Acta
Technica
Jaurinensis

Vol. 9, No. 2, pp. 158-170, 2016
DOI: 10.14513/actatechjaur.v9.n2.408
Available online at acta.sze.hu

Advanced Modelling of Virtualized Servers

Á. Kovács[1], G. Lencse[2]

[1,2]Széchenyi István University, Egyetem tér 1. H-9026 Győr, Hungary
Phone: +36 96 613 646
e-mail: [1]kovacs.akos@sze.hu, [2]lencse@sze.hu

Abstract: In the recent years, server virtualization is one of the most important directions of IT infrastructure development. Simulating virtualized infrastructures are unavoidable for designing cloud systems that are customized perfectly for a company. In this paper, we used Opennebula, Haizea and some other tools under public licenses to experiment with. We executed several experiments to examine measurement and simulation capabilities of Haizea and we also tested some typical cluster compilations from the point of view of usability and power consumption. We also tested an open source high availability web server that used virtual machines as computing resource.

Keywords: simulation, Opennebula, Haizea, Cloud Computing, consumption

1. Introduction

This paper is an extended version of our former conference paper [1]

Virtualized infrastructures are spreading around the world. They can optimize the performance resulting in lower TCO (Total Cost of Ownership) and greatly increased manageability of IT systems. The next evolution jump was the cloud computing systems. In this solution, the IT engineer only maintains the hardware, and the end users only rent the infrastructure. The most accepted definition of cloud computing was published by NIST [2] which defines the five essential characteristics of the cloud computing systems.

Modelling these system are one of the most researched topics. Many companies hosts virtual machines to sell them as a service. Predicting how many virtual machines can be operated using a given hardware infrastructure, or how much time it consumes to create a given number of virtual machines is very important to them. Opennebula [3] is a virtual infrastructure engine, which can deploy, monitor and control virtual machines across many physical nodes. Haizea [3] was developed by the University of Chicago. It is an open source lease management architecture which can be used by Opennebula as a regular scheduler. Using these two tools, one can manage physical nodes to automate the generation of virtual machines defined by templates. Haizea can also work as a virtual infrastructure simulator, which can predict (based on a model) how many virtual machines can be safely run in an infrastructure.

We simulated and analysed a system built using a Bladecenter and Haizea in simulation mode as well as Opennebula mode to do experiments and compare them to each other. The remainder of this paper organized as follows. In section 2, a brief introduction is given about the system, what kind of hardware was used for the experiments. In section 3, the modelling with Haizea is illustrated. In sections 4, our experiments are described and results are presented. In section 5, our results are discussed. Finally, our conclusions are given.

2. Test Environment

An IBM Bladecenter was used as the test environment and VMware virtual machine was used as the cloud engine. The specifications were the following:

- Cloud engine: VMware ESXi Virtual Machine (VM version 7) 2 CPUs, 2GB RAM 1x20GB Disks (iSCSI), 1x100GB (NFS), Debian 6.0.4 OS;
- Cloud nodes: HS21 Blade server 2x L5240 Dual-core Xeon, 8GB DDRII ECC RAM, 73GB SAS Disk, 2x1Gb NIC, Debian 6.0.4 OS.

The topology of the test system is shown in Fig. 1. After the installation, we set up Opennebula on the cloud engine virtual machine. It's hardware requirements were minimal, it only consumed several MBs of disk space.

Opennebula supports a bunch of virtualization solutions, including VMware, Xen, and KVM[4].

We used KVM virtualization on the cloud nodes, with access to two networks. One for management and one for Internet access.

