TREBALL FINAL DE GRAU

TÍTOL DEL TFG: Cloud computing for wireless systems

TITULACIÓ: Grau en Enginyeria de Sistemes de Telecomunicació

AUTOR: Martí Floriach Pigem

DIRECTOR: Antoni Gelonch Bosch

DATA: 22 de maig del 2015

Titol: Cloud computing for wireless systems
Autor: Martí Floriach Pigem
Director: Antoni Gelonch Bosch
Data: 29 de maig de 2015

Resum

En els darrers anys les tecnologies sense fils estant avançant molt ràpidament provocant una gran demanda de centres de processament de dades. Actualment aquest processament es realitza en la mateixa estació base, però amb el desplegament de les xarxes de fibra òptica es planteja la possiblitat de moure tot el processament a granges de PCs, o Clouds, aplicant el concepte Software Define Radio. Amb aquest nou paradigma s'obren diferents preguntes sobre com s'han de gestionar aquestes noves estructures.
Una de les eines que disposa de funcionalitats adients per realitzar aquesta gestió és l'OpenStack. Així, l'OpenStack, com que és un projecte Open source pensat per implementar Clouds a gran escala ens ofereix una plataforma per concentrar tot el processament en un sol lloc geogràfic.
Desafortunament, el Cloud es va dissenyar per donar servei a servidors d'Internet, per això aquest treball s'orienta en analitzar la viabilitat d'un Cloud per fer processament de senyal i tenir cura de l'execució en temps real del software que implementa el sistema wireless.

Titol: Cloud computing for wireless systems
Autor: Martí Floriach Pigem
Director: Antoni Gelonch Bosch
Data: April 29 rd 2015

Overview

In recent years, wireless technologies are advancing very rapidly causing a strong demand for data processing centers. Currently wireless system processing is done in the same base station, but the deployment of fiber optic networks raises the possibility of moving all that processing to farms of PCs, or Clouds, applying Software Radio concept in all its fullness. With this new paradigm several open questions about how the scheduler or management should be for these new structures arise.
One of the available tools, with suitable functionalities, to perform management tasks in cloud arena is OpenStack. Thus, OpenStack, an open source project designed to implement large-scale Clouds, offers a platform to concentrate all processing in a single geographic location. Unfortunately Cloud system was designed to provide services to Internet servers, so this work is aimed at analyzing the feasibility of a Cloud for dealing with signal processing of wireless waveforms and take care of the real-time performance of software-based wireless implementations.

ÍNDEX

LISTA DE FIGURES

LLISTA DE TAULES

ACRÒNIMS

3GPP	3rd Generation Partnership Project
ACLR	Adjacent Channel Leakage Radio
ADC	Analog to Digital Converter
ALOE	Abstractio Layer and Operating Environment
CFS	Completely Fair Scheduler
Cloud-RAN	Cloud Radio Access Network
CoMP	Coordinated Multipoint
CPU	Central Processing Unit
FIFO	First In First Out
H-ARQ	Hybrid Automatic Repeat Request
HVM	Hardware Virtual Machine
I/O	Input/Output
IaaS	Infrastructure as a Service
ICI	Intercarrier Interference
ICIC	Inter-cell interference coordination
KVM	Kernel-based Virtual Machine
LTE	Long Term Evolution
FLOPS	Floating point Operations Per Second
MIMO	Multiple Input Multiple Output
MACS	Multiply Accumulate Cycles per Second
OFDM	Orthogonal Frequency Division Multiplexing
PaaS	Platform as a Service
PV	ParaVirtualization
RAN	Random Access Network
RR	Round Robin
RTDAL	Real-Time Distributed Abstraction Layer
OESR	Operating Environment for Software-defined Radio
POSIX	Portable Operating System Interface Unix
SaaS	Software as a Service
OS	Operation System
RF	Radio Frequency
SR	Software Radio
USRP	Universal Software Radio Peripheral
vCPU	Virtual Central Processing Unit
VM	Virtual Machine
VMM	Virtual Machine Monitor

INTRODUCCIÓ

Normalment quan comprem o dissenyem un producte ens interessa poder-ne treure el màxim rendiment malgastant els mínims recursos possibles. Però en aplicacions on el consum de recursos depèn d'algun factor (el temps per exemple) és molt complicat d'optimitzar.
El processament de senyal d'una estació base de telefonia mòbil és un d'aquests casos on la càrrega del sistema depèn de l'hora del dia, per tant s'optimitza a partir de l'hora més problemàtica del dia malgastant recursos en altres franges horàries.

La informàtica des de fa uns quants anys té especial interès en aprofitar tots els recursos de la computadora mitjançant la virtualització. Una de les solucions de les quals es parla més últimament és el cloud computing. Però fins fa un parell d'anys era impensable fer el processament de senyal dins d'un cloud, per culpa de que el retard és massa gran.

Però gràcies a l'expansió de la fibra òptica sobren noves possibilitats per les aplicacions de Cloud Computing. Ja que la fibra òptica presenta un retard molt petit entre dos extrems (antena i el centre de dades) i si el processament del senyal fos prou ràpid, seria possible fer una transmissió a través del cloud. Així doncs, podríem concentrar tot el processat en granges d'ordinadors per gestionar el processat alhora que podríem optimitzar els recursos. Com podem veure a la següent figura, els eNodeB desapareixerien de cada antena col·locant un mòdul de RF i un convertidor ADC que interconnectés el centre de dades i l'antena.

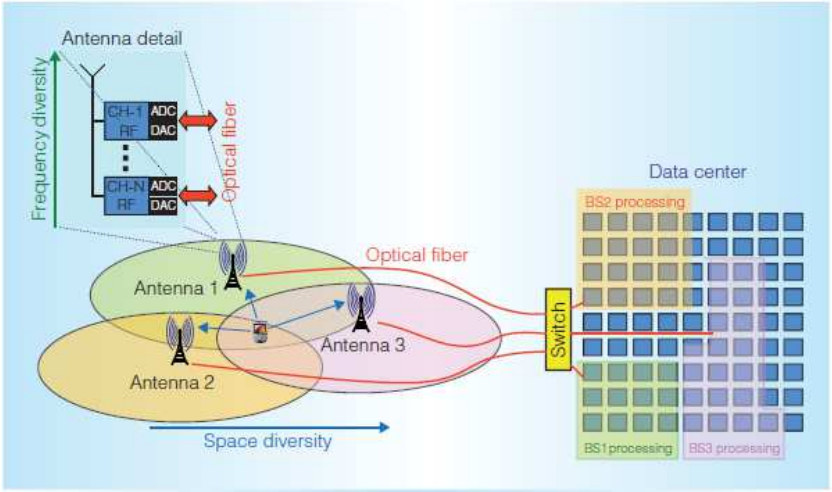

Figura I.1 Exemple de Cloud-RAN [1]

No obstant això, aquestes infraestructures no serien possibles sense l'ús de programari específic de Radiocomunicacions (SDR). Aquesta tecnologia ens permet implementar sistemes de comunicacions ràdio com mescladores, filtres, amplificadors, moduladors/demoduladors, detectors, etc. mitjançant peces de software que s'executen dins d'una computadora. [2]

A més de permetre fer un millor ús dels recursos utilitzant els recursos sobrants per altres aplicacions, el sistema coneix l'estat de totes les antenes. Donant lloc a molts algoritmes per reduir les interferències entre cel·les veïnes.

L'algoritme de Coordinació d'interferències intercel·lular (ICIC) podria funcionar millor en aquest tipus d'escenaris, gràcies al fet que els centres de dades coneixen la xarxa global i, per tant, es pot optimitzar la gestió i assignació de l'espectre, els intervals de temps o antenes. [3]

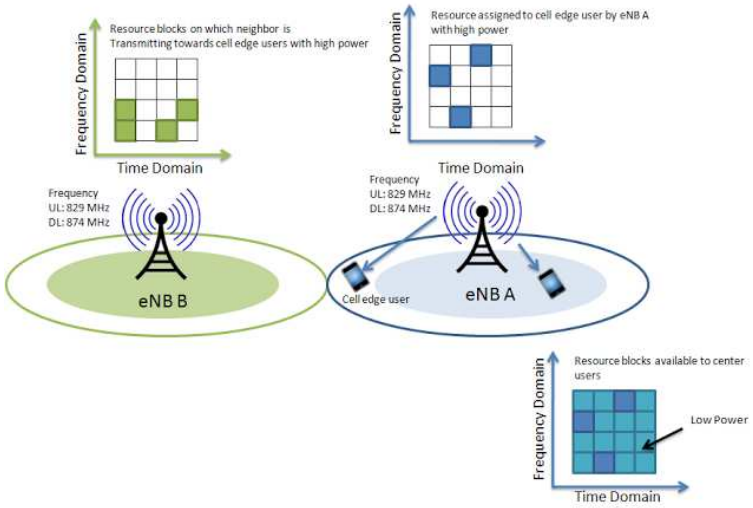

Figura I.2 Representació del ICIC [3]

La coordinació multipunt (CoMP), també es beneficiaria si funcionés dins d'un cloud, millorant el rendiment del sistema en general, utilitzant recursos ràdio amb més eficàcia i millorant la qualitat del servei de l'usuari final. Aquesta tècnica es basa en l'enviament de dues senyals contràries perquè es sumin en contrafase i es redueixi la interferència [3].

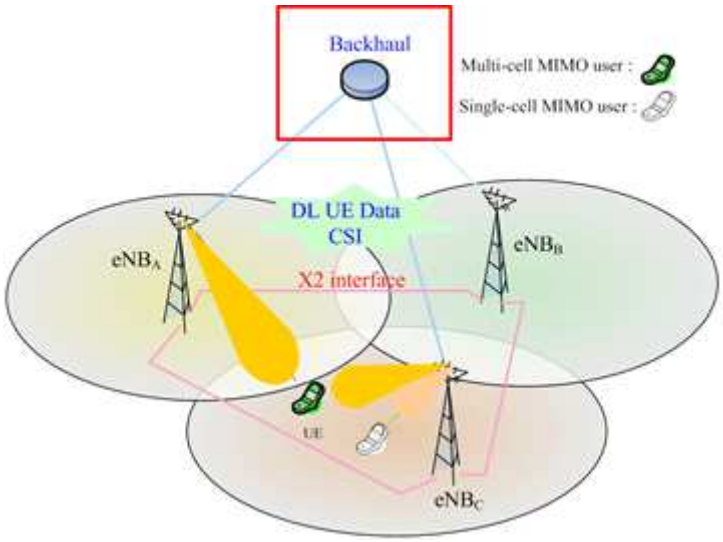

Figura I.3 Representació del CoMP

El CoMP ens ajuda a veure la xarxa com una gran matriu MIMO. Per això necessitem una gran quantitat de recursos per a proporcionar aquest servei i mitigar la interferència.

Tot això fa pensar que les properes estructures de comunicacions sense fils evolucionaran cap a un model centralitzat on el cloud computing tindrà gran importància donant lloc als cloud radio accés network (cloud-RAN).

CAPÍTOL 1. ESTAT DE L'ART

1.1. Long term evolution (LTE)

És un estàndard del 3GPP per a comunicacions sense fils i mòbils d'alta velocitat que s'està desplegant actualment. Es caracteritza per l'ús de Orthogonal Frequency Division Multiplexing (OFDM) en la modulació d'enllaç descendent amb una velocitat màxima al voltant de 326.5Mbps d'acord amb les condicions de l'entorn [4]. La informació de cada usuari és transmesa en resources blocks (RB). Les dades s'han de transmetre ràpidament, introduint retards molt petits per tal no perdre el temps real.
L'LTE mostra un alt nivell de flexibilitat en l'ús de recursos ràdio i depenen de la configuració, els requeriments de recursos de processat creix exponencialment, convertint-se en un bon estàndard per implementar-se sota una infraestructura de cloud computing.

1.1.1. Requeriments dels sistemes de temps real

Un sistema de temps real es descriu com: un sistema que executa una acció amb la suficient rapidesa per garantir un determinat termini d'entrega [5].

A més, la naturalesa repetitiva de les cadenes de processament sense fils suggereix la selecció d'un període d'execució adequat per a tots els processos. Pel que fa a l'LTE, i tenint en compte la durada d'una subtrama (1 ms), un bloc de transport ha de ser transmès i processat en un termini d'execució al voltant d'1 ms.

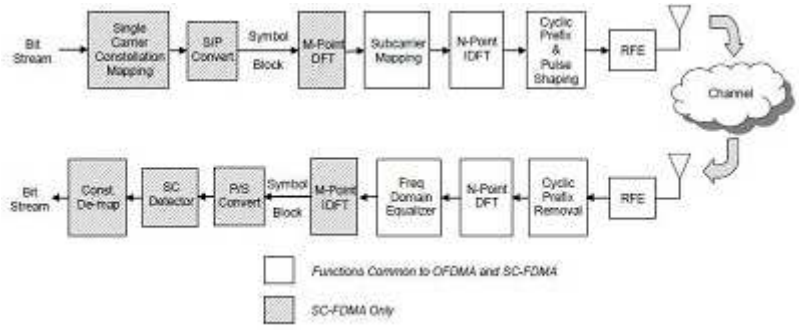

Figura 1.1 Cadena de processat LTE [6]

A més, les especificacions de l'LTE indiquen que d'extrem a extrem la latència no ha d'excedir els 8 ms per a paquets petits, dedicats principalment a les limitacions d'H-ARQ.
Amb tota aquesta informació en ment el temps de processat adequat seria de 2 ms per a la cadena de transmissió i 3.4 ms per la cadena receptora, donant prou temps a altres processos com: transmissió de fibra, la propagació radio

1.2. Virtualització

Abans que ens centrem en el concepte de Cloud Computing és necessari fer una breu introducció a la virtualització, ja que tots dos conceptes estan estretament vinculats.

La virtualització es refereix a l'acte de crear programari mitjançant versions virtuals d'algun recurs tecnològic, com per exemple el hardware, el sistema operatiu (SO), els dispositius d'emmagatzematge i les xarxes [7]. Existeixen diferents tècniques per virtualitzar un SO, els més rellevants són els que descriurem a continuació:

1.2.1. Traducció binaria i Paravirtualització (PV)

Una de les solucions de virtualització és la traducció binaria, que consisteix a emular una instrucció del SO convidat en una instrucció del SO amfitrió mitjançant una traducció del codi binari [8].

La paravirtualització parteix de la mateixa idea, però en aquest cas el SO convidat és qui analitza el codi, per traduir-lo i l'enviar-lo al kernel del SO amfitrió. Per això, és necessari afegir programari addicional en el sistema operatiu convidat. Gràcies a això, la gestió de recursos es dividiria entre VM i el SO amfitrió augmentant l'eficiència, ja que SO amfitrió no ha de fer la conversió del codi [8] [9].

Però no tots els sistemes operatius poden ser paravirtualitzats, ja que hem de modificar el codi font i els sistemes operatius privats, com Windows o MAC SO no ens el proporcionen.

1.2.2. Virtualització completa (HVM)

Una altra solució és la virtualització assistida per hardware (HVM) que utilitza les extensions de virtualització VT-x per Intel i AMD-v per AMD, proporcionant a les VM una nova gamma de privilegis. Així que el SO convidat necessita menys cicles de rellotge per obtenir recursos [8].

Però la virtualització assistida per hardware no és millor que la paravirtualització. Generalment la paravirtualització mostra millor rendiment que la HVM. Per això la millor solució seria un híbrid que agrupa el millor de cada tecnologia.

1.2.3. Híbrid

La virtualització híbrida es basa en la combinació de la paravirtualització i la virtualització completa. En aquesta tècnica, parts del sistema operatiu convidat utilitza la paravirtualització per alguns drivers específic del hardware mentre que el sistema amfitrió utilitza la virtualització completa per altres funcionalitats. Aquest enfocament, sovint ofereix un rendiment superior sense necessitat de fer una paravirtualització completa del sistema operatiu convidat. Un exemple d'això seria la virtualització completa dels convidats del kernel que paravirtualiza les sol·licitud d'I/O quan s'utilitza un controlador especial. D'aquesta manera, el sistema operatiu convidat no ha d'estar plenament paravirtualitzat [9].

1.3. Hipervisor

L'hipervisor o VMM (Virtual Machine Monitor) crea una capa d'abstracció entre el hardware de la màquina física (amfitrió) i el sistema operatiu virtual (màquina virtual convidada). Aquesta capa de programari (VMM) opera, gestiona i arbitra els quatre principals recursos d'un ordinador (CPU, memòria, perifèrics i connexions de xarxa). Aquests recursos s'assignen de forma dinàmica entre totes les màquines virtuals. Això vol dir que podem tenir diverses màquines virtuals executant-se dins del mateix equip físic, compartint el mateix conjunt de recursos sense xocar [10].

