


Figure 4: Activity Diagram –Set up ‘N’ UML clients

3. Client limit test

1)Vary RAM allocated to each UML client and check no. of clients getting initialized with the start script

a)Minimum no. of clients being tried to initialized with the start script - 15
Ram size (RAM allocated by the program to each UML CLIENT) - 8MB

No. of UML clients tried	System RAM (Occupied by UML clients (in MB)	Host system RAM(in GB)	Clients initialized
15	120	1.0 (307 MB occupied)	<<15(*some screen phenomenons/ boot up error)
30	240		
60	480		

Table 1

8MB RAM allocated to each UML Client was insufficient to start 15 UML clients with the help of start script.

b)Minimum no. of clients being tried to initialized with the start script - 15
Ram size (RAM allocated by the program to each UML CLIENT) - 16MB

No. of UML clients tried	System RAM (Occupied by UML clients (in MB)	Host system RAM(in GB)	Clients initialized
15	240	1.0 (307 MB occupied)	15
30	480		30
60	960		60 (some screen phenomenons)

Table 2

16MB RAM allocated to each UML Client was sufficient to start 15 UML clients with the help of start script.

c)Minimum no. of clients being tried to initialized with the start script - 15
Ram size (RAM allocated by the program to each UML CLIENT) - 32MB

No. of UML clients tried	System RAM (Occupied by UML clients (in MB)	Host system RAM(in GB)	Clients initialized
15	480	1.0 (307 MB occupied)	15
30	960		30
60	1920		Few clients initialized

Table 3

32MB RAM allocated to each UML Client was sufficient to start 15 UML clients with the help of start script

2)
Find minimum RAM parameter size in script needed to start 15 UML CLIENTs

a)
RAM parameter size needed to start 15 UML clients with start script > 8MB
RAM parameter size needed to start 15 UML clients with start script < = 16MB

Ram size (in MB)	No. of clients tried	Clients initialized
9	15	<15(screen phenomenons / boot up error occurs)
10	15	15
11	15	15

Table 4

Ideal RAM – 10MB,11MB

b)Increase the no. of clients to test the limit

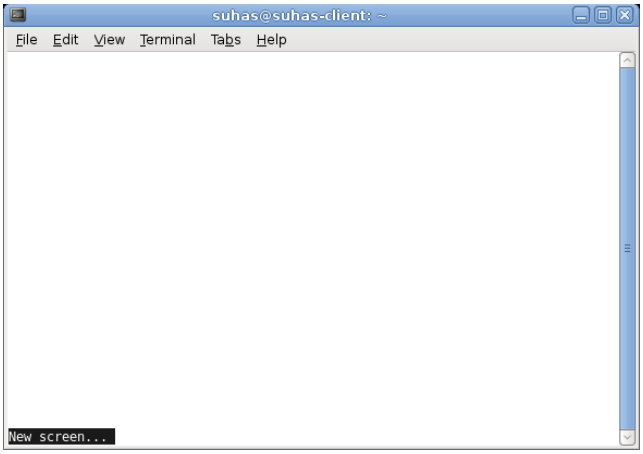
RAM size (in MB)	No. of clients tried	Clients initialized	System RAM occupied by clients (in MB)	Host system RAM (in GB)
10	60	~60(Some screen phenomenons/boot up error)	600	1.0(307 MB occupied)
11	60	<60(More screen phenomenon as compared to above)	660	

Table 5

*Screen phenomenon –

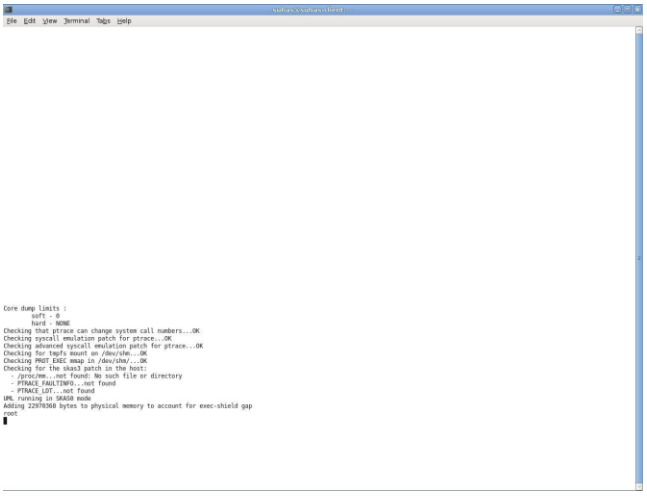
When trying to start UML clients with start script ,screen phemonenon occurs, few no. of clients are getting initialized than desired.

When trying to access ith Uml client by -
screen -R H$i
A new screen starts by screen command, UML process is not loaded on the screen.



The lagging part is that script should be able to start ith screen in which command to start UML client is getting executed but UML boot process should be shown interrupted due to reason such as insufficient host system memory.

*Boot up error



UML boot process hangs.

4. Client Simulation

1)No. of clients tried – 30
RAM parameter size in UML start script – 10 MB
Result- 30 clients started successfully.