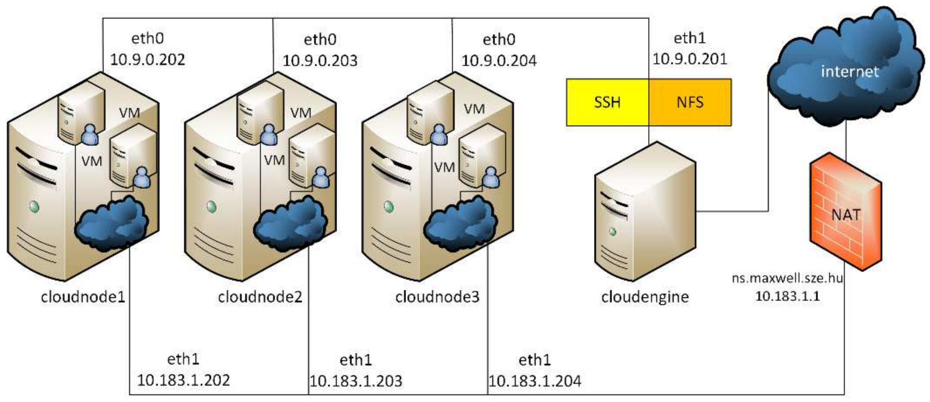


Figure 1. Test system structure

Opennebula uses remote command execution trough SSH tunnel, and for that we had to set up key based authentication between the cloud engine and the cloud nodes. For being able to manage the cloud nodes, we also had to set up the proper drivers for the virtualization solution we have chosen. These were the following:

- im_kvm (Information Manager): this driver gathers information about the cloud nodes e.g. numbers of running virtual machine and available memory;

- vmm_kvm (Virtual Machine Manager): this driver monitors the virtual machines on the cloud nodes;
- tm_nfs (Transfer Manager) this driver transfers the virtual machine images which are defined by the virtual machine template.

For the proper operation, we have to declare a virtual network in Opennebula with pairs of MAC addresses and IP addresses. With this, we are able to add a DHCP server with host directives to manage the Virtual Machines to get the right IP addresses. Opennebula provides shared storage to the cloud nodes, which we implemented by an NFS server on the cloud engine.

3. Modelling in Haizea

Haizea can be used as a scheduler instead of Openebula's *best-effort leases* [5]**Hiba! A hivatkozási forrás nem található.** where a virtual resource is allocated as soon as it is available, or the request is placed in a queue when it is necessary (as no resource is available). Haizea leases contain various information which include the hardware and software resources and the time or availability when these resources can be accessed. Haizea commands are separated into three main blocks:

- *request block*: incoming requests, which can be added manually through CLI (Command Line Interface) or can be read from a special formatted XML file;
- *scheduler block*: this block processes the requests which determine which virtual machine starts or stops and when;
- *working block*: this block sends orders to the simulation (or in *openebula* mode to the Opennebula) to manage Virtual resources.

To run *Haizea* instead of the default scheduler of the Opennebula, the lease must contain the `Haizea` option in an Opennebula request. The simulation can be set up by two files. The first file (see Fig. 2.) contains the performance of the infrastructure and the second one contains the load of the system. The first file also has information about the transfer parameters of the virtual machine images (image size and transmission channel speed) which must be defined based on the real system [6].

In simulation mode, we have to define the resources for the Haizea. The most important ones are CPU and memory parameters and the number of cloud nodes. Also some other things have to be defined such as the clock is simulated or real time. We added four CPU cores for each of the nodes, and 7700 MB memory, because the Operating system that runs the KVM virtualization also uses some system memory from the available 8192MB. After that, we created the XML file that describes the load of the system. The XML file must contain various information for example the amount of virtual resources (CPU, memory, and system image file), and the starting time of the lease. This file also describes the duration of the lease as well. We added all the leases into one file. We defined homogenous load for the simulation. All the virtual machines had 1 CPU core, 1 GB memory and 1 NIC. All the machines working with the same vanilla Debian image we created manually. The request of the virtual machines was sent at the 00:00:00 time. The duration of the virtual machine lifecycle was generally 1 hour for all of them. And the starting time was generated with Poisson distribution shown in Fig. 3. The request were overlapping with each other so it forced Haizea to use best-effort algorithm.

```
[general]
loglevel: DEBUG
logfile: log/Haizea_sim_tilb_1.log
lease-failure-handling: exit-raise
mode: simulated
lease-preparation: imagetransfer
[scheduling]
policy-preemption: ar-preempts-everything
wakeup-interval: 10
suspension: none
migration: no
[simulation]
clock: simulated
starttime: 2013-04-04 11:03:15
resources: 3  CPU:400 Memory:7700
imagetransfer-bandwidth: 60
stop-when: all-leases-done
[tracefile]
tracefile: /srv/cloud/one/sims/sze_tilb_sim.lwf
```