D'acord amb la ubicació de l'hipervisor en el sistema, el rendiment de la VM canvia, així doncs podem dividir els hipervisors en dos tipus:

1.3.1. Tipus I: Bare metal o hipervisor natiu

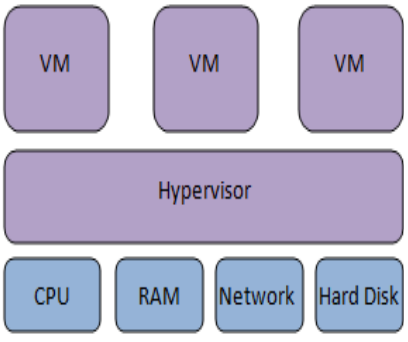

Figura 1.2 Arquitectura dels hypervisor de tipus 1

L'hipervisor de tipus 1 s'executa directament sobre el hardware del sistema, proporcionant una bona eficiència. Normalment se l'anomena "bare metal", "natiu" o "integrat". Aquest tipus d'hipervisors proporcionen un millor rendiment i una major flexibilitat, ja que operen dins del nivell més baix del software. Reduint la sobrecàrrega necessària per executar el propi hipervisor [10].

1.3.2. Tipus II: hipervisor amfitrió

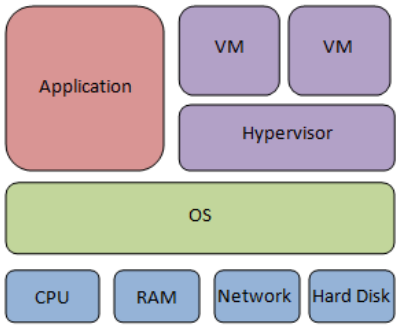

Figura 1.3 Arquitectura dels hypervisor de tipus 2

Els hipervisors de tipus 2 s'instal·len dins del SO amfitrió i provoquen que les VM estiguin molt lluny del hardware en termes de capes de software. Això provoca un empitjorament del rendiment de les VM [10].

1.4. Computació en el cloud

La computació en el cloud és una metàfora que es va desenvolupar a finals de 1990, basada en la utilització de recursos de computació de l'equip físic a través d'Internet. Les aplicacions són executades dins d'un entorn virtualitzat i gestionades a través d'una plataforma web. Els clouds es poden classificar en públics, privats i híbrids [11].

1.4.1. Cloud públic

Un cloud públic és un proveïdor de serveis a través d'Internet. Per tant, el proveïdor de serveis alberga la infraestructura i les aplicacions del client a les seves instal·lacions. Sense que els clients sàpiguen on s'emmagatzema la informació ni tingui capacitat per gestionar-la. Per tant, moltes organitzacions comparteixen la infraestructura física a través del cloud, però les dades de cada organització i l'ús de les aplicacions estan lògicament separades i només els usuaris autoritzats poden accedir-hi [12].

1.4.2. Cloud privat

El cloud privat o cloud empresarial es defineix generalment com un servei de d'ús privat. Això significa que la infraestructura es troba allotjada en una plataforma privada, generalment en el centre de dades del client. El servei es dedica únicament a una organització en particular i no es comparteix amb altres organitzacions. Aquests clouds són creats per les mateixes empreses, utilitzant els recursos existents o adquirir-ne de nous. A través de la virtualització, la companyia espera agrupar tots els servidors dedicats en un de sol. Aquests clouds són només per a ús intern per als empleats de

l'empresa. [12]

1.4.3. Cloud híbrid

Un servei de cloud híbrid és la combinació d'un cloud privat i un cloud públic. En essència, un cloud híbrid es refereix a una organització que manté algunes de les seves operacions internes (cloud privat) i al mateix temps utilitza un proveïdor extern fora (cloud públic).
Així que, en aquest tipus de clouds, l'empresa pot demanar més recursos al cloud públic sempre que sigui necessari, proporcionant una producció òptima en el seu entorn privat [12].

1.4.4. Serveis de cloud computing

Segons el control sobre les instàncies que tenen els clients, es classifiquen els clouds segons el servei que ofereixen:

Software as a Service (SaaS)

El concepte de SaaS s'ha utilitzat durant molt de temps, però potser en els últims anys s'ha definit clarament el què significa. Bàsicament, es tracta de qualsevol servei basat en web. Tenim alguns exemples com ara Gmail Webmail, Dropbox, etc. En aquest tipus de servei normalment accedim a través del navegador sense adonar-nos que estem accedint a un cloud. Tot el desenvolupament, manteniment, actualitzacions i còpies de seguretat són responsabilitat del proveïdor [12].

Platform as a Service (PaaS)

La PaaS és una categoria de la cloud computing que ofereix un entorn que permet als desenvolupadors crear aplicacions i serveis a través d'Internet. En aquest cas, les imatges VM inclouen una gran quantitat d'eines i frameworks pel desenvolupament d'aplicacions, etc. D'aquesta manera els clients només se centren en la creació de software sense preocupar-se per la gestió dels recursos de les instàncies, que es converteixen en la responsabilitat del proveïdor de serveis [12].

Infrastructure as a Service (IaaS)

En un model d'IaaS, el proveïdor acull el hardware, software, els servidors, l'emmagatzematge i d'altres components de la infraestructura en nom dels seus usuaris. Els proveïdors d'IaaS també emmagatzemen les aplicacions dels usuaris i administren tasques que inclouen el manteniment del sistema i còpies de seguretat. Les plataformes IaaS ofereixen recursos altament escalables que es poden ajustar a la demanda.
Altres característiques dels entorns IaaS inclouen l'automatització de tasques administratives, escalament dinàmic, la virtualització d'escriptoris i els serveis basats en les polítiques [12]. Els seus clients paguen una tarifa, normalment per hora, setmana o més. Alguns proveïdors també cobren als clients basant-se en la quantitat d'espai utilitzat per la màquina virtual. Aquest model "pay as you go", elimina la despesa de capital de desplegar el hardware i software a casa.

Alguns exemples d'aquest tipus de clouds són: Rackspace Cloud, Amazon web Service, etc

La següent figura descriu els rols assignats al proveïdor i als clients [13].

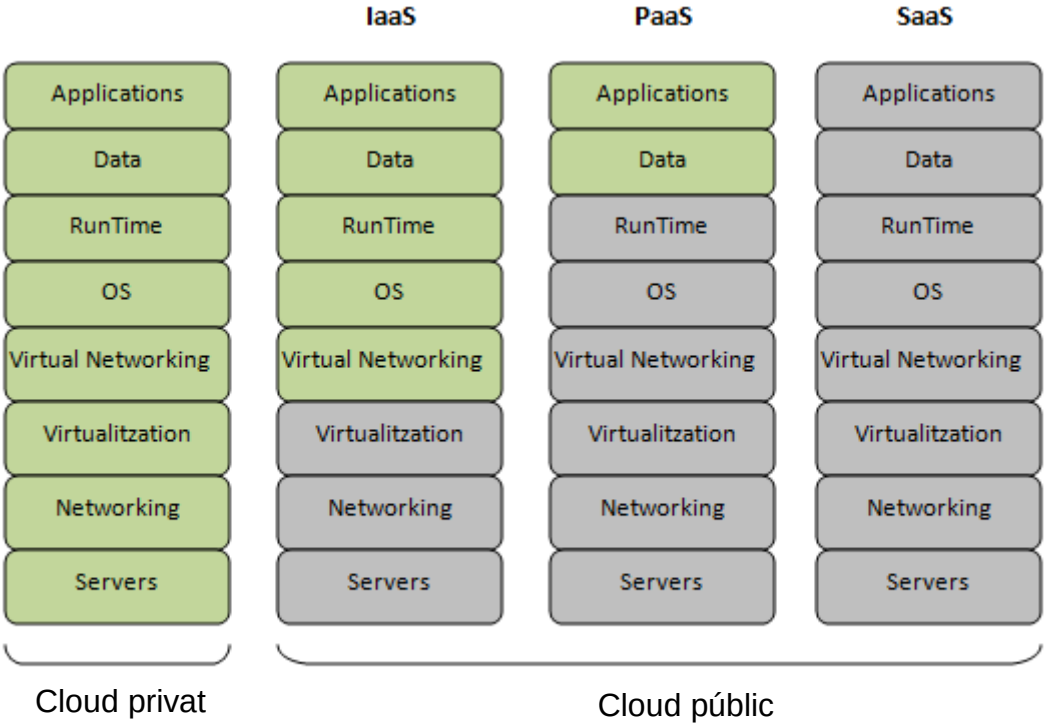

Figura 1.1 Separació de responsabilitats dins del cloud .

1.5. Cas d’estudi

L'objectiu d'aquest projecte és crear un cloud, que es convertiria en públic o privat, d’acord amb els requeriments de processament i gestió de funcionalitats, a través d'un servei d’IaaS.

Un dels principals objectius és assolir les restriccions de temps real imposades pel processament de la longitud d’ona, dins d’un entorn virtualitzat. Els clouds actuals es centren en executar aplicacions a través d’Internet, i per tant els seus requisits en temps real són més relaxats, amb una resposta al voltant de 1s. El nostre objectiu és aconseguir una periodicitat d'una tasca cada 1 ms i una latència màxima de processat a la capa física de LTE de 5 ms.
Per l'altre banda, i com a segon objectiu principal en el projecte, analitzarem les possibilitats de gestió dels recursos que ens ofereix el cloud, especialment els recursos de CPU i la identificació d'estratègies per maximitzar l'eficiència.

En el nostre escenari de treball pensem executar una cadena de processament d’LTE en una màquina virtual enviant el flux de dades a una USRP la qual convertirà el flux de bits a senyal de RF. Creant una granja d’ordinadors, de manera que es pugui gestionar un cloud-RAN a través de la creació d'instàncies, xarxes i connexions amb la USRP [14].

I finalment, tenim la intenció d'assumir el rol d'administrador del sistema i controlar l'estat de les màquines virtuals, actuant en cas de fallada o de sobrecàrrega.

CAPÍTOL 2. PERSONALITZACIÓ DE LES VM

2.1. ALOE

L'ALOE és un framework que gestiona recursos d'una computadora per tal de poder executar aplicacions de SDR. Treballa mitjançant mòduls que s'han d'executar sota uns estrictes requeriments de temps real i a més és compatible amb l'API de la USRP.
Per altra banda l'ALOE, és capaç de treballar fent multi processat. Analitzant les CPU disponibles dins d'una xarxa delegant l'execució dels mòduls a altres CPUs. A més, automàticament distribueix els mòduls, analitzant les prestacions de cada CPU, sense cap configuració per part de l'administrador [15].

Paral·lelament a aquest projecte, un altre estudiant està construint una cadena d'LTE utilitzant l'ALOE. L'objectiu és que quan els dos projectes estiguin acabats puguem executar la cadena d'LTE dins del cloud la qual s'executarà dins d'una VM creada per un hipervisor.

2.2. Hipervisors escollits

Per avaluar el millor hipervisor per executar SDR hem escollit dos hipervisors de codi obert amb diferents tècniques de virtualització.

- KVM que treballa amb la tècnica de virtualització HVM, sent una de les solucions més famoses actualment per sistemes operatius GNU/Linux.

- Xen que treballa com un hipervisor de tipus 1 a través de les tècniques de virtutalitzacio HVM i PV.

2.2.1. KVM-QEMU

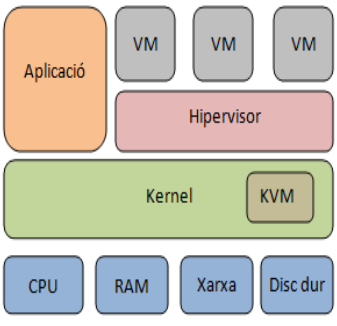

Figura 2.1
Arquitectura de KVM

El primer hipervisor que estudiarem és KVM. Té totes les característiques que busquem en un hipervisor i és altament configurable a través d'un arxiu d'arrencada.

Kernel Virtual Machine base (KVM) és un mòdul del nucli de Linux que permet als programes obtenir recursos de computació utilitzant la tècnica de

virtualització HVM [16].
D'altra banda tenim l'emulador, en aquest cas utilitzem un emulador genèric anomenat QEMU. El mòdul de KVM té un millor rendiment que la traducció binaria però només pot funcionar quan l'arquitectura de la VM és la mateixa que l'arquitectura del sistema amfitrió (GNU). Aquest mòdul beneficia tant el SO l'amfitrió com el SO convidat, ja que els processos del sistema convidat s'envien directament al kernel sense passar per l'amfitrió [17].

2.2.2. Citrix Xenserver

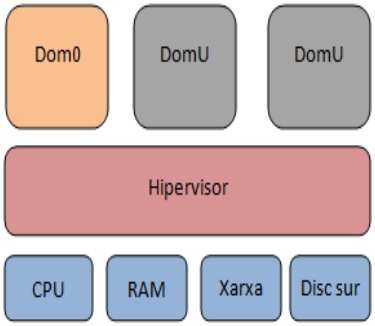

Figura 2.2 Arquitectura de XEN

L'altre hipervisor que té una gran popularitat és Xen. Va ser dissenyat a l'Universitat de Harvard i és el més utilitzat en l'àmbit comercial per una gran quantitat d'empreses com Amazon, Sony o Rackspace.

Xen és l'únic hipervisor de codi obert de tipus 1. És capaç d'executar múltiples VM amb diferents sistemes operatius en paral·lel dins d'una màquina física.

Existent diferents versions de Xen, nosaltres hem provat la versió original Xen i una versió mantinguda per Citrix, anomenada Citrix Xenserver. Vam fer una primera prova per determinar sobre quina versió fer les proves. L'execució d'ALOE dins de Xen era molt més lenta que Citrix Xenserver, per tant, vam optar per treballar amb Citrix Xenserver.

Aquesta versió ens permet crear instàncies que utilitzen la tecnologia de paravirtualització i la virtualització completa a través de l'instal·lació d'un programa. Semblant al guest addiction per virtualBox. A través d'aquest programa podem canviar una VM amb virtualització completa per una VM paravirtualitzada [9].

Les instàncies es creen dins de Dom0, creant una partició dins del disc dur per a cada VM mitjançant contenidors LVM (els quals són redimensionaples [18]), la configuració es defineix en un fitxer xml. Funciona com KVM, però és un sistema operatiu molt més rígid.

2.3. vCPU

Els hipervisors extreuen recursos de la computadora per tal de poder executar les aplicacions de les VM. La quantitat de recursos que extreuen s'agrupen formant vCPUs. Les vCPUs poden ser els recursos de què disposa un o

diversos nuclis o un percentatge de la CPU física. Aquest recursos es mesuren en número d'operacions i són la unitat mínima que poden executar les CPUs.

2.4. Scheduler

L'scheduler és un mòdul del kernel que gestiona les tasques que s'han d'executar i en quin ordre, d'acord amb unes prioritats dins d'un ordinador.

El nostre objectiu és executar SDR en temps real, però no tots els algoritmes d'scheduler es van pensar per aquest tipus d'aplicacions. Així que en les següents seccions analitzarem els algoritmes d'scheduler pels sistemes GNU/Linux [19].

2.4.1. Completely Fair Scheduler (CFS)

Aquest algoritme té com a objectiu maximitzar l'ús de la CPU, permet a l'usuari que interactuï amb la màquina. És a dir, intenta que en cap moment un usuari pugui veure una caiguda en el rendiment, per això es fa servir principalment en distribucions que necessiten el desktop. És el scheduler que ve instal·lat per defecte a les distribucions GNU/Linux .

El CFS divideix els recursos en fraccions de temps, dividint el nombre de recursos de computació pel número de tasques que estan en cua, donat com a resultat el temps en nanosegons en què s'executarà una tasca. Per tant, s'executarà una tasca amb el màxim de recursos de computació durant el temps assignat [20].

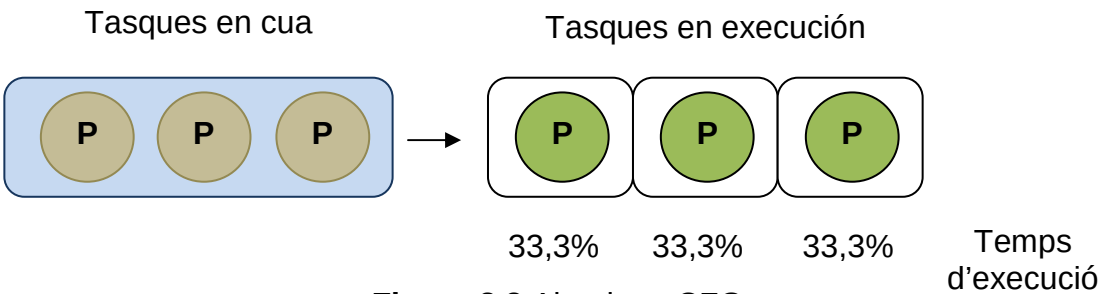

Figura 2.3 Algoritme CFG

Però aquest algoritme no està dissenyat per a aplicacions que s'han d'executar en temps real. Els següents algoritmes que presentarem tenen una millor eficiència per aplicacions en temps real.