2)No. of clients tried – 50
RAM parameter size in UML start script – 10MB
Result – 47 clients successfully started out of 49

3)No. of clients tried – 60
Ram parameter size in UML start script – 10MB
Result - 55 clients successfully started out of 59

General memory consumption of host with 1 GB RAM in start up state is 307MB (697MB free)
No. of clients tried were not beyond 60 as they are started with RAM parameter size in UML start script, 10MB ,implies 60 virtual machines consume 600 MB of system memory

5. Execute command on N uml clients simultaneously

1)Use while loop -

For N=30

```
#!/bin/sh
execute_script() {
i=1
while [ $i -lt 30 ]
do
screen -S H$i -p 0 -X eval 'stuff date\012'
i=`expr $i + 1`
done
}
execute_script
```

As, while loop executes command to start UML client in new screen starting with 1 to N, some delay is probable as iteration proceeds from 1 to N.

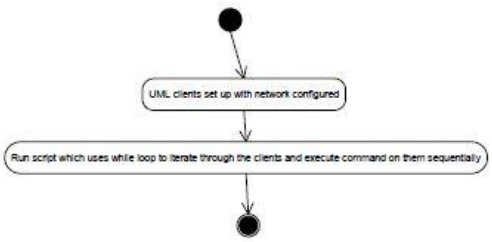


Figure5: Activity Diagram – Execute command on N UML clients using while loop

2)Use cron for script scheduling on N uml clients -
Set same time for script execution on N uml clients.

a) Script scheduling using cron daemon -

```
#!/bin/sh
#Using cron daemon to schedule script on UML
clients
set_crontab() {
i=1
while [ $i -lt 30 ]
do
screen -S H$i -p 0 -X eval 'stuff
crontab\040/host/home/suhas/Desktop/cron.txt\012'
i=`expr $i + 1`
done
}
set_crontab
```

b)
Cron file – cron.txt

```
0 14 * * * date
```

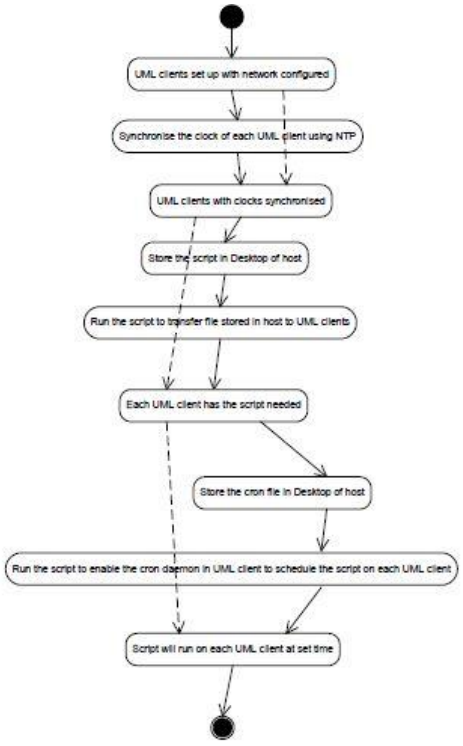


Figure6: Activity Diagram - Execute command on N UML clients simultaneously using cron daemon

6. Acknowledgements

I would like to thank Prof. Beat Stettler and Michael Schneider for their immense guidance and support.

7. Conclusion

In this paper, we showed how to simulate a cluster of 'N' PCs using User Mode Linux. Various functions like network set up ,access to internet, executing commands on different UML clients simultaneously are also configured. We also studied what are maximum no. of UML clients which can be simulated on a given host with given RAM. We also figured ideal RAM parameter in UML start script to start 'N' UML clients. This simulation is extremely useful for experiment purposes where a task need to be performed on large no. of PCs like running scripts on 'N', no. of PCs in parallel, when we have to study server response when 'N' no. of clients connect to it or execute commands on it. When, this 'N' is vey large, using 'N' no. of physical hosts become very expensive, moreover configuring each and every physical PC is very cumbersome as compared to configuring different UML clients running on a single host, which can be configured by transferring configuration script from host to each UML client with the help of script running in host and then running these scripts in 'N' UML clients with the help of script running in host. For testing product MPP, we had to simulate N clients, with each client having a configuration, which try to connect simultaneously to central server,then study the behavior of central server.With the help of UML ,we successfully simulated this environment at a very low cost.

8. References

1)UML project page -http://user-mode-linux.sourceforge.net/
2)Documentation : http://user-mode-linux.sourceforge.net/old/UserModeLinux-HOWTO.html
3) Linux.com : How to run linux inside linux using UML
4)The UML book (c) Jeff Dike - Chapter 7 – UML networking in Depth
5)http://www.howtoforge.com/dhcp_server_linux_debian_sarge
6) Bridging tutorial- http://edeca.net/articles/bridging/index.html
7) DFN 2005 Paper: Advanced Simulation under User-Mode Linux
8)Screen man pages
9)http://fbsdtest.pvp.se/index.php?f=gnu-screen
10)http://www.linuxquestions.org/
How to send a command to screen session?
11)https://www.os3.nl/2008-2009/students/stefan_roelofs/umlauto
12)http://blogs.technet.com/teamdhcp/archive/2006/10/26/when-is-dhcp-nak-issued.aspx
13)http://www.freesoft.org/CIE/RFC/2131/23.htm
14)http://cssh.sourceforge.net/docs/cssh_man.html
15)http://www.akadia.com/services/ntp_synchronize.html
16)http://marc.info/?l=user-mode-linux-user&m=117448523820348&w=2