Figure 2. HAIZEA configuration file

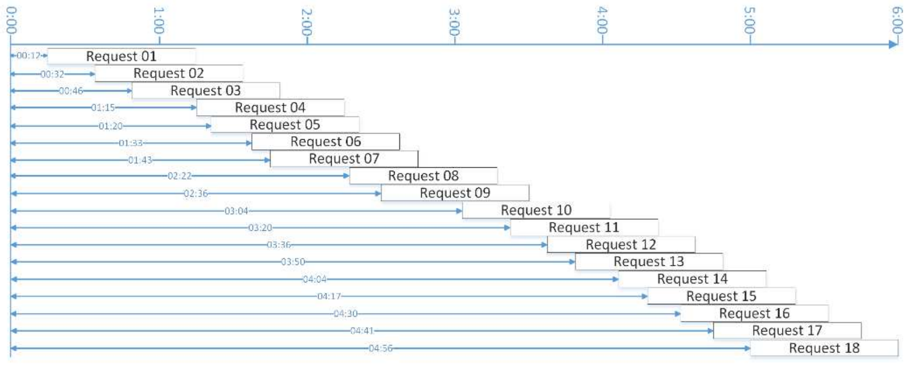


Figure 3. The POISSON distribution of the requests

4. Measurements

4.1. Testing the Virtual Machines

4.1.1. Testing the starting of the virtual machines

For the measuring mode, we generated the same jobs as for the simulation. We created 18 virtual machine description files with the same parameters as we defined in the simulation, only the virtual machine names and the starting times were different in each file. We used a simple bash script to run the measurement. It is important to notice that whereas the simulation finished almost instantly, the execution of the bash script took 5 seconds in average. Because the whole measurement took about 6 hours this difference was negligible. In measurement mode we used Haizea to schedule the virtual machines and Opennebula for deploying them. Only a few changes had to be made in the configuration file because the available resources were given by Opennebula and not manually and of course we had to define the Opennebula host, which one, in our case, was the same machine that executed Haizea. After we started the measurement, the Opennebula was filled up with the requests, and at their starting time when it copied the given virtual machine image file to a separate directory. After that it generated the virtual machine definition file and finally the virtual machine booted with the given parameters. After the experiments we used some Linux based text processing tools (*sed*, *awk*, *grep*) to process the log files for producing the results. We did not deal with the stopping time of the virtual machines, because we modified the shutdown script not to shut down but delete the virtual machines for simplifying the experiments.

Table 1. Difference Between simulation and measurement

HAIZEA	***OpenNebula***	***Difference***
(hh:mm:ss)		
8:29:09	8:33:17	0:04:08
8:49:09	8:52:44	0:03:35
9:03:09	9:07:21	0:04:12
9:22:10	9:26:17	0:04:07
9:37:10	9:41:38	0:04:28
9:50:10	9:53:38	0:03:28
10:00:11	10:04:00	0:03:49
10:21:11	10:24:46	0:03:35
10:39:11	10:43:43	0:04:32
10:53:12	10:56:47	0:03:35
11:21:12	11:25:17	0:04:05
11:37:12	11:40:55	0:03:43
11:53:12	11:56:54	0:03:42
12:07:13	12:10:49	0:03:36
12:21:13	12:24:47	0:03:34
12:34:13	12:37:47	0:03:34
12:47:14	12:51:27	0:04:13
12:58:14	13:01:56	0:03:42
13:13:14	13:16:45	0:03:31
Average:		0:03:51
Std. deviation:		0:00:20