2.4.2. First in first out (FIFO)

L'algoritme més senzill és el FIFO, on l'scheduler executa el primer procés que troba dins de la cua, donant-li tots els recursos que disposa la CPU fins que la tasca s'acabi o necessiti fer una crida I/O. Quan hi ha una crida I/O aquest procés torna al final de la cua i espera el seu torn. Així doncs, aquest scheduler executa els processos segons l'ordre en què es troben dins la cua [21].

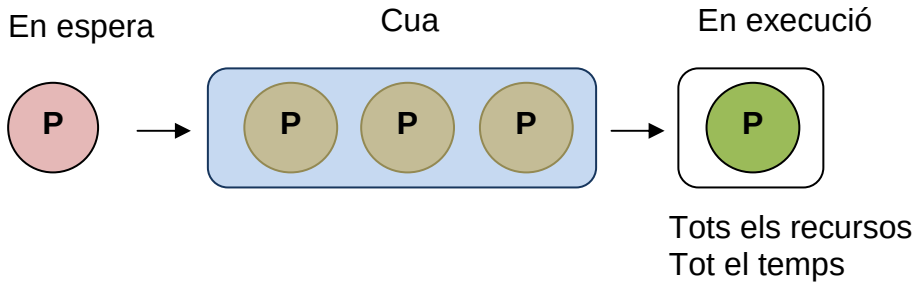

Figura 2.4 Algoritme FIFO

Un punt important d'aquest algoritme és que l'scheduler executa un procés donant-li tots els recursos de CPU. Gràcies a això, és una de les solucions més famoses per al funcionament d'aplicacions en temps real.

2.4.3. Round Robin (RR)

Els processos s'envien a la cua d'execució i s'executen seqüencialment, però es permet que cada procés només s'executi durant una quantitat limitada de temps anomenat interval de temps o quàntum. Si un procés no es completa o es bloqueja a causa d'una operació d'I/O en l'interval de temps corresponent al seu torn d'execució, la porció de temps expira i el procés tornar de la cua d'execució, i haurà d'esperar fins que torni a ser el seu torn [21].

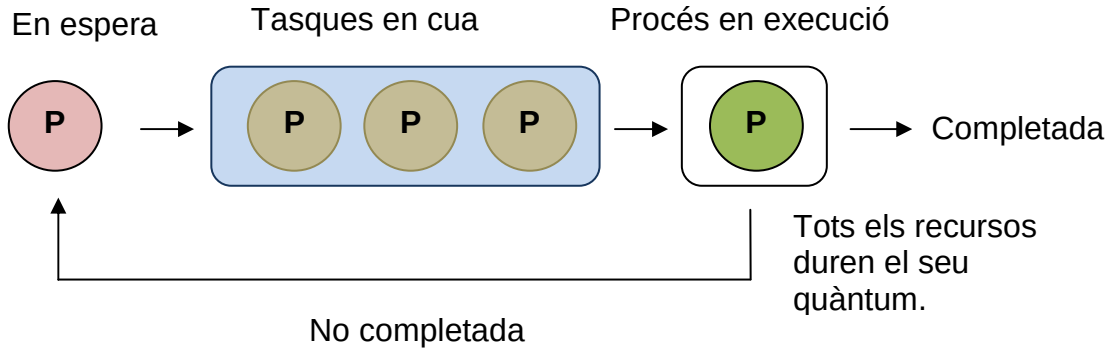

Figura 2.5 Algoritme Round Robin

2.5. Pinning

L'afinitat amb el processador o CPU pinning permet la assignació d'una tasca a una unitat de processament central (CPU) o un rang de CPUs. Per tant, la tasca només s'executarà en una o múltiples CPUs. Cada element de la cua té una etiqueta que indica els seus processadors assignats. Per tant, un cop l'scheduler hagi donat torn d'execució, a la tasca aquesta s'executarà dins del rang de CPUs assignades [22].

El pinning ens permet repartir la càrrega de la computadora entre els diferents nuclis ajudant a que no es sobrecarregui un nucli en concret.

2.6. Libvirt (API de virtualització)

La Libvirt és una API de codi obert que gestiona els recursos i perifèrics que se li assignaran a les VM mitjançant una plantilla. De fet, se la coneix com la API de virtualització, i busca unificar en una sola API tota la gestió dels diferents sistemes de virtualització existents [8].

Aquest template conté totes les opcions per arrencar una VM, per tant l'execució d'una VM és l'execució d'aquesta plantilla. En aquesta plantilla cal indicar el nombre de recursos que volem assignar, com ara el nombre de CPUs virtuals, els recursos de memòria RAM i la xarxa on es connectarà la màquina virtual. En altres camps, indicarem els perifèrics.

L'estructura que ha de seguir la plantilla és la següent:

```
<domain type='qemu'>
  <name>nodea</name>
  <uuid>c7a5fdbd-cdaf-9455-926a-d65c16db1809</uuid>
  <memory>219200</memory>
  <currentMemory>219200</currentMemory>
  <vcpu>2</vcpu>
  <os>
        <type arch='i686' machine='pc'>hvm</type>
        <boot dev='cdrom'/>
  </os>
  <devices>
        <emulator>/usr/bin/qemu-system-x86_64</emulator>
        <disk type='file' device='cdrom'>
        <source file='/home/user/boot.iso'/>
        <target dev='hdc'/>
        <readonly/>
        </disk>
        <disk type='file' device='disk'>
        <source file='/home/user/fedora.img'/>
        <target dev='hda'/>
        </disk>
        <interface type='network'>
        <source network='default'/>
        </interface>
```

```
        <graphics type='vnc' port='-1'/>
    </devices>
  </domain>
```

Figura 2.6 Template de virtualització d'una VM

Entre totes les opcions disponibles de configuració, trobem interessants els elements relatius a l'assignació de recursos de la CPU. La següent figura mostra les diferents opcions de personalització de la vCPU:

```
<domain>
 ...
 <cputune>
        <vcpupin vcpu="1" cpuset="0,1"/>
        <emulatorpin cpuset="1-3"/>
        <iothreadpin iothread="1" cpuset="5,6"/>
        <iothreadpin iothread="2" cpuset="7,8"/>
        <shares>2048</shares>
        <period>1000000</period>
        <quota>-1</quota>
        <emulator_period>1000000</emulator_period>
        <emulator_quota>-1</emulator_quota>
        <vcpusched vcpus='0-4,^3' scheduler='fifo' priority='1'/>
        <iothreadsched iothreads='2' scheduler='batch'/>
 </cputune>
 ...
</domain>
```

Figura 2.7 Personalització dels recursos de CPU

A continuació passarem a explicar cada una de les opcions:

Vcpupin
Es compon per dos variables: vcpu que indica el número de vcpus que tindrà la VM i vcpuset, que indica el rang de CPU físiques on es farà el pinning

Period
Es mesura en microsegons, indica el temps en què la CPU física executarà una tasca de la VM. El rang de valors que pot prendre està entre 1000 i 1000000.

Quota
També es mesura en microsegons, indicant l'amplada de banda (recursos de CPU) que té disponible la VM. El rang de valors que pot prendre va des de 1000 als 18446744073709551.

Iothreadpin
Aquesta opció ens deixa escollir en quin thread s'executarà l'instància. Un thread és una porció de codi juntament amb una pila de la màquina, les quals permeten més d'una tasca el mateix temps [23] [24].

Emulatorpin
El hipervisor en si és un programa, i com a tal gasta recursos envien tasques. A través de cpuset podem indicar en quines CPUs físiques volem que s'executi el hipervisor.

emulator_period
De la mateixa manera que podem limitar els recursos que gasten les VM podem limitar els recursos que gasta el hipervisor. L'emulator_period' és l'homòleg de 'period'.
emulator_quota
Fa la mateixa funció que l'opció 'quota' però aquest cop va destinada a limitar els recursos del propi hipervisor.

Share
Aquesta opció indica la prioritat d'una VM respecte a les altres. Així doncs, una VM amb un valor més gran, tindrà prioritat respecte a una altra màquina virtual si la màquina amfitriona no té suficients recursos per executar les dues VM.

Vcpusched
Aquesta opció assigna un algoritme d'scheduler a un rang de nuclis en concret. Les vcpus són el rang de nuclis i l'scheduler indica el tipus d'algoritme. Existeixen 4 tipus d'algoritmes disponibles, els algoritmes de temps real (fifo i rr explicats anteriorment, secció 2.4) i els algoritmes de no temps real:

- Idle: que executarà les tasques de la VM quan el SO amfitrió no tingui tasques pròpies a executar [25].
- Batch: que executa les tasques de la VM com si fossin tasques de background (executant-se només quan la computadora no té peticions interactives).

Cal mencionar que si el kernel de la màquina física no té un scheduler de temps real, no es podran assignar scheduler de temps real.

A més, podem assignar una prioritat d'execució als processos d'una màquina virtual, on el rang de valor entre el 0 i 99 són processos amb prioritat màxima per sistemes GNU/Linux.

Aquestes són les opcions disponibles per gestionar els recursos de CPU, però no tots els hipervisor poden treballar amb totes i cada una d'aquestes opcions [26].

2.7. Load balancer

Fins ara hem explicat com gestionar els recursos que assigna el SO amfitrió al SO convidat, però també ens pot interessar gestionar la quantitat de càrrega que tenen les vCPU mitjançant un balancejador de càrrega. Aquests són molt utilitzats pels administradors de servidors que necessiten gestionar un gran nombre de peticions [27].

Aquesta eina fa un seguiment de les CPU virtuals, d'acord amb una sèrie d'algoritmes retransmeten un paquet a una instància o un altre segons la càrrega de les instàncies.
Es compon pels següents objectes:

- **Pool**: és l'equilibrador pròpiament dit, que retransmetrà o no un paquet segons el protocol d'Internet que porti.
- **Vip**: és la IP on l'usuari enviarà les peticions.
- **Members**: són les instàncies candidates a processar els paquets.
- **Monitor**: fa un seguiment dels members i informa el pool.

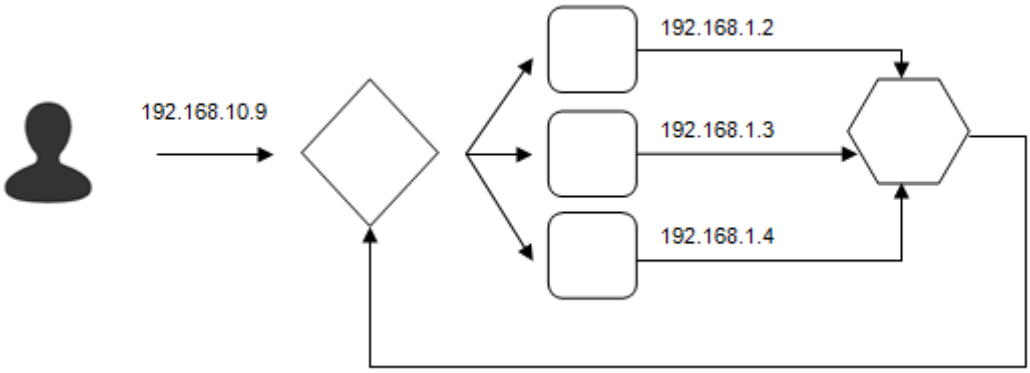

Figura 2.8 Funcionament d'un balancejador de càrrega

En capítols posteriors parlarem de la seva utilitat dins del cloud.

CAPÍTOL 3. OPENSTACK

Un dels temes més importants és l'elecció de la plataforma per crear el nostre cloud-RAN. Existeixen múltiples solucions, a continuació descriurem les plataformes més famoses:

CloudStack
Programari de codi obert dissenyat per implementar i administrar un gran nombre de màquines virtuals a través de diferents xarxes, comportant-se com un tipus de plataforma IaaS, molt resistent i escalable. És utilitzat pels proveïdors de serveis per llogar serveis com un cloud públic i moltes altres companyies amb l'objectiu de crear un cloud privat o com a part d'una solució de cloud híbrid [28].

OpenNebula
Projecte de codi obert desenvolupat com una solució estàndard per crear i gestionar centres de dades virtuals i clouds privats per a empreses. Pertany al tipus IaaS i proporciona solucions per temes de xarxes, emmagatzematge, virtualització i monitorització.

Eucalyptus
Programari de codi obert per a la creació de núvol privat i híbrid compatible amb Amazon Web Services (AWS). L'objectiu és utilitzar la infraestructura virtual per crear recursos de cloud computing amb el propòsit de càlcul, emmagatzematge i creació de xarxes distribuïdes i escalables, donen suport a múltiples hipervisor.

OpenStack
És un projecte de computació en el cloud del tipus IaaS amb un gran equip de desenvolupadors i experts en tecnologia, amb la finalitat de crear la plataforma de codi obert vàlida per qualsevol proposta, tant per cloud públic o privats. Aquest projecte té com a objectiu proporcionar una multitud de solucions per a tot tipus de clouds que són fàcils d'implementar, escalables i rics en característiques [28].

A causa de la impossibilitat d'avaluar totes les plataformes, hem decidit utilitzar l'entorn OpenStack pel desenvolupament del nostre projecte. La raó era que OpenStack té el nombre més gran d'usuaris, seguit d'Eucaliptus, CloudStack i OpenNebula. A més, l'OpenStack també té una comunitat més activa. Aquesta informació es va obtenir per mitjà d'una comparació entre OpenStack, OpenNebula, Eucaliptus i CloudStack basada en les comunicacions entre els membres de les comunitats de desenvolupadors respectius, com llistes de correu i fòrums de discussió [29].

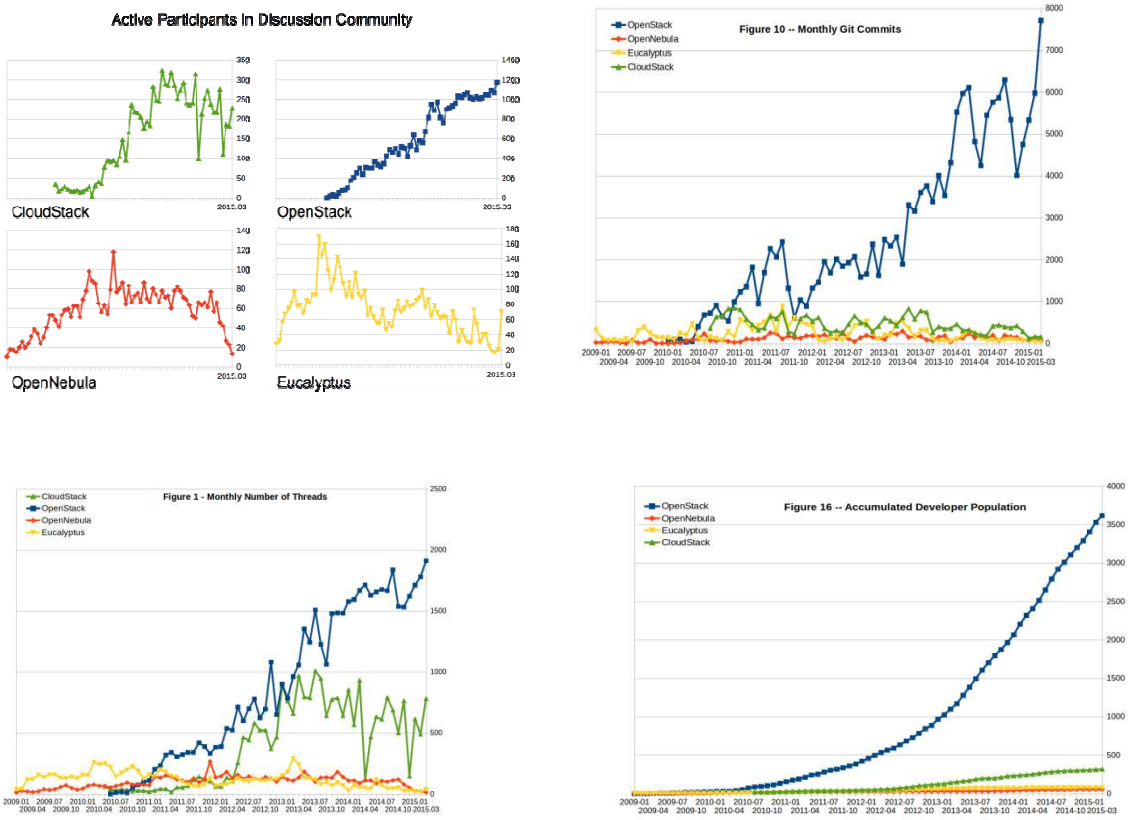

Figura 3.1 Comparativa de la popularitat de les diferents plataformes [29].

3.1. Cloud Computing i terminologia de OpenStack

Com hem discutit a la secció 2.6, el Cloud Computing utilitza els hipervisor per realitzar les virtualitzacions, però utilitza una nomenclatura diferent on els conceptes més rellevants són:

- **Images**: és el software que fa servir la màquina virtual, el SO i els diferents programes instal·lats. Però és un model (objecte), que significa que un cop definit no pot canviar la seva configuració.