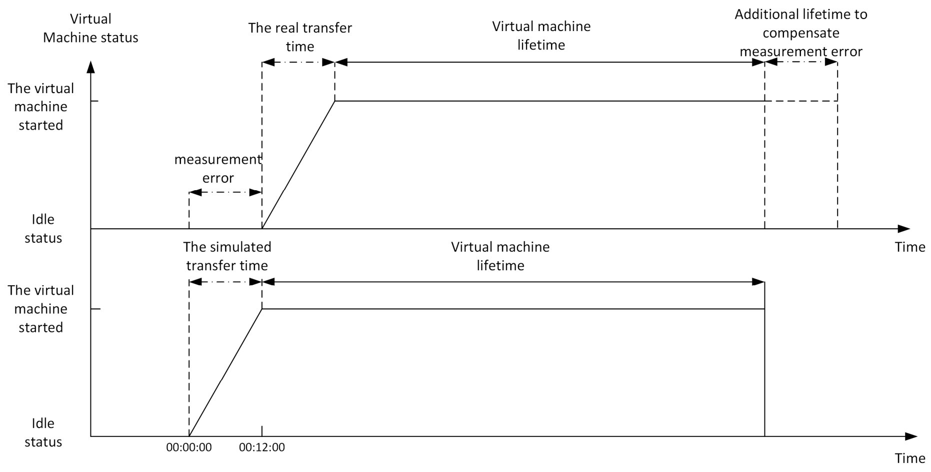


Figure 4. The static difference between the simulation and the measurement

As we can see in Table I there is a static difference between the simulation and the measurement with an average of 3 minutes and 51 seconds. The generated image file size was 4096 MB. It took averagely this time to clone an image file to the predefined place.

4.1.2. Examining the limits of the system

We examined the available recourses both in simulation mode and in measurement mode. We generated a special Virtual machine with only a 40 MB system image, to decrease the deploying load of the system and minimalize the static delay. Based on the parameters we defined in the previous simulation it took about 3 seconds to deploy such a tiny Virtual Machine. We defined a script that generated 24 requests with homogenous 1 GB memory allocations and 1 CPU. In the Linux system, a quad-core CPU is indicated as 400% CPU, whereas in Haizea all the CPUs are specified in percentage but only scaled up to 100%. To be sure that Haizea simulation and measurement uses only one core per Virtual Machine we had to define one core as 25% of the maximum CPU available in one host.

We got the same results in the simulation and in the measurement. The system could not generate the 24th Virtual Machine because the lack of available memory. In the simulation, we defined 7700MB memory per host total of 23.1 GB memory. It could simulate only 23 Virtual Machines the same as the measurement result.

4.1.3. Limits with maximum load

To test the stability of the system, we repeated the experiment with adding high CPU consuming script. To minimalize system image size, we used a simple script that copies random numbers from `/dev/urandom` to `/dev/null`. This script generated high CPU usage without charging the I/O subsystem.

The high load of the system did not cause significant difference in the starting times. Whereas in the experiment without high CPU usage the deploying time was about 3 seconds, in the experiment with high CPU usage it took about 4 seconds per Virtual Machine.

4.2. Analysis of Different Clusters

In this chapter, we compare two different clusters with different CPUs and we also examine which other parameters must be kept in mind.

The two different clusters are made of:

CLUSTER_DUAL:

4pcs IBM HS21 blade modules, each of which containing:

2x Intel Xeon E5160 (Dual Core 3GHz, 65nm, 80W TDP)

8GB DDRII-667 ECC RAM

73GB SAS HDD

CLUSTER_QUAD:

2pcs IBM HS21 blade modules, each of which containing:

2x Intel Xeon E5320 (Quad Core 1,86 GHz, 65nm, 80W TDP)

16GB DDRII-667 ECC RAM

73GB SAS HDD

4.2.1. Performance comparison of the different clusters

First, we executed the tests of the previous chapters. Because of the lack of the information about the host computers, Haizea created the virtual machines using round-robin algorithm. It did not use the potential of the four core CPUs until all dual core CPUs where fully utilized, as Haizea made no difference between dual core and four core CPUs. This is the main reason that Haizea cannot be used to compare different clusters.