- **Flavors**: fa referència al hardware virtual que tindrà la imatge, es defineix a partir d'un número de vCPU i memòria RAM.

- **Instances**: com el seu nom indica, és una instància de l'objecte imatge, que s'executa dins d'un hipervisor amb un flavor concret. Les instàncies són la caracterització de les imatges de tal manera que es puguin caracteritzar instal·lant i desinstal·lant programes. Però un cop tanquem la instància tots els canvis realitzats es perdran.

Resumint, les VM i la instància són el mateix concepte i aquest concepte esta compost d'una imatge i un flavor. Tots els hipervisors fan servir aquests conceptes, però només l'OpenStack els utilitza per separat.

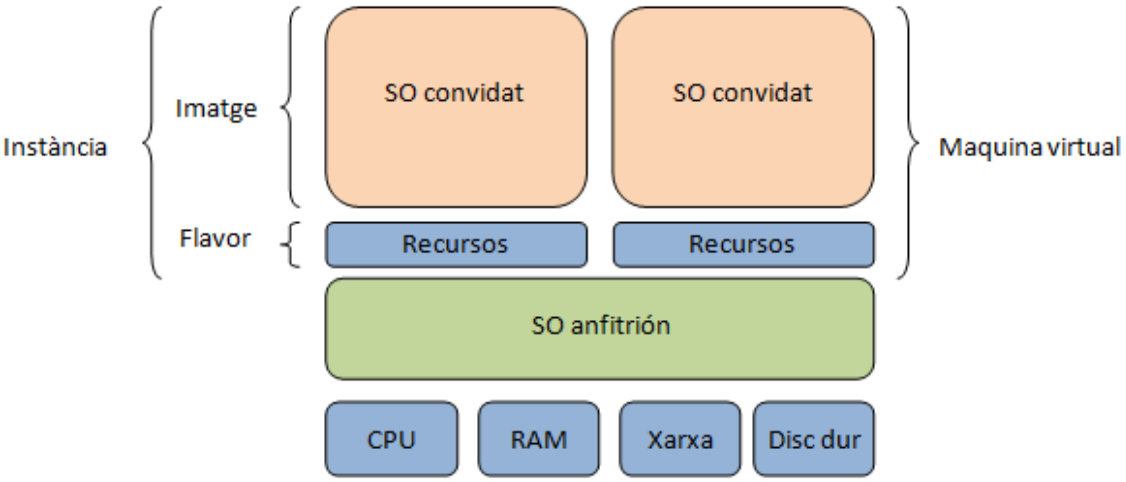

Figura 3.2 Relació entre VM i terminologia de OpenStack.

3.2. OpenStack Service

L'OpenStack es compon de múltiples programes coneguts com a serveis, estan escrits en Python i es comuniquen entre sí, en molts casos utilitzant programes secundaris, com ara bases de dades, etc. Els serveis d'OpenStack són els següents:

3.2.1 Identify service (Keystone)

Aquest servei proporciona control d'accés al cloud i als altres serveis disponibles [30].

Per entendre com funciona aquest mòdul introduirem els següents conceptes:

- **User:** és la representació virtual d'un client o d'un servei d'OpenStack.
- **Credentials*:*** són les claus que identifiquen a un User, mitjançant un nom i una contrasenya.
- **Authentication*:*** es la validació d'un usuari mitjançant les credencials.
- **Token:** un cop l'usuari s'ha identificat ja no es fan servir més les credencials sinó que s'utilitzen els token per fer el API REST full.
- **Tenant:** contenidor utilitzat per agrupar o aïllar recursos. Depenen de l'operador, un tenant pot mapejar un client, un compte, organització o projecte.

- **Service:** és la representació d'un component d'OpenStack.
- **Endpoint:** és l'adreça que identifica a un servei, composta per una URL i un port
- **Role:** defineix el nivell de permisos d'un usuari per accedir als diferents serveis. Per exemple, l'administració té diferents permisos que els clients [30].

3.2.2. Image service (Glance)

Té com a principal objectiu controlar totes les accions referents a les imatges. Registrant, controlant i proveint imatges als usuaris. També permet un repositori on emmagatzema aquestes imatges, acceptant diferents formats per a cada tipus d'hipervisor [30].

Hypervisors	Extension
KVM, Xen	qcow2
KVM	qed
Raw	raw
VirtualBox	vdi
Hyper-V	vpc
VMware	vmdk

Taula 3.1 Extensions de les imatges

3.2.3. Compute service (Nova)

Gestiona el cicle de vida de les instàncies dins d'OpenStack. Les seves responsabilitats són: l'administració de recursos d'administració, així com la creació i eliminació de VM sota demanada [30].

3.2.4. Networking component (Neutron)

L'OpenStack Networking (Neutron) gestiona totes les facetes de xarxa per a la infraestructura de xarxes virtual (VNI) i els aspectes de la capa d'accés a la infraestructura de xarxa física (PNI) dins de l'entorn OpenStack. Està compost per plugins i pot ser implementat per allotjar als diferents equips de xarxes i software, proporcionant flexibilitat a l'arquitectura del cloud [30].

Entre les seves funcionalitats trobem la gestió del router, xarxes, tallafocs, equilibradors de càrrega, VPN, etc.

3.2.5. Dashboard (Horizon)

L'Horizon és una interfície web que permet als administradors i usuaris del cloud gestionar els diferents recursos i serveis assignats [30].

3.2.6. Orchestration module (Heat)

L'Orchestration ajuda als administradors a automatitzar tasques. A través de plantilles, els administradors poden definir un escenari fent ús dels altres serveis. Les plantilles permeten crear la majoria de recursos d'OpenStack, com ara: crear instàncies, crear xarxes, IPs flotants, volums, grups de seguretat... També proporciona una funcionalitat avançada, com per exemple l'autoescalat [30]. Aquest templates es composen de 3 parts:

- **Parameter:** aqui s'indicant tots els inputs. Es consideren inputs totes les opcions que puguin ser variables, per exemple la imatge per fer la instància. Això dóna flexibilitat al template.
- **Resources:** són les accions que realitza heat amb els inputs, per exemple crear una instància. Estan formades per un seguit de comandes predefinides que podeu trobar a la referència [31]
- **Outputs:** és un paràmetre opcional però és molt recomanat posar per tal de comprovar que el template s'ha creat correctament.

3.2.7. Telemetry modul (Ceilometer)

Aquest servei supervisa les instàncies, per tal d'informar a l'administrador si alguna instància es sobrecàrrega o simplement per porta una estadística del cloud. Recull dades de molts tipus com la càrrega de vCPUs, tràfic de bits enviats, espai de disc ocupat, etc. Per poder emmagatzemar aquesta alta quantitat d'informació fa servir una base de dades no SQL, en aquest cas MongoDB. Aquest tipus de bases de dades tenen un millor rendiment que la resta quan l'aplicació té un número molt gran d'entrades i sortides [30].

3.3. Diagrama de serveis

La figura següent descriu la interconnexió entre tots els serveis explicats anteriorment. Per il·lustrar-ho farem un exemple on es fa consulta típica.

En primer lloc el client accedeix a OpenStack a través d'interfície web (Horizon) amb nom d'usuari i contrasenya. Keystone autenticar l'usuari i li dóna accés als recursos i els serveis. Des d'Horizon l'usuari pot accedir a:

- **Neutron:** per administrar les seves xarxes virtuals.

- **Nova:** per administrar les seves instàncies.
- **Glance:** per pujar o eliminar imatges.
- **Heat:** per administrar tots els serveis a través d'una plantilla.

A partir d'ara farem servir el service Heat perquè és més explicatiu i és fàcil de demostrar les interconnexions entre tots els serveis perquè tots apareixeran en escena.
Suposem que volem crear una instància en una nova xarxa amb una alarma si la instància es sobrecarrega. En aquesta situació Heat interactuarà amb Neutron, Nova i Glance per crear l'escenari. Un cop fet l'escenari, Heat configura Celiometer per crear alarmes i fer un seguiment de les instàncies.

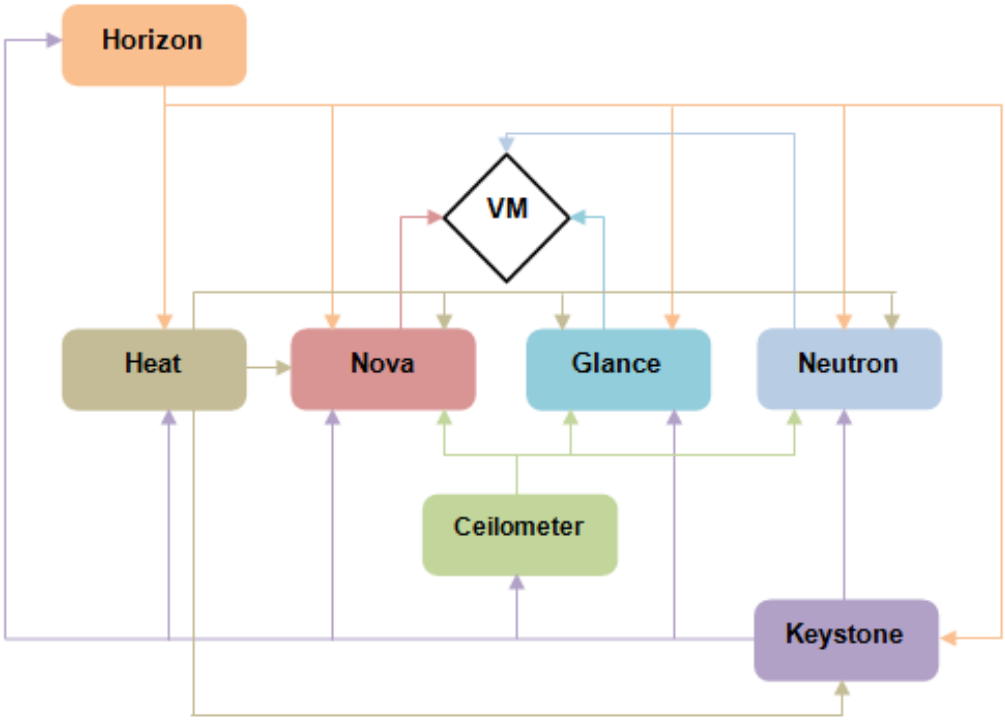

Figura 3.3 Interconnexió entre serveis

CAPÍTOL 4. TESTBED

4.1 Introducció

En aquest capítol avaluarem las possibilites d'OpenStack per executar SDR. Començarem descrivint les tecnologies que hem utilitzat, així com el tipus de cloud que hem desplegat.

Estudiarem la compatibilitat dels hipervisors esmentats en la secció 2.6 respecte bare metal (dins la computadora física sense virtualitzar), per determinar quin té la millor resposta, executant operacions dins d'una franja de temps concreta (1ms).

Per altra banda, hem realitzat dues proves desplegant dos tipus de cloud:

- **Cloud públic:** l'objectiu d'aquesta prova és avaluar les opcions que presenta l'OpenStack, per administrar els recursos d'administració.
- **Cloud privat:** ens centrarem en executar una cadena de processat (LTE) dins d'OpenStack i comprovarem que l'ALOE treballa amb temps real.

Finalment, avaluarem les opcions d'administració de xarxes i projectes que ens proporciona l'OpenStack, intentant donant-li una utilitat dins de la cadena de processat.

4.2. Hardware

Les característiques de les computadores utilitzades són les següents:

Processor Type	Intel I7 Nephalem	AMD Opteron
Model	920	6238
Cores/Threads	4/8	24/24
L3 Cache	L2 2MB i L3 8MB	L3 16MB
Clock	3,67GHz	2,6GHz
RAM	8GB	32GB
Hard Disk	250GB HDD	250GB HDD
Numero	3	1

Taula 4.1 Característiques dels ordinadors

Hem combinat diferents tecnologies. Així com la tecnologia d'Intel permet dos threads per cada nucli, AMD només permet utilitzar un thread per nucli.

Per interconectar els ordinadors hem fet servir dues tecnologies:

- Gigabit Ethernet per la majoria de connexions, tant per accedir a Internet com per interconnectar els ordinadors.

- També hem utilitzat la tecnologia InfiniBand per connectar dues computadores, ja que necessitàvem un delay més petit. Concretament hem optat per un Switch Melanox IS5022 8-port Non-blocking Unamanaged 40Gb/s, el qual treballava sobre fibra òptica fullduplex amb un throwput de 40Gb/s i dues targetes ConnectX®-2 VPI adapter card, dual-port 2, IB 40Gb/s and 10GbE, PCIe2.0 x8.

4.3. Plataforma software: OpenStack

Existeixen dues maneres de desplegar OpenStack, nosaltres hem desplegat les dues opcions però em treballat majoritàriament amb la segona:

- **Devstack layout:** Es caracteritza per desplegar l'OpenStack dins d'una sola màquina física, enfocada principalment al testing.

- **Neutron layout:** Treballa dins de tres o més computadores físiques on cada computadora té un rol específic. És el desplegament recomanat per a clouds d'ús diari.

4.3.1. Devstack layout

Devstack és un script, que mitjançant uns inputs, instal·la l'OpenStack dins d'una sola màquina física. Es va pensar per els desenvolupadors, ja que en només 20 minuts i configurant un sol fitxer podem crear múltiples clouds amb múltiples serveis disponibles.

Els inputs es defineixen a través d'un arxiu local.conf on s'especifiquen els serveis que es volen instal·lar, així com totes les contrasenyes pels usuaris i serveis. A l'annex 4.1 trobareu el arxiu local.conf que hem utilitzat.

Els problemes que presenta aquesta configuració són que el desplegament no és persistent, amb la qual cosa si reiniciem la computadora caldrà arrancar l'script una altra vegada. També és poc configurable, per exemple no podem triar quin hipervisor volem utilitzar. Però el principal problema és que és bastant inestable, provocant que la instal·lació falli segons els paquets instal·lats dins del sistema operatiu.

És una bona eina per realitzar proves o per submergir-se dins del món cloud computing.

4.3.2. Neutron layout

L'altre desplegament és mitjançant tres computadores. Anteriorment, aquesta configuració es feia amb dos ordinadors: Controller i Compute. Però els administradors necesitaven poder administrar les seves xarxes virtual, així que es va incorporar un tercer ordinador destinat a aquesta tasca. Aquesta és la configuració oficial, recomenada per desplegar un cloud d'ús diari. Els rols de cada computadora són els seguents:

Controller node
Aquest node s'encarrega d'administrar el cicle de vida de les instàncies, enviant ordres a cada servei. És l'únic node on els usuaris poden accedir, mitjançant una interfície web. La majoria de serveis estan instal·lats en aquest node.
Sovint instal·lem el mateix servei en el Controller i en una altre node per tal de que el Controller pugui enviar comandes als altres nodes. Es pot veure com una relació entre màster i esclau, on els esclaus són els altres dos nodes.

Network node
Aquest node s'encarrega d'administrar la xarxa virtual d'acord amb els desitjos de cada usuari. A més a més, interconnecta la xarxa virtual amb la xarxa externa per tal de donar accés a Internet a les instàncies. D'altra banda, disposa de múltiples eines per administrar la xarxa com ara: balancejadors de càrrega, VPN, firewalls etc

Compute node
El seu únic objectiu és donar recursos a les instàncies. Si necessitem més recursos, podem afegir més nodes de computació i construir una granja. Un altre dels objectius d'aquest node és monitoritzar les instàncies i informar al Controller node. Finalment, és necessari interconnectar el Computer node amb el Network node perquè les instàncies puguin sortir a la xarxa externa.

La següent figura mostra els serveis instal·lats a cada node, es pot veure que el Controller node té molts més mòduls instal·lats, ja que necessita connectar-se amb els altres nodes alhora que a d'administrar el cloud.

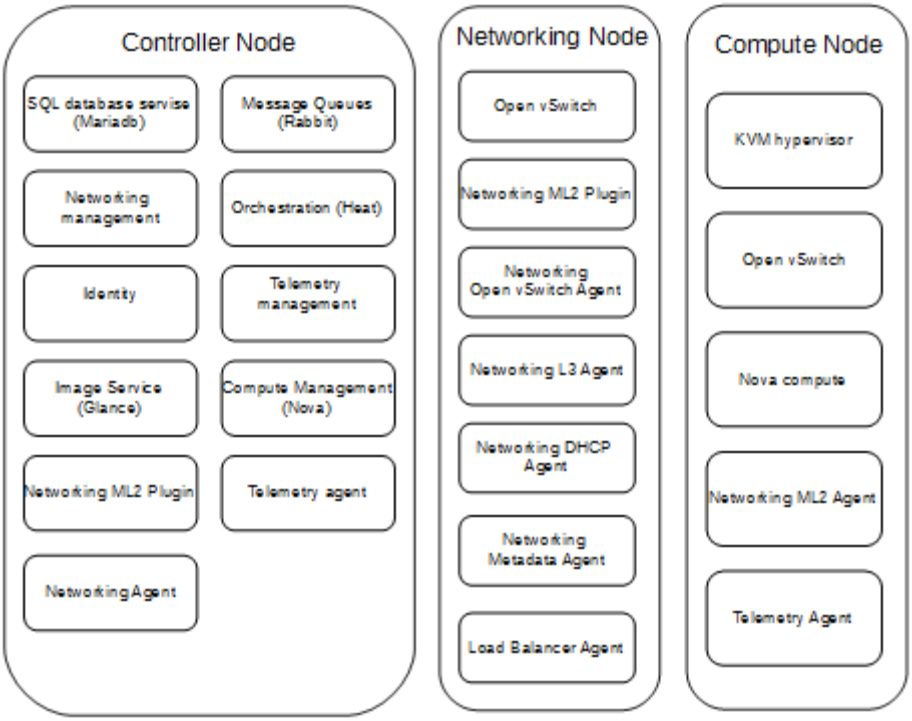

Figura 4.1 Serveis segons cada node

Els layouts s'han creat amb les següents computadores:

Node	Controller	Network	Compute
Característiques	Intel I7 Nephalem 3,67GHz x 4	Intel I7 Nephalem 3,67GHz x 4	AMD Opteron 2,6GHz x 24

Taula 4.2 Característiques i rols de cada ordinador

La figura següent il·lustra el nostre escenari de treball, on es poden veure les diferents interconnexions. Excepte el Controlador, totes les altres computadores accedeixen a Internet per DHCP, per la xarxa 192.168.0.0/16. Com que el Controlador dóna accés al cloud li hem assignat una IP pública per poder administrar el cloud remotament.

Totes les interfícies són gigabit Ethernet excepte el rang de IPs 10.0.1.0/24, que és Infiniband. Aquest rang de IPs interconnecten el Compute node amb el Network node, amb la qual cosa tot el tràfic que generen les instàncies cap a la USRP passa per a aquí. Hem escollit instal·lar l'Infiniband aquí per poder utilitzar la tecnologia SRIOV a les instàncies (explicat a la secció 4.7.2).

Les instàncies surten a la xarxa externa a través del Network node i es connecten amb la USRP. Per això les instàncies i la USRP treballen dins del mateix rang de IPs (192.168.10.0/24). A més a més, necessitem que les instàncies tinguin accés a Internet per fer proves. Amb la qual cosa hem posat una nova computadora que fa el forwarding.

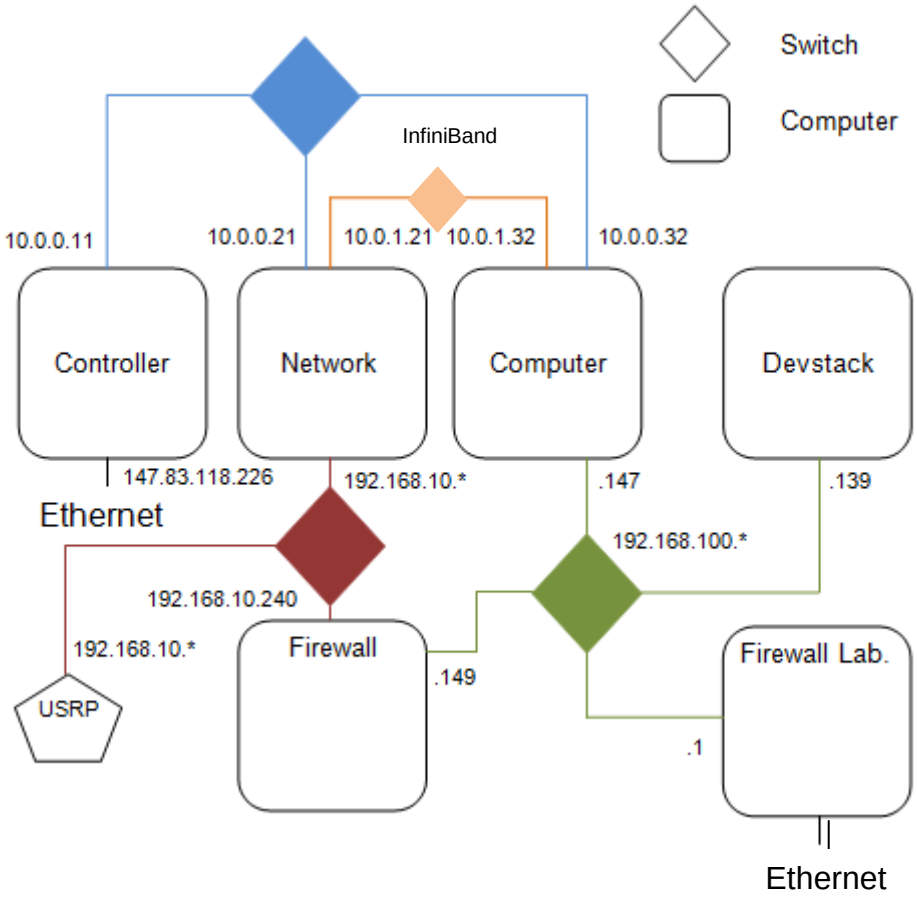

Figura 4.2 Interconnexió dels diferents escenaris.

4.4. Virtual layers

En aquesta secció explicarem com hem configurat les imatges i el hardware virtual (flavor) per realitzar les proves dins dels dos escenaris proposats en l'apartat 4.1.

4.4.1. Virtualització del hardware: Flavors

Per personalitzar les instàncies OpenStack hem utilitzat la llibreria libvirt (explicada en la secció 2.5), encara que no totes les opcions de les que disposa libvirt estan implementades dins d'OpenStack. Com hem comentat anteriorment, volem administrar els recursos de la CPU per crear vCPU. Per defecte, OpenStack crea vCPU amb tots els recursos d'un nucli de la CPU, a nosaltres ens interessa crear vàries instàncies que puguin executar-se dins d'un mateix nucli.

Per administrar els recursos d'un nucli fem servir les opcions de 'period' i 'quota' explicades a la secció 2.5, disponibles dins d'OpenStack. La figura següent mostra com crear un flavor que utilitzi el 50% d'un nucli.

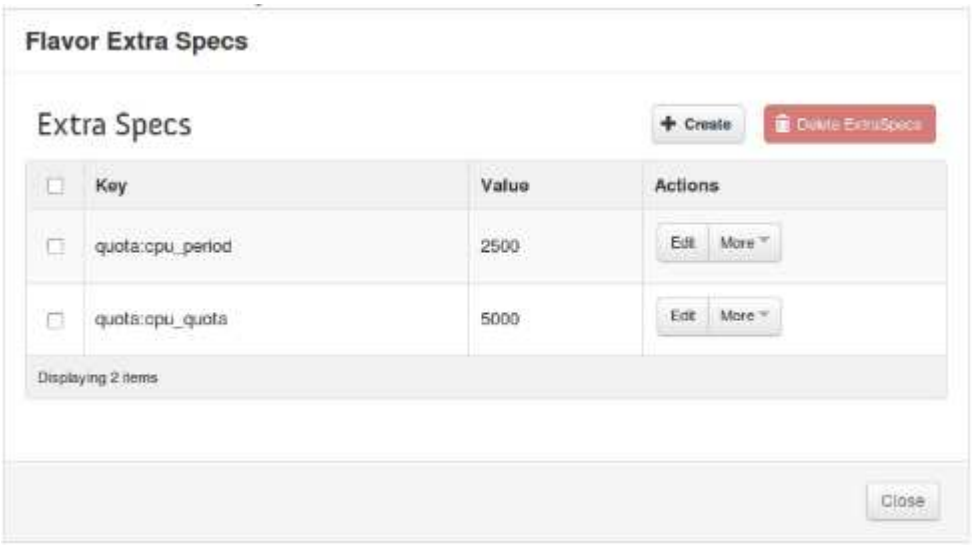

Figura 4.3 Definició dels paràmetres de configuració

Per poder veure els resultants dins de l'administrador de sistemes, és necessari desactivar el pinning. Un cop desactivat podem veure els resultats.

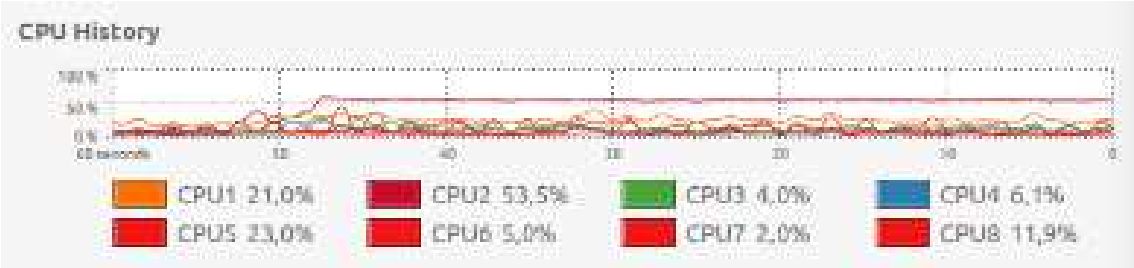

Figura 4.4 Utilització de la CPU segons la configuració descrita anteriorment

Per més informació sobre els flavors dirigiu-vos a l'annex 4.2.

4.4.2. Images

Durant el desenvolupament del projecte, hem utilitzat diverses imatges, algunes descarregades i algunes altres creades per nosaltres mateixos. En aquest apartat explicarem com hem creat imatges que es poden executar dins d'OpenStack, així com els paquets que hem instal·lat a cada imatge.

Nosaltres sempre hem treballat amb Ubuntu, i les nostres imatges no seran una excepció. Hem triat aquesta distribució perquè és la més estesa amb un volum de informació disponible molt gran. Concretament hem treballat amb Ubuntu 14.04 LTS.

Hem creat dues imatges, una amb Ubuntu server sense entorn gràfic per instal·lar el transmissor i una altra amb Lubuntu que disposa d'un entorn gràfic xfce per poder veure la senyal rebuda.
En ambdues imatges hem instal·lat ALOE i els paquets necessaris per interconnectar les instàncies amb l'USRP.

Per realitzar la creació i configuració de les imatges hem utilitzat virt-manager, un front-end similar a Virtualbox que construeix màquines virtuals amb extensió .qcow2, preparades per pujar-les dins del cloud. Per a més informació sobre la creació d'images podeu visitar l'annex 4.3.

4.4.3. CloudInit

Tant les imatges descarregades d'Internet com les imatges fetes per nosaltres, porten el paquet CloudInit instal·lat. És un petit programa que ens permet configurar una imatge un cop aquesta està creada.

En el nostre cas concret ens interessa que quan llancem una instància, automàticament es configuri per enviar mostres a l'USRP i executi ALOE. Entre altres opcions, CloudInit ens permet fer això a través d'un script. Quan estem definint una instància indiquem que volem càrrega aquest script i l'OpenStack executarà l'script dins de la instància, amb permisos d'administrador [32]. A l'annex 4.4 podreu trobar l'script que hem creat.

4.5. Anàlisi de l'hipervisor

Hi ha diferents hipervisors amb diferents característiques. Hem definit una metodologia per provar la seva compatibilitat amb bare metal a través de la detecció d'errors en temps real.

Les proves les realitzarem amb dos mòduls que executen diverses operacions. Anem incrementant el nombre d'instruccions i anotem el número d'errors al voltant dels dos minuts d'execució.

Hem compilat i instal·lat ALOE amb les següents opcions de compilació:

"CFLAGS='-O3 -march=native -mfpmath=sse'"

- CFLAGS: està compost per diferents nivells on el nivell més alt d'optimització és el O3, on es necessita una gran quantitat de temps i memòria per compilar el codi [33].
- March: indica al compilador (gcc) l'arquitectura del processador on es compilarà el codi. Llavors el compilador crearà codi específic per aquest processador. La opció 'native' indica al compilador que intenti descobrir l'arquitectura del processador per ell mateix [33].

- Mfpmath = sse: grup d'instruccions que executen amb múltiples dades en un cicle de rellotge. Es va pensar per a aplicacions que necessiten càlculs intensius.

L'equip utilitzat durant la prova ha estat un AMD Opteron de 2.6GHz x 24, amb una configuració optimitzada d'ALOE treballant dins d'un thread d'un nucli. Els resultats preliminars indiquen que la computadora nativa mou al voltant 2124250000 operacions MAC per segon.

Fent la mateixa prova preliminar però dins d'una VM obtenim els següents resultats segons els diferents hipervisors seleccionats.:

	Bare metal	KVM	Citrix Xenserver
MMACS	2124	2054	2432
MFLOPS	360	300	294

Taula 4.3 Número d'operacions MAC i FLOPS per hipervisor

Primerament, ens guiarem pels FLOPS ja que la realització d'una operació d'aquest tipus és més costosa que no una operació MAC.
La taula anterior indica que hi haurà poca diferència entre KVM i Citrix, ja que tenen una pèrdua de rendiment d'aproximadament 6,6% respecte bare metal.

La gràfica següent mostra la tendència del número de fallades que té bare metal:

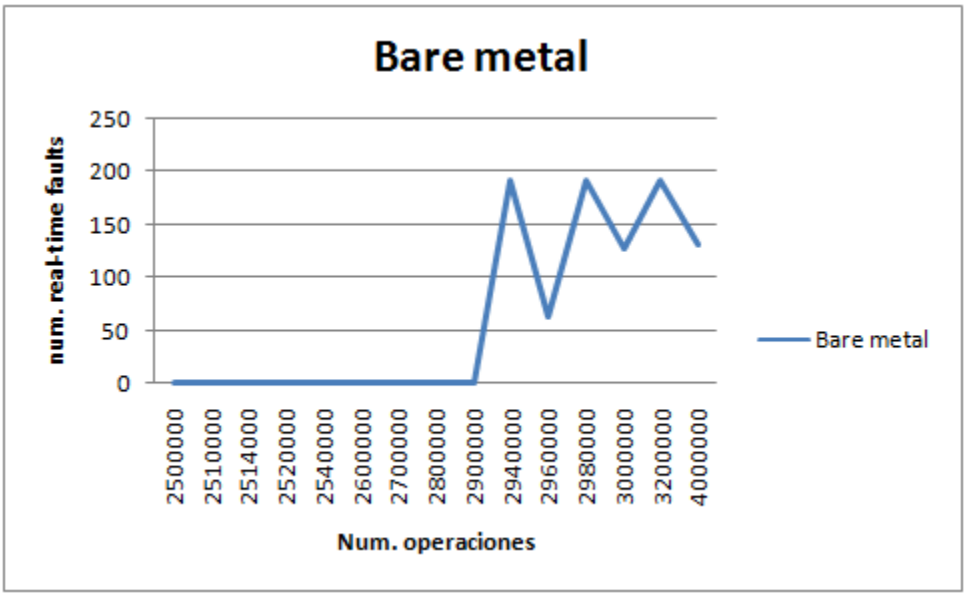

Figura 4.5 Fallades de temps real dins del ordinador natiu

En aquesta gràfica es pot veure el nombre de fallades (eix y) respecte el nombre d'instruccions executades (eix x).

Podem veure que els errors comencen a aparèixer més enllà del valor màxim que suporta la CPU (2600000), això és degut a que el codi està optimitzat.

En la tendència d'error es pot veure que un cop entra en zona d'error, els errors creixen molt ràpidament per després decréixer, provocant un pic. Això es degut a que el processador intenta servir les peticions que li envia ALOE, però finalment es desborda provocant una forma que no té sentit.

4.5.1. KVM

Els resultats mostren que el nombre de fallades del sistema és més gran que el bare metall, però el punt en què comencen a produir fallades està molt a prop. El gràfic mostra que el comportament de KVM és semblant al comportament de bare metal, però que finalment el sistema es desborda.

Creiem que el sistema intenta respondre a la demanda de recursos però que finalment té tants paquets en cua d'espera que el sistema es desborda.

En el gràfic següent podem veure una comparació entre bare metall i KVM.

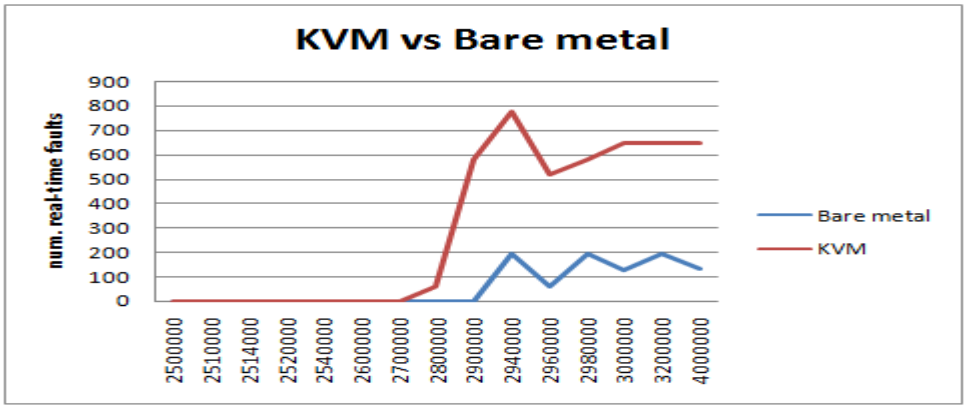

Figura 4.6 Compatibilitat KVM contra bare metal

Respecte a la gràfica es pot concloure que els resultats són bastant comparables al Bare metall. Només presenta un reduccié del rendiment del 6,9%. D'acord amb la documentació, KVM hem d'esperar una pèrdua de compatibilitat al voltant de 5%. Els nostres resultats són una mica pitjors, degut a que el nostre benchmark no és tan precís.