After that, we analyzed that how much it costs to execute the same tasks on the different clusters. We used the sysbench software to fully utilize all the CPU cores available in the clusters.

We wrote a simple BASH script to run the test 10 times.

```
#!/bin/bash

for i int `seq 1 10`
do
CPU=`cat /proc/cpuinfo | grep MHz | awk -F \: '{print $2}'`
MAC=`ifconfig |grep HW | awk '{print $5}' | tr ":" "_"`

sysbench --test=cpu --cpu-max-prime=10000 run | grep 'total time:' | awk
'{print $3}'  | cut -c1-6 | tr "." "," >> /home/calc/”$MAC$CPU”.csv
done
```

First, to have a reference, we executed this script native on the blades under Debian Operating systems.

Table 2. The running time of the script in native environment

	DUAL_NAT	QUAD_NAT
	16.829	24.986
	16.829	24.984
	16.829	24.983
	16.829	24.984
	16.829	24.983
	16.829	24.978
	16.829	24.982
	16.828	24.986
	16.830	24.983
	16.829	24.988
average:	16.830	24.984
std. deviation:	0.001	0.004

As we can see, the results are very stable with minimal std. deviation. The proportion between the two cluster running time (16.830/24.984=0.6736) is nearly inversely proportional to the clock rate ratio of the two processor types (1.86GHz/3GHz=~0.62). The minimal (~5%) difference is because we used the same hardware for both CPU types, so the memory and the disk speed were the same on both clusters.

After that, we executed the same scripts, but we used virtual machines. To ensure that the memory is not relevant, we tested the two clusters with virtual machines both with 512MB and 1024MB dedicated memory. The following table shows the execution times of the tests.

Table 3. The execution time of the script in virtualized environment

	DUAL_NATIVE		QUAD_NATIVE	
	16.830s		24.984	
	DUAL_1024MB	DUAL_512MB	QUAD_1024MB	QUAD_512MB
Average:	24.581s	24.068s	36.201s	36.254s
std. deviation	1.0759	**0.5726**	1.2952	**1.0986**
Proportion	DUAL_NATIVE/ DUAL_1024M	**DUAL_1024M/ DUAL_512M**	QUAD_NATIVE/ QUAD_1024M	**QUAD_1024M/ QUAD_512M**
	68%	**~102%**	69%	**~100%**

In this table we can see that the performance with 512MB memory and 1024 memory is almost the same, so we can say that the only important parameter is the performance of the CPU. We note that the performance of the virtual machine is only about 70% of the native environment. This is the optimized performance of the KVM libvirt package.

We can also see the increased std. deviation, this is because when we create a virtual machine it consumes some CPU time.

These tests had very stable results so we created twice as many virtual machines as CPU cores we had to overload the system. The following table shows the execution times of the tests.

Table 4. The running time of the script in virtualized environment (overload)

	DUAL_NATIVE		QUAD_NATIVE	
	16.830s		24.984s	
	DUAL_1024M	DUAL_HCPU	QUAD_1024M	QUAD_HCPU
average	24.581s	43.534s	36.254s	64.339s
std. deviation	1.076	**10.467**	1.099	**13.925**
	100%	**55%**	100%	**56%**

The table shows us some of the previous result to make it easier to compare. As we can see it has almost the half performance that we had before. The running time was greatly increased as much as when the last virtual machine was created the first one is finished with it's job. So the overload was not the same during the measurement.

4.2.2. Examination of the power consumption data

During or tests we continuously logged the power consumption data, and the heat load of the different systems. We used the built in diagnostics of the IBM Bladecenter. Because the chassis of the Bladecenter also uses some energy (due to the power consumption of the built in fans and switch modules) we used 3 different situations to measure these parameters.