4.5.2. Citrix Xenserver

El Citrix Xenserver és un hipervisor de tipus 1 amb la qual cosa esperàvem un rendiment superior que KVM. Però els resultats obtinguts mostren que la compatibilitat amb bare metal és del 86.2%. Un 10% menys que KVM i molt pitjor que els resultats públicats a la web oficial.
Hem treballat amb dos tipus de virtualització HVM i PV. Els resultats són molt semblants, per realitzar les gràfiques hem fet servir la PV. Com es pot observar a la figura 4.7, comencen a aparèixer les primeres fallades de temps real al

voltant dels 2500000 instruccions. Amb un comportament molt particular on tant punt apareixen els primers errors els sistema de desborda i deixa de funcionar.

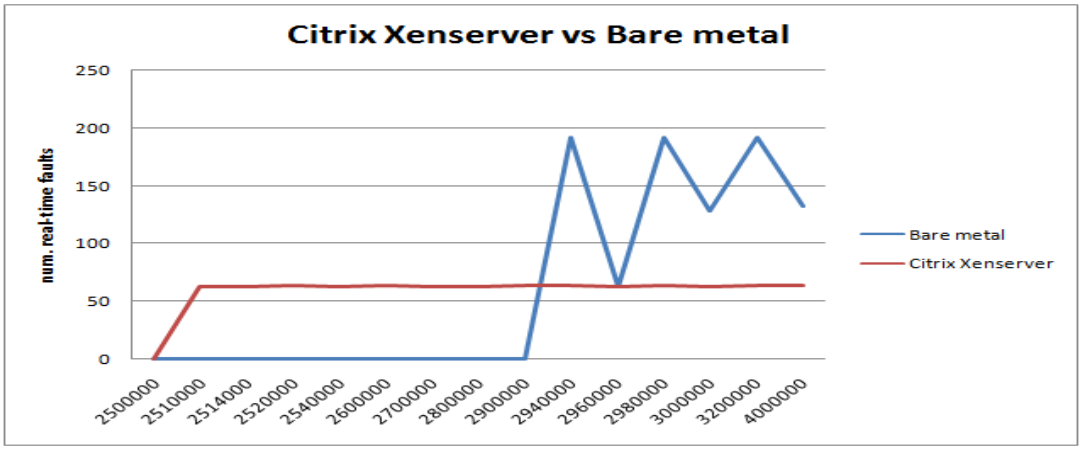

Figura 4.7 Compatibilitat Citrix Xenserver contra Bare metal

Una de les possibilitats per les quals Citrix Xenserver té una compatibilitat tant dolenta, és que no ha executat ALOE amb les opcions de optimització corresponents. Amb la qual cosa, ens fa pensa que no executa les instàncies sobre la CPU nativa sinó que crea vCPU amb una arquitectura diferent. La qual cosa provoca que el compilador no pugui saber l'arquitectura de la vCPU.

Per altre banda, Citrix Xenserver té les seves pròpies opcions de configuració. Tot i ser molt abundants i venir en la instal·lació per defecte, no té opcions per administrar els recursos de la CPU, amb la qual cosa, una vCPU sempre equivaldrà a un thread d'un core de la CPU.

4.5.3. Selecció final de l'hipervisor

Contrari a les nostres expectatives Citrix Xenserver té un pitjor rendiment, tot i ser un hipervisor de nivell 1 i utilitzat PV com a tècnica de virtualització.

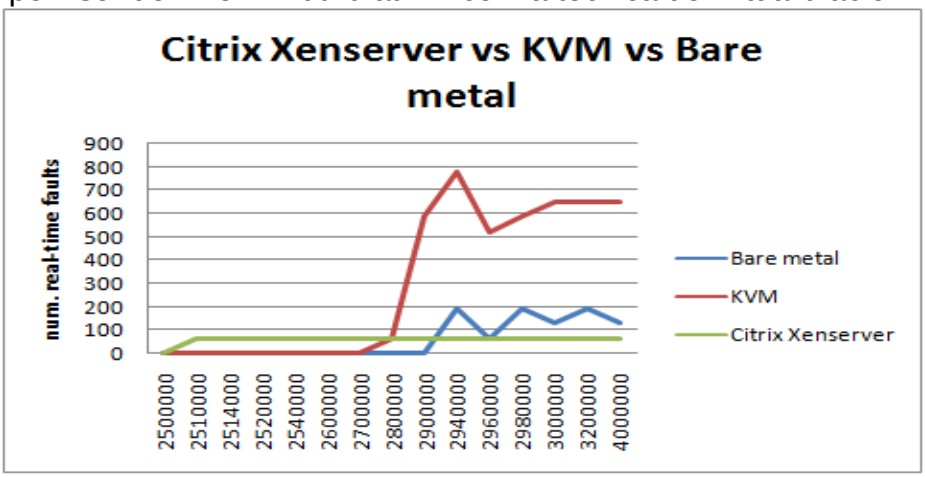

Figura 4.8 Comparativa KVM vs Citrix Xenserver vs Bare metal

El KVM té una integració més fàcil dins de OpenStack, així com una documentació més acurada i actualitzada. A més, té un millor rendiment tot i no ser un hipervisor de tipus 1. L'instal·lació KVM és més fàcil i hi ha diverses eines per personalitzar les instàncies. Per tant, vam triar KVM per treballar.

Citrix Xenserver és un sistema molt rígid amb poques opcions per administrar els recursos de computació. A més moltes vegades ens trobem documentació antiquada per instal·lar nou programari o crear noves imatges.

Una altra solució interessant és Ironic, presentat per Docomo fa un parell d'anys. Aquest hipervisor no funciona amb màquines virtuals pel que no és exactament un controlador de la màquina virtual. Gràcies a aquest fet, no desaprofiten recursos addicionals durant l'execució. Desgraciadament, encara es troba en fase beta i fins la següent versió d'OpenStack no podrem provar aquest hipervisor.

4.5.4. Load balancer

D'igual forma, estem interessats en gestionar les càrregues de les instàncies, enviant les peticions a la instància menys carregada. El nostre objectiu era trencar la cadena de processat en mòduls i col·locar cada mòdul dins d'una xarxa juntament amb un equilibrador de càrrega. L'equilibrador de càrrega monitoritzarà les instàncies i enviarà el tràfic a una instància en concret, segons l'algoritme de balancejat escollit. Partint d'aquest esquema, seria molt fàcil escalar la cadena segons la càrrega del sistema en cada moment, creant més mòduls d'un tipus o traient-ne.

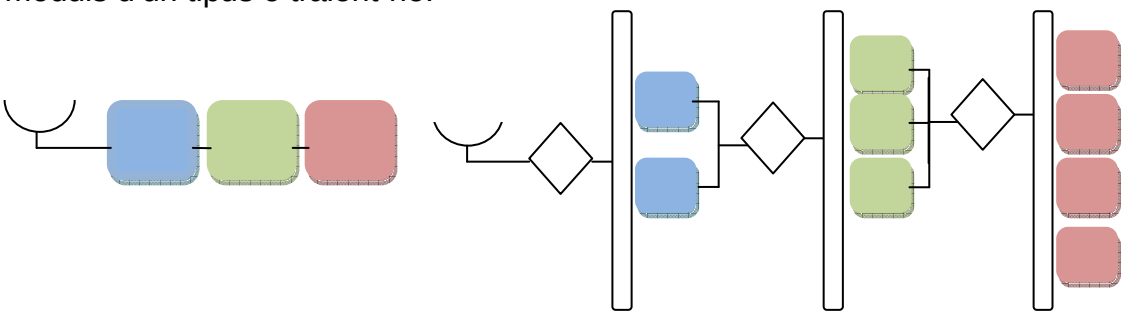

Figura 4.9 Proposta de load balancer

Però la metodologia de treball d'ALOE és incompatible amb l'equilibrador de càrrega. Ja que necessita saber totes les vCPUs disponibles abans d'executar codi. Així que no podem afegir mòduls addicionals quan ALOE ha començat l'execució. Trobareu informació sobre la creació d'equilibradors de càrrega al annex 4.5.

4.6. Configuració d'un cloud públic

El nostre objectiu és definir el grup de recursos disponibles per a cada usuari.
Els usuaris podran gestionar aquest recursos executant les seves pròpies cadenes de processat, creant instàncies, xarxes i routers. El client final que fa ús del recursos no és el propi administrador, amb la qual cosa estem davant d'un cloud públic.

Els recursos es defineixen a través d'un número d'instruccions de la CPU i un percentatge de memòria RAM. Els usuaris administraran aquest recursos mitjançant els flavors, per tant podran crear instàncies, xarxes i routers fins que els recursos assignats s'esgotin.

Però l'OpenStack treballa amb vCPUs, no amb recursos. Per tant l'Administrador pot delimitar el nombre vCPU assignat a cada client, però no el nombre de recursos.
Si ens posem en la posició de l'usuari, la creació d'una instància amb un flavor petit (flavor amb pocs recursos), provoca que estem malgasta'n una vCPU i per tant estem malgasta'n recursos.

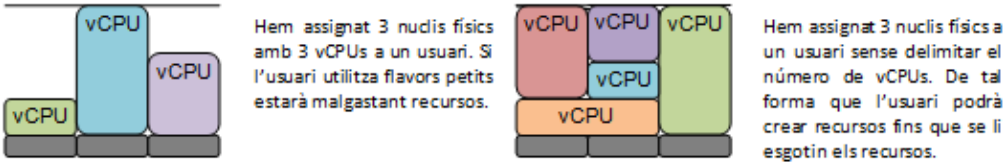

Figura 4.10 Administració de recursos

D'altra banda, els usuaris no poden crear flavors.

Per tant, per definir un paquet de recursos personalitzat per cada usuari utilitzarem flavors i projectes.

- En el apartat de **projectes**, delimitem el número de vCPUs que pot utilitzar el usuari. Posteriorment, és necessari assignar aquest projecte a un usuari. Trobareu més informació sobre la creació de projectes a l'annex 4.6.
- Dins dels **flavors**, hem creat vCPUs amb percentatges de la CPU d'acord amb els desitjos de cada usuari.

Desafortunadament, els usuaris depenen molt de l'administrador si volen treballar amb percentatges de la CPU.

4.7. Configuració d'un cloud privat

En aquest segon escenari, volem demostrar que és possible executar una cadena de processat LTE en temps real dins d'un entorn virtualizat (OpenStack). El client final serem nosaltres mateixos, el propi administrador amb la qual cosa estem treballant en un cloud privat.

Per realitzar aquesta prova hem creat dues configuracions:

- **Processat dins d'un thread**: tota la cadena de processat del transmissor s'executarà íntegrament dins d'un sol thread d'un nucli.
- **Multi processat**: executarem la cadena de processat dins de vàries instàncies.

Per a la realització del **processat dins d'un thread**, hem creat dues instàncies amb el mateix flavor, dues xarxes i un router. En la primera instància, carregarem Ubuntu i en la segona instància farem servir Lubuntu. El router interconnectarà les xarxes virtual amb la xarxa externa per poder accedir a les USRP. Finalment, es connectarà cada una de les instàncies a una xarxa virtual. Podreu troba més informació sobre la creació d'instàncies a l'annex 4.7 i la creació de xarxes l'annex 4.8.

L'objectiu és transmetre des de la instància Ubuntu fins la instància Lubuntu a través de les USRP, i poder veure el espai radioelèctric capturat.

L'ALOE per defecte, busca les USRP disponibles dins de la xarxa mitjançant un missatge de broadcast. Però per seguretat, OpenStack té bloquejat tot el tràfic broadcast, ja que fàcilment es podria congestionar la xarxa. Per corregir aquest problema hem hagut de configurar ALOE perquè dirigeix-hi els paquets cap a una IP fixa. D'aquesta manera, apart de solucionar el problema del broadcast aconseguim dirigir el tràfic cap a una USRP en concret.

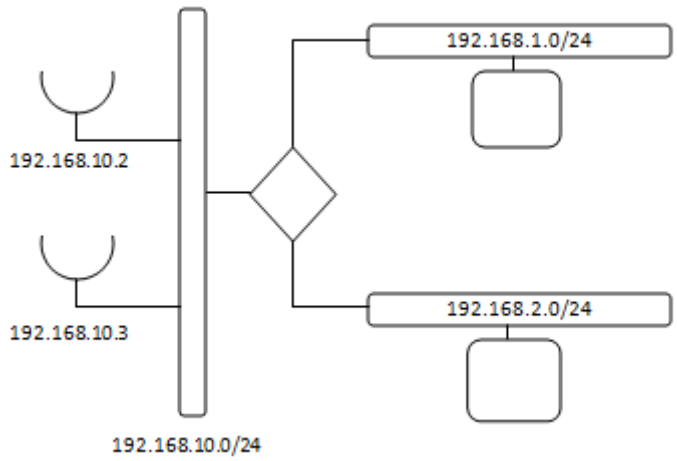

Figura 4.11 Configuració del processat dins d'un thread

Les primeres proves les hem realitzat mitjançant bare metal per tal de poder fer una comparativa amb els resultats obtinguts fent el processament dins del cloud.

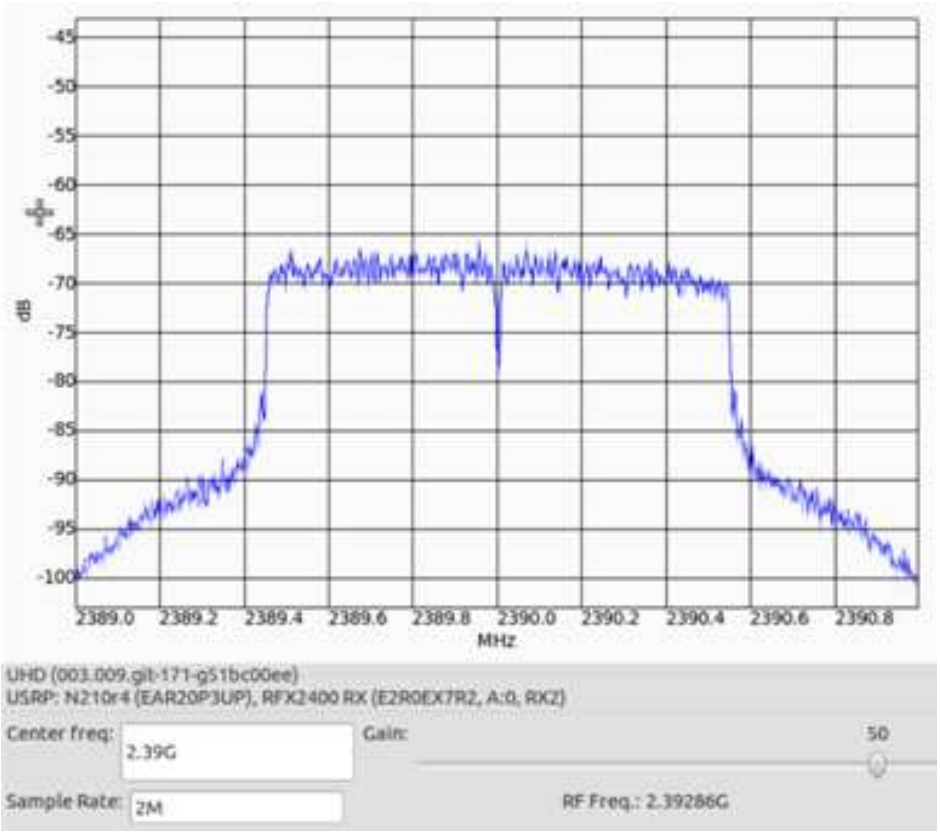

Figura 4.12 Senyal transmès a través de Bare metal

A la figura es mostra l'espectre d'una senyal OFDM real on el transmissor era un computador físic. Vam escollir la freqüència central 2.39 GHz ja que presentava menys soroll dins la banda de wifi. Presenta un ACLR (Adjacent Channel Leakage Ratio) de 20 dB i un espectre ben conformat. Si la forma del espectre és semblant al teòric podem afirmar que el codi està ben escrit i que treballa amb temps real ja que en cas contrari apareixeria una degradació clara d'aquest sobretot per l'aparició de comportaments no lineals.

La nostre infrastructura es pot veure a la figura 4.13. on en el prestatge més elevat trobem las dues USRPs, en el següent pis es troben els switchos tant els gigabit ethernet com el switch InfiniBand (la capsa metàl·lica situada al mig) i en el tercer prestatge trobem les computadores. De dreta a esquerra trobem el Compute (Optheron), Controller (Nephalem), Devstack (Nephalem), Network (Nephalem).

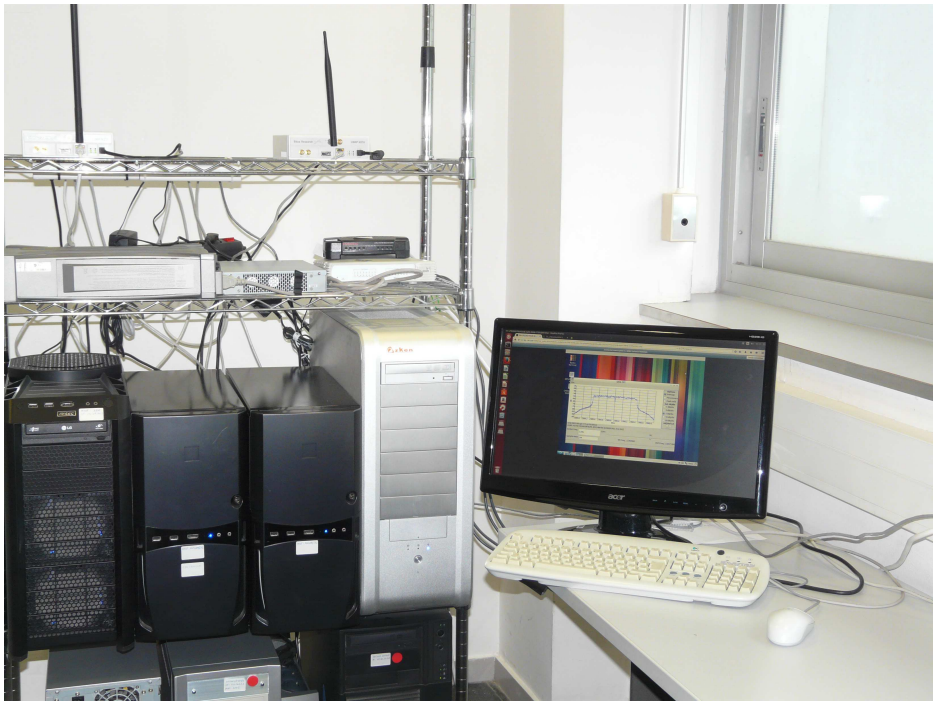

Figura 4.13 Estructura OpenStack

A la pantalla es pot apreciar l'espectre rebut, però anem a fer un zoom per poder comentar les característiques de l'espectre.

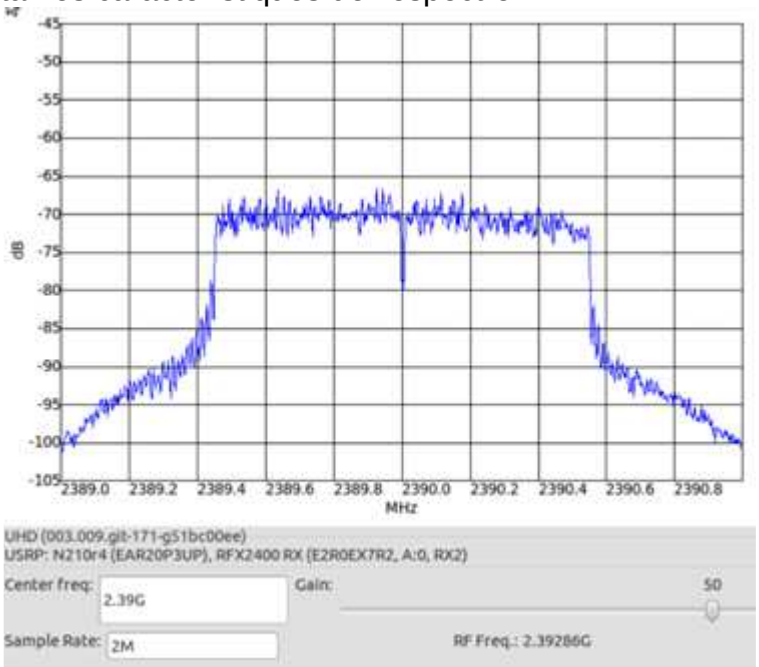

Figura 4.14 Senyal rebut a través del Cloud

La senyal obtinguda de l'execució en el cloud presenta una certa degradació (ACLR de 15 dB). Tot i així, la qualitat sembla prou bona i per tant la cadena de processat segueix treballant em temps real. En aquest sentit, i a falta de

comprovar la recepció d'aquesta senyal, podem assumir que hem complert el nostre objectiu.
En configuració **multi processat**, creem dues instàncies dins d'una sola xarxa amb flavors que tenen pocs recursos assignats. Així forcem que ALOE hagi de trencar la cadena de processament en mòduls i executi cada mòduls dins d'una instància.

Els mòduls s'han d'enviar informació els uns als altres. Sorprenentment, el retard que s'observa entre dues instàncies dins de la mateixa computadora és alt, causant múltiples fallades en temps real d'acord amb el nombre de connexions. Per resoldre aquest problema, vam optar per utilitzar tecnologia SRIOV que incorporant les places InfiniBand. Aquesta tecnologia treballa de forma similar al VT-x o el AMD-v però en aquest cas virtualitzem la placa de xarxa. Malauradament, és necessari que la placa base del ordinador permeti fer aquest tipus de virtualitzacions, la placa base dels nostres ordinadors tot i ser relativament noves no incorporen aquest tecnologia. Al ser una tecnologia relativament nova i que pocs usuaris necessiten, hi ha pocs models disponibles.

4.7.1. Orchestration

Una de les opcions que ens proporciona l'OpenStack és la facilitat de poder automatitzar tasques.
Entre altres opcions Heat ens permet crear i llençar escenaris. Hem creat dos escenaris durant el transcurs del treball:

- **Creació d'un projecte**: aquesta plantilla crea tots els objectes descrits en la secció 4.8 per poder realitzar proves.
- **Autoscaling**: aquest template està més orientat al món dels servidors, on desplegarem un escenari i monitoritzarem les instàncies per poder actuar davant d'una sobrecàrrega.

La creació d'un projecte, és una plantilla que crea dues instàncies, un router i dues xarxes virtuals. El router interconnecta la xarxa virtual on estaran connectades les instàncies, amb la xarxa exterior. Un cop tota la infraestructura estigui creada, a través de CloudInit arrencarà ALOE i GNU radio per poder realitzar una transmissió. Com a inputs hem definit: les imatge i els flavor.

La plantilla que hem creat es pot trobar a l'annex 4.9.

L'Autoscaling crea una instància on la seva imatge serà un servidor, un router, una xarxa i un balancejador de càrrega.
El balancejador monitoritzarà la instància, quan la càrrega de la vCPU superi el 70% durant 2 minuts, l'OpenStack crearà una nova instància i la connectarà amb el balancejador. Tanmateix si la càrrega d'una vCPU és del 10% durant 2 minuts borrera la instància.

La plantilla es pot trobar a l'annex 4.10.

CAPÍTOL 5. CONCLUSIONS

Aquest projecte ha suposat el primer pas per a l'execució de SDR dins d'un entorn virtualitzat, executant una cadena LTE íntegrament dins d'un cloud i avaluant les eines de gestió de les que disposa OpenStack per la gestió de la infraestructura.

El món del cloud computing encara està molt enfocat a la gestió de servidors, tot i que cada vegada més s'utilitzen per propòsits ben diferents. Nosaltres hem introduït el processament del senyal dins del cloud. Però no hem fusionat les dues tecnologies SDR i el cloud computing sense les quals moltes aplicacions com el CoMP o el ICIC no es podrien implementar. Tot i això hem demostrat que el cloud-RAN és una realitat que vindrà en els pròxims anys.

L'OpenStack pretén ser la plataforma predilecta per a l'execució de SDR. Des del nostre punt de vista, té molts punts a favor. Deixant de banda que és una plataforma totalment open source i per tant és altament personalitzable. OpenStack té molt camí ja fet, un alt nivell d'administració de recursos dins de la computadora i dins de l'instància. Així com moltes eines d'administració de xarxes que podrien ser molt útils, com per exemple el balancejador de càrrega. Per tant, no és estrany que moltes empreses especialitzades en el sector apostin per aquesta plataforma per la realització d'un cloud-RAN.

La compatibilitat dels hypervisors amb bare metal és molt alta, permetent executar aplicacions amb uns requisits molt precisos (ALOE) sense malgastar recursos en la virtualització alhora que no perdem el temps real dins de la instància. D'altra banda les nombroses eines existents dins del món de la virtualització, permetent fer una caracterització molt concreta de les característiques de les instàncies.

ALOE ha sigut una eina que ens ha ajudat molt en el desenvolupament del projecte. Ja que ens ha permès fer les proves de benchmark així com executar el SDR. Tot i així, seria necessari replantejar-se utilitzar altres frameworks, ja que la filosofia de treball d'ALOE el dia d'avui ens tanca moltes portes.

El cloud computing ja és una tecnologia suficientment madura com perquè es comenci a plantejar un canvi en el model de desplegament d'infraestructures wireless, passant d'un model distribuït a un model centralitzat, on es podran aplicar nous algoritmes per tal de reduir les interferències, augmentant l'experiència de l'usuari final.

CAPÍTOL 6. TREBALL DE FUTUR

A partir dels resultats obtinguts en el treball s'obren molts camins d'investigació per aquest tipus de plataformes.

El principal problema que s'hauria de resoldre és reduir el retard per transmissions instància a instància i transmissions instància a USRP.
Tot i que el processament de la senyal treballa en temps real el retard que introdueixen les transmissions fa que no complim amb els estàndards proposats per LTE. (secció 1.1)
Una de les tecnologies que pot ajudar a resoldre aquests problemes és el SRIOV que malauradament no hem pogut implementar. L'implantació d'aquesta tecnologia hauria de reduir el retard entre instàncies així com reduir el retard de les transmissions entre VM i USRP, ja que treballa amb fibra òptica.

Un altre aspecte que s'hauria d'estudiar és l'adaptació del SDR dins del món del cloud computing. Ja que amb l'estructura que hem realitzat trèiem poc profit de les possibilitats que ofereix el cloud, seria interessant fer una planificació de com fer el processament del senyal: processant la senyal de cada antena per separat o ajuntar tot el tràfic de totes les antenes i després processar-lo. Així doncs apareixen molts temes d'estudi com la realització d'algoritmes de ICIC o CoMP dins d'un cloud-RAN. Aquestes són algunes de les idees que serien interessants d'avaluar.

Unes millores dins d'ALOE ens permetrien fer més proves per adaptar el SRD dins del cloud. La seva filosofia de treballar amb mòduls ens segueix sent vàlida però arribats a aquest punt seria interessant poder fer proves d'escalat de recursos. ALOE és fàcilment escalable dins d'un entorn que no canvia en el temps, un centre de processat tindrà més càrrega o menys segons l'hora del dia per tant hauria de permetre adherir o eliminar nous moduls un cop aquesta ha entrat en funcionament. Per tant, un framework que fos escalable en el temps seria un bon candidat.

Per altra banda, l'estudi dels hipervisors segueix obert, ja que com hem comentat en la secció 2.6 l'hipervisor Ironic podria oferir un millor rendiment que KVM o Xen.

BIBLIOGRAFIA

[1] Resource management for software-defined radio clouds – Ismael Gómez Miguelez, Vuk Marojevic, Antoni Gelonch Bosh

[2] Wikipedia – Radio definida por software
http://es.wikipedia.org/wiki/Radio_definida_por_software

[3] 3GPP Long Term Evolution (LTE) – ICIC and eICIC http://4g-lte-world.blogspot.com.es/2012/06/icic-and-eicic.html

[4] Wikipedia – Long Term Evolution (LTE)
http://es.wikipedia.org/wiki/Long_Term_Evolution

[5] Wikipedia – Real-time computing
http://en.wikipedia.org/wiki/Real-time_computing

[6] Overview of the 3GPP Long Term Evolution Physical Layer - 2.5 SC-FDMA
https://www.freescale.com/files/wireless_comm/doc/white_paper/3GPPEVOLUTIONWP.pdf

[7] Wikipedia – Virtual machine
http://en.wikipedia.org/wiki/Virtual_machine

[8] Integració de KVM, Libvirt i Monitoring de baix nivell - Marc Gonzalez Mateo
http://upcommons.upc.edu/pfc/bitstream/2099.1/10446/1/pfc%20marc%20gonzalez%20mateo.pdf

[9] Xen project software Overview – guest types
http://wiki.xen.org/wiki/Xen_Project_Software_Overview#HVM

[10] Wikipedia – hypervisor
http://en.wikipedia.org/wiki/Hypervisor

[11] Wikipedia – Cloud Computing
http://es.wikipedia.org/wiki/Computaci%C3%B3n_en_la_nube

[12] Interoute – What is Paas? / What is *?
http://www.interoute.com/what-paas

[13] CBC Radio-Canada – Services Paradim
http://www.cbc.radio-canada.ca/en/reporting-to-canadians/sync/sync-issue-1-2012/cloud-services/

[14] Wikipedia – C-RAN
http://en.wikipedia.org/wiki/C-RAN

[15] ALOE project
https://github.com/flexnets/aloe/wiki/ALOE-Project

[16] Kernel Based Virtual Machine
http://www.linux-kvm.org/page/Main_Page

[17] Wikipedia - QEMU
http://en.wikipedia.org/wiki/QEMU

[18] ochobitshacenunbyte – LVM para torpes (I)
http://www.ochobitshacenunbyte.com/2015/02/27/lvm-para-torpes/
[19] Wikipedia – Scheduling
http://en.wikipedia.org/wiki/Scheduling_%28computing%29

[20] Linux jounal – Completely fair scheduler
http://www.linuxjournal.com/magazine/completely-fair-scheduler

[21] pk.org – Process scheduling https://www.cs.rutgers.edu/~pxk/416/notes/07-scheduling.html

[22] IBM – pinning
https://www01.ibm.com/support/knowledgecenter/linuxonibm/liaat/liaattunpingoodbad.htm

[23] Wikipedia – hilo de ejecución
http://es.wikipedia.org/wiki/Hilo_de_ejecuci%C3%B3n

[24] Rastersoft – concepto de threat
http://www.rastersoft.com/OS2/CURSO/THREAD.HTM

[25] Ramdom samples – Idle scheduler
http://www.randomsample.de/cedetdocs/semantic-user/Idle-Scheduler.html

[26] Libvirt – CPU tunning
https://libvirt.org/formatdomain.html#elementsCPUTuning

[27] RDO – Configuring and deploying LBAAS in OpenStack Havana
https://www.rdoproject.org/LBaaS

[28] Estudi i implantació d'un servei d'infrstructura federat (IaaS) – Oriol Martí Bonvehí http://upcommons.upc.edu/pfc/bitstream/2099.1/18664/1/84369.pdf

[29] Qyjohn – CY15-Q1 Community Analysis
http://www.qyjohn.net/?p=3801

[30] OpenStack – OpenStack installation guide for Ubuntu 14.04
http://docs.openstack.org/juno/install-guide/install/apt/content/

[31] OpenStack –Template guide
http://docs.openstack.org/developer/heat/template_guide/index.html

[32] Rackspace – Wordpress and LAMP installation via cloud-init
https://developer.rackspace.com/blog/using-cloud-init-with-rackspace-cloud/

[33] Gentoo linux – Optimización de GCC
https://wiki.gentoo.org/wiki/GCC_optimization/es

[34] Cloud-init documentation
https://cloudinit.readthedocs.org/en/latest/

ANNEXOS

Annex 4.1. Template devstack

Al arxiu local.conf que hem utilitzat és el següent:

```
[[local|localrc]]
# Default passwords
ADMIN_PASSWORD=devstack
MYSQL_PASSWORD=devstack
RABBIT_PASSWORD=devstack
SERVICE_PASSWORD=devstack
SERVICE_TOKEN=devstack
RECLONE=yes

SCREEN_LOGDIR=/opt/stack/logs
disable_service n-net
enable_service q-svc
enable_service q-agt
enable_service q-dhcp
enable_service q-l3
enable_service q-meta
enable_service neutron
enable_service q-lbaas
disable_service tempest
enable_service s-proxy s-object s-container s-account
SWIFT_HASH=66a3d6b56c1f479c8b4e70ab5c2000f5
```

Figura A.4.1.1 Devstack template

Hem habilitat tots els serveis, incloent el Neutron amb totes les seves eines.