1. The consumption of the chassis with blades powered off
2. The consumption of the chassis with blades powered on, but with idle CPUs
3. The consumption of the chassis with blades powered on and fully utilized CPUs

Table 3. The running time of the script in virtualized environment

Dual cluster			Quad cluster		
off	idle	full	off	idle	full
412W	847W	1290W	383W	639W	865W
ΔP	**435W**	**443W**	**ΔP**	**256W**	**226W**

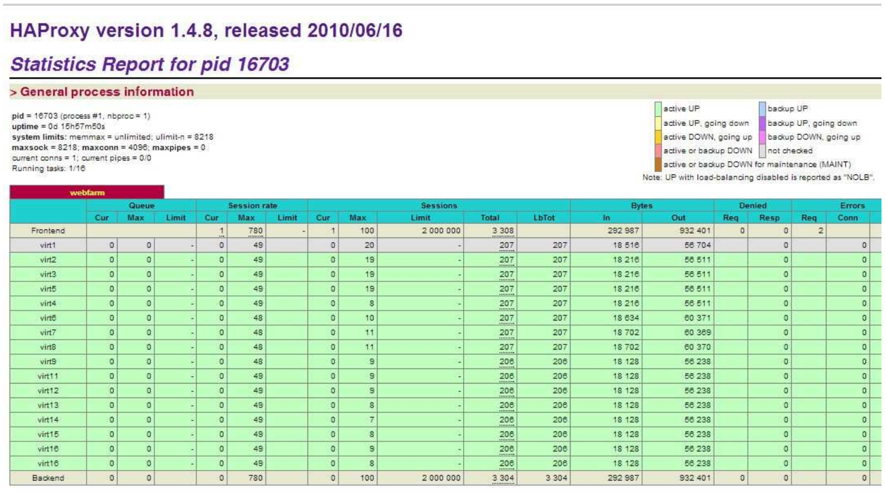
As we can see the two Bladecenter chassis consumes almost the same energy when the blades are powered off. After we powered on the blades, the Dual Core cluster used 200W more energy in idle state. It is understandable, because we operated twice as many blades in the DUAL cluster than in the QUAD cluster. After we fully utilized both clusters, the difference increased about 200W more.

If we use applications on the cluster which do not utilize all the CPU performance then we should use a cluster with fewer nodes with more CPU cores. It will decrease our costs.

4.3. Creating a high availability cluster using virtual machines

To demonstrate the usage of a realistic task, we created a high availability (HA) web server. It contains the DUAL and the QUAD cluster and one of the most current open source load balancer, HAproxy, which was running on a dedicated blade.

After the configuration, we created the virtualized server farm on each cluster. We deployed 16 virtual machines with 1GB memory on each cluster and we installed the "Joomla!" content management system on each virtual machine with default configuration.



HAProxy version 1.4.8, released 2010/06/16

Statistics Report for pid 16703

> General process information

pid = 16703 (process #1, nbproc = 1)
uptime = 0d 15h57m50s
system limits: memmax = unlimited; ulimit-n = 8218
maxsock = 8218; maxconn = 4096; maxpipes = 0
current conns = 1; current pipes = 0/0
Running tasks: 1/16

🟩 active UP	🟦 backup UP
🟨 active UP, going down	🟪 backup UP, going down
🟧 active DOWN, going up	🟪 backup DOWN, going up
🟥 active or backup DOWN	⬜ not checked
🟫 active or backup DOWN for maintenance (MAINT)	

Note: UP with load-balancing disabled is reported as "NOLB".

webfarm

	Queue			Session rate			Sessions					Bytes		Denied		Errors	
	Cur	Max	Limit	Cur	Max	Limit	Cur	Max	Limit	Total	LbTot	In	Out	Req	Resp	Req	Conn
Frontend				1	780	-	1	100	2 000 000	3 308		292 987	932 401	0	0	2	
virt1	0	0	-	0	49		0	20	-	207	207	18 516	56 704		0		0
virt2	0	0	-	0	49		0	19	-	207	207	18 216	56 511		0		0
virt3	0	0	-	0	49		0	19	-	207	207	18 216	56 511		0		0
virt5	0	0	-	0	49		0	19	-	207	207	18 216	56 511		0		0
virt4	0	0	-	0	49		0	8	-	207	207	18 216	56 511		0		0
virt6	0	0	-	0	48		0	10	-	207	207	18 634	60 371		0		0
virt7	0	0	-	0	48		0	11	-	207	207	18 702	60 369		0		0
virt8	0	0	-	0	48		0	11	-	207	207	18 702	60 370		0		0
virt9	0	0	-	0	48		0	9	-	206	206	18 128	56 238		0		0
virt11	0	0	-	0	49		0	9	-	206	206	18 128	56 238		0		0
virt12	0	0	-	0	49		0	9	-	206	206	18 128	56 238		0		0
virt13	0	0	-	0	49		0	8	-	206	206	18 128	56 238		0		0
virt14	0	0	-	0	49		0	7	-	206	206	18 128	56 238		0		0
virt15	0	0	-	0	49		0	8	-	206	206	18 128	56 238		0		0
virt16	0	0	-	0	49		0	9	-	206	206	18 128	56 238		0		0
virt16	0	0	-	0	49		0	8	-	206	206	18 128	56 238		0		0
Backend	0	0		0	780		0	100	2 000 000	3 304	3 304	292 987	932 401	0	0		0

Figure 5. The running HAproxy with 16 nodes

To measure the performance of the HA webserver, we used the industry standard apache benchmark tool. We developed a simple script to automatize the measurements in which we had to change the number of the working nodes.

```
#!/bin/bash
for i in `seq 200 20 400`
do
ab -n $i -c 100 http://10.183.1.44/index.php >> $1
done
```

This script emulates 100 concurrent clients with request from 200 to 400 per seconds.

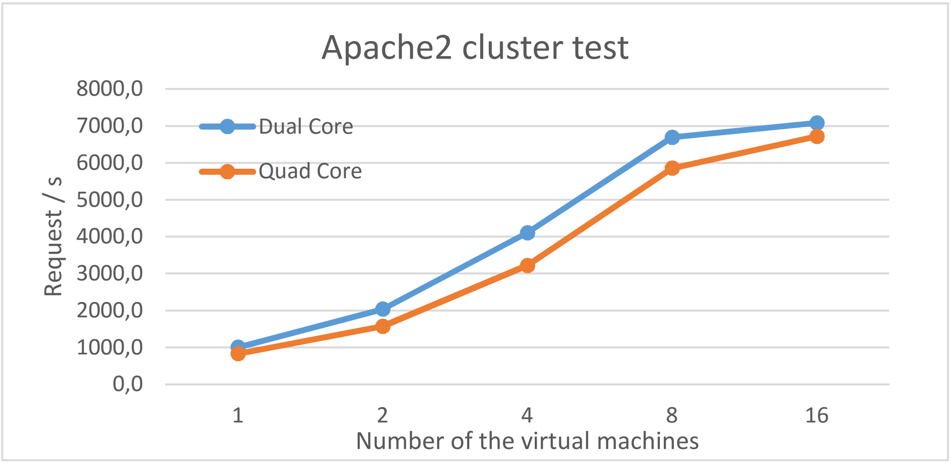


Figure 6. The running HAproxy with 16 nodes

As we can see from the diagram, the two cluster performed almost the same. There was no big difference like it was before in other tests. The web server is not a CPU critical application, so the same number of CPU cores performed very well despite their clock speed was less than that of the dual core CPUs.

5. Discussion of the Results

Haizea can manage two cloning techniques. The first called *image preparation* which means that we define the moment when the virtual machine must be reachable and usable. In that case, Haizea uses the transfer bandwidth which is defined in the configuration file to calculate how much time it takes to transfer image files depending on the size. The second one called *unmanaged* when we only could define the moment when Opennebula starts to clone the virtual machine image file. Unfortunately Haizea supports *image preparation* only in simulation mode but not in Opennebula mode [7]. This the reason of the static delay between simulation and the measurement results.

As Fig. 4 shows, we could compensate the Virtual machine lifetime in measurement mode, to manage the exact 1 hour lifetime. However the knowledge of the time necessary for the image transfer is a prerequisite for this kind of compensation.

In some cases it is not a problem that a virtual machine deployment is delayed a few minutes. For example, when a virtual machine will be a productive unit of a system and we do not want to delete it in the near future. But in some special cases it is necessary to deploy Virtual Machines in time. For example, a lesson must be started exactly at a predefined time in a school. It is unacceptable to delay it because of the IT infrastructure.

Intel launched the CPUs we tested in 2009 [8]. The DUAL core CPU costed 851 USD and the QUAD core CPU costed 690 USD. Based on our measurements, we recommend to buy less hardware with more CPU cores. It is not much slower, in some test it performs almost the same, but TCO is lower. We recommend the DUAL core CPU with higher clock speed only for CPU critical application, but in normal usage for daily tasks, the CPUs with more core and less clock speed are a better choice.

5.1. Recommendation of the Development of HAIZEA

We miss some parameters from Haizea which should be implemented in a future version. First, the image preparation feature in Opennebula mode for better usage. Second, in simulation mode we can't define standard deviation for the deployment of the images. It is unequivocal that when we copy a system image with the size of 4 GB about 30-40 times its copying time will not take exactly the same time.

6. Conclusion

We have demonstrated that the deployment of the images causes a static difference between the starting times of the simulation and the Opennebula mode results.

We have shown that one can minimize the image deployment time for example with tiny images and using a remote file system to store non system files therefore the difference between the simulation and the Opennebula mode can be efficiently reduced.

We have shown that the high CPU load caused no significant difference in the starting time of the Virtual Machines.

We also tested some different clusters with different configurations. We have pointed out that the virtualization layer decreases the performance in some cases about 30%. We also have showed that using same infrastructure with different hardware configuration is essential to get the best performance/power consumption ratio for some tasks. For example, a high availability web server cluster does not depend only on the CPU computing performance. The same CPU core number with lower clock speed from fewer hosts can perform almost the same but with lower power consumption.

References

[1] Kovács Á, Lencse G: Modelling of virtualized servers, Proc. 38th International Conference on Telecommunications and Signal Processing (TSP 2015), Prague, Czech Republic, July 9-11, 2015, Brno University of Technology, pp. 241-245. DOI: 10.1109/TSP.2015.7296260

[2] Mell P, Grance T: SP 800-145. The NIST Definition of Cloud Computing, National Institute of Standards & Technology, Gaithersburg, MD, 2011

[3] Sotomayor B, Montero RS, Llorente IM, Foster I: Virtual Infrastructure Management in Private and Hybrid Clouds, IEEE Internet Computing, Vol. 13, No. 5, pp. 14-22, 2009 DOI: 10.1109/MIC.2009.119

[4] Rafael Moreno-Vozmediano, Rubén S. Montero, Ignacio M. Llorente: IaaS Cloud Architecture: From Virtualized Data Centers to Federated Cloud Infrastructures, Computer, vol.45, no. 12, pp. 65-72, Dec. 2012

[5] Sotomayor B, Montero RS, Llorente IM, Foster I: Capacity Leasing in Cloud Systems using the OpenNebula Engine, in Workshop on Cloud Computing and its Applications 2008 (CCA08), Chicago, Illinois, USA, October 22-23, 2008

[6] Haizea, [Online] http://haizea.cs.uchicago.edu

[7] Sotomayor B: [Haizea] image transfer not working in Haizea 1.0 + Opennebula 1.4, [Online] https://mailman.cs.uchicago.edu/pipermail /haizea/2011-April/000307.html

[8] List of Intel Xeon microprocessors, wikipedia.org [Online] http://en.wikipedia.org/wiki/List_of_Intel_Xeon_microprocessors