Annex 4.2. Creació d'un flavor

Dins de les opcions de configuració del menú de l'esquerra com administrador, clicarem a "Identity"->"flavors". Posteriorment, clicarem a 'create new flavor' i apareixerà la següent finestra:

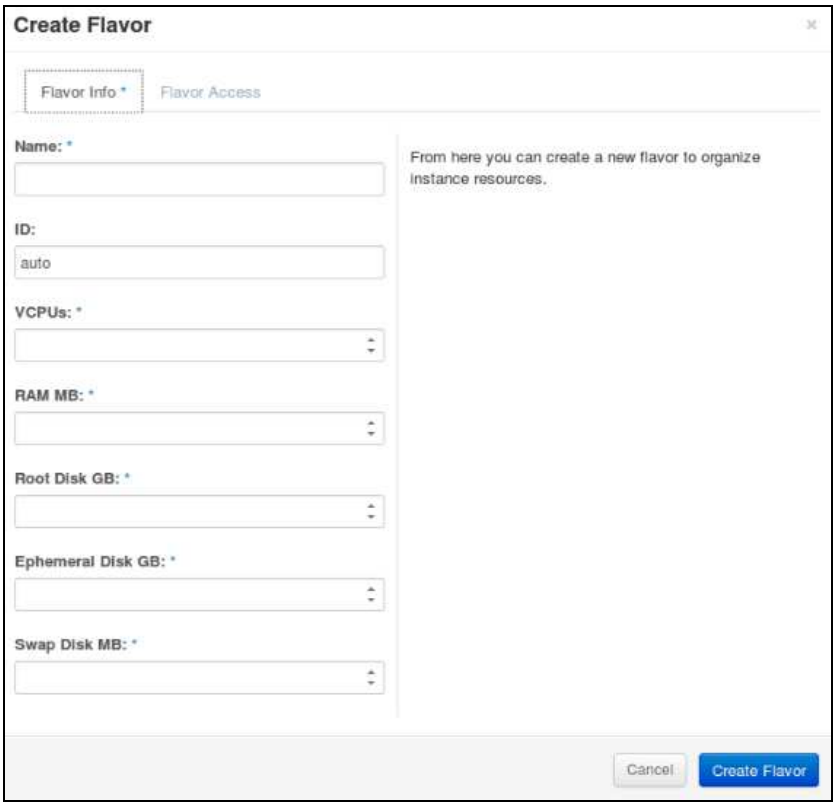

Figura A.4.2.1 Flavor info

A la primera pestanya, escollirem un nom, un número de vCPUs i els megabytes de RAM que volem utilitzar. Les altres opcions són per crear particions dins de la instància: "Root disk GB" són el GB que donem al directori "/" i el 'Ephemeral Disk GB' són els GB que li donem als usuaris, en termes de directoris seria "/home/". Posteriorment, podem delimitar quins usuaris volem que puguin utilitzar aquest flavor a la pestanya "Flavor access".

Annex 4.3. Creació d'una imatge

Virt-manager té la següent aparença, molt semblant a altres hipervisors com virtualbox. Ens permet crear VM de forma molt fàcil a partir d'un front-end molt intuïtiu. A més, és molt fàcil personalitzar la instància.

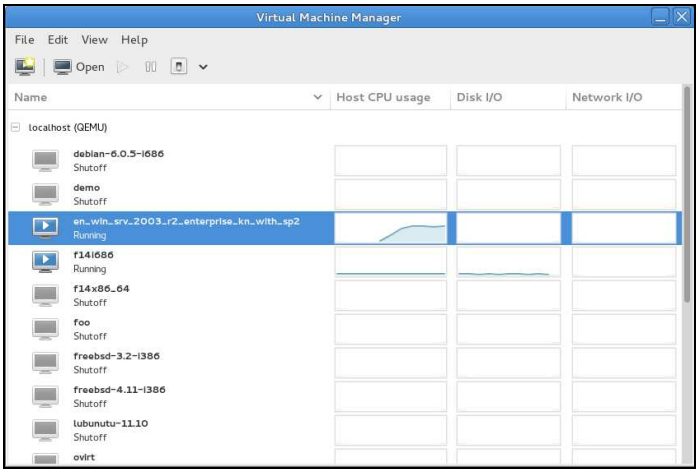

Figura A.4.3.1 Virt-manager

Un cop tenim la VM creada és hora de pujar-la al cloud. Tots els usuaris poden pujar imatges al cloud. Primer de tot, tenim que localitzar la VM, que es troba al directori: "/var/lib/libvirt/images/". Es necessari moure la instància per poder pujar-la a OpenStack, ja que és troba per sota el directori home i per tan no tenim permisos per accedir-hi. Un cop moguda a una carpeta coneguda és necessari canviar-li els permisos amb la comanda: sudo chmod 777 *NOM*.

Un cop realitzats els pasos anterior, ens dirigirem al Horizon dins del panell esquerre, al apartat d'imatges-> create new Image.

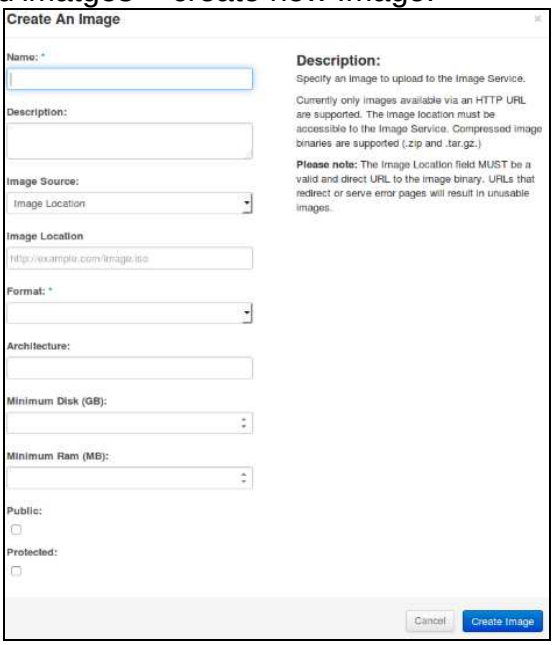

Figura A.4.3.2 Creació d'una imatge

Se'ns obrirà la següent finestra, només tenim que definir el nom, indicar el path on es troba la imatge, escollir el format (nosaltres treballem amb KVM per tant, escollim QCOW2, segons l'hipervisor l'extensió canvia, mira la taula 3.1) i finalment marca la pestanya de "públic" si volem que tothom pugui accedir a l'imatge.

Annex 4.4. Template CloudInit

La plantilla que hem creat és la següent:

```
#cloud-config
runcmd:
  -  cd /home/nodea/DADES/lib_10_2_2015/libLTE/
  -  rm –R build
  - mkdir build
  - cd build
  - cmake ../
  - make
  - sudo make install
  - cd lte/examples
  ./pdsch_enobeb_antiguo –l 0.25 –g 50.0 –f 239000000 –a
addr=192.168.10.3
```

Figura A.4.4.1 CloudInit template

Aquesta plantilla sempre té que començа de la mateixa manera, amb la comanda: '#cloud-config'. Seguidament, s'ha d'indicar la opció predefinida que es vol utilitzar, en el nostre cas 'runcmd' (per executa comandes amb permisos de superusuari) i finalment les comandes a realitzar [34].

Annex 4.5. Creació d'un load balancer

La creació de l'equilibrador de càrrega comença amb la creació del pool, aquí indicarem en quina xarxa volem treballar, el protocol que volem balancejar així com l'algoritme de balanceig. Els algoritmes disponibles són:

- **Round robin**: s'envien les peticions aleatòriament entre les instàncies.
- **Source IP**: les peticions d'una IP en concret s'envien sempre a la mateixa instància.
- **Least connections**: s'envia la petició a la instància amb menys connexions.

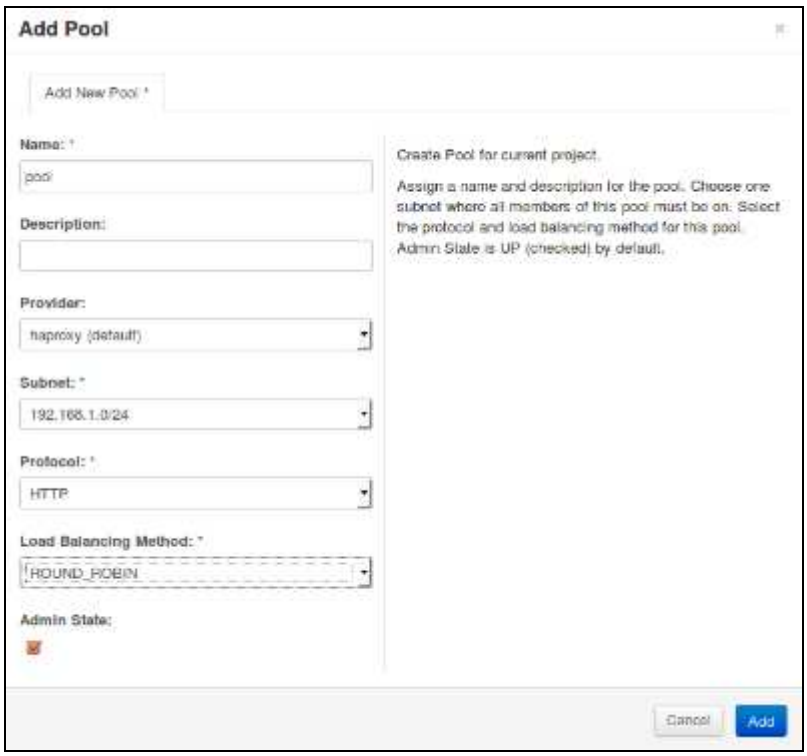

Figura A.4.5.1 Creació d'un pool

Posteriorment indicarem quins members (instàncies) composaran l'equilibrador, el port per el cual balancejaran la càrrega i finalment el 'Pes' (podem posar prioritats a cada instància per tal de que la majoria de peticions vagin a un member en concret).

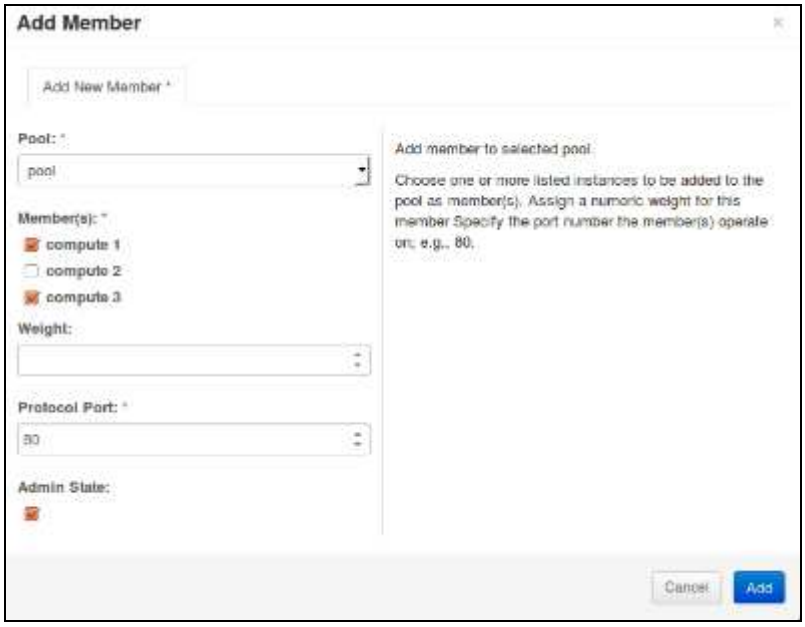

Figura A.4.5.2 Afegir nous members

En el següent pas crearem el monitor, aquí indicarem el tipus de protocol que volem monitoritzar. Els atributs delay, timeout i max retries serveixen per comprovar la càrrega dels members, enviant un determinat tipus de paquets. Si una instància no respon a una petició passarà a estar en estat inactiu i el equilibrador no li enviaria peticions.

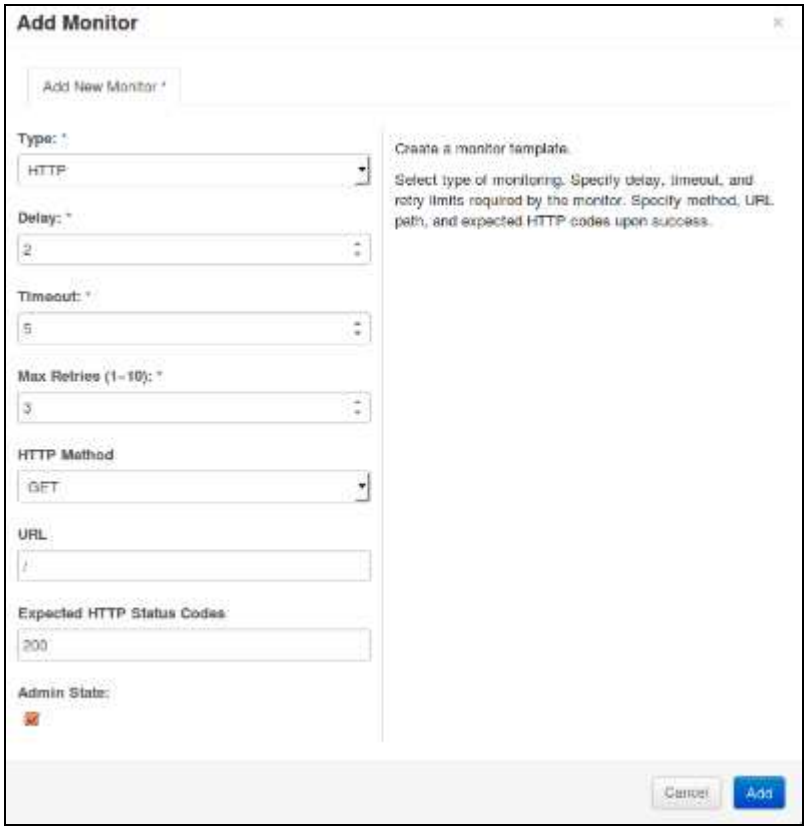

Figura A.4.5.3 Afegir un monitor

Finalment, hem de crear el VIP. Escollint una IP on s'enviaran les peticions per ser posteriorment redirigides a un member en concret.

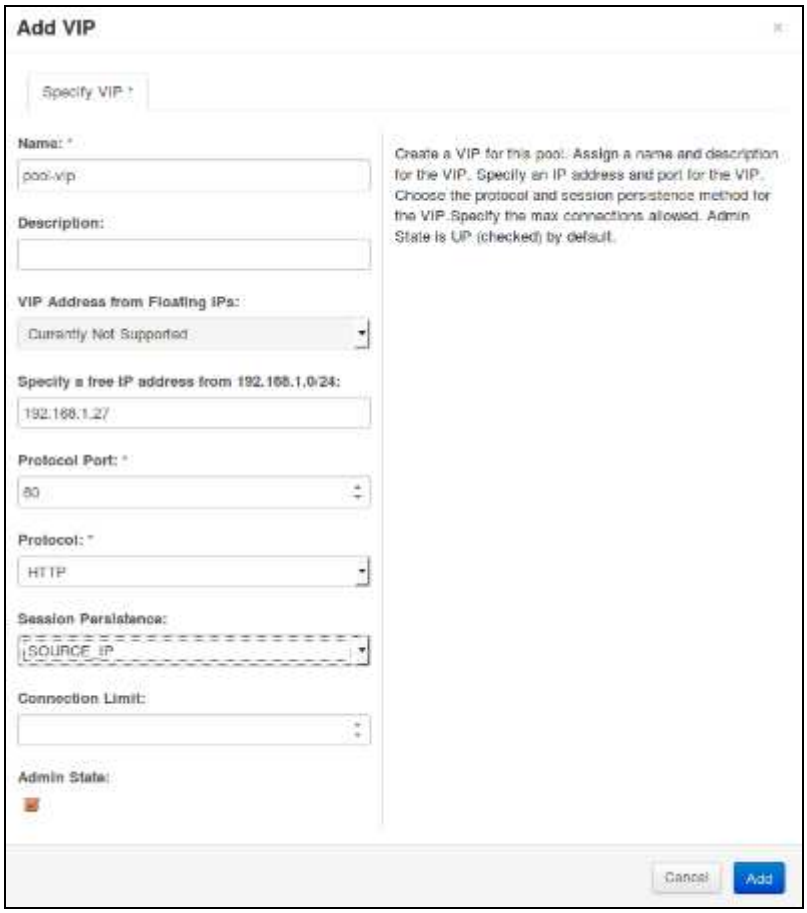

Figura A.4.5.4 Afegir un VIP

Annex 4.6. Creació d'un Usuari i un Projecte

La creació d'usuaris és molt senzilla. Dirigint-nos a la pestanya Identity com a administrador, clicarem a 'User'. Dins d'aquesta finestra clicarem dins de 'create user'. A la següent finestra es poden veure tots els camps que s'han d'específicar. Molts són camps típics, excepte els dos últims:

- **Projecte principal**: els usuaris treballen dins de projectes, que no són res més que els recursos que l'administrador li assigna a l'usuari.
- **Rol**: indica els permisos que té aquest l'usuari, '_member_' és usuari sense permisos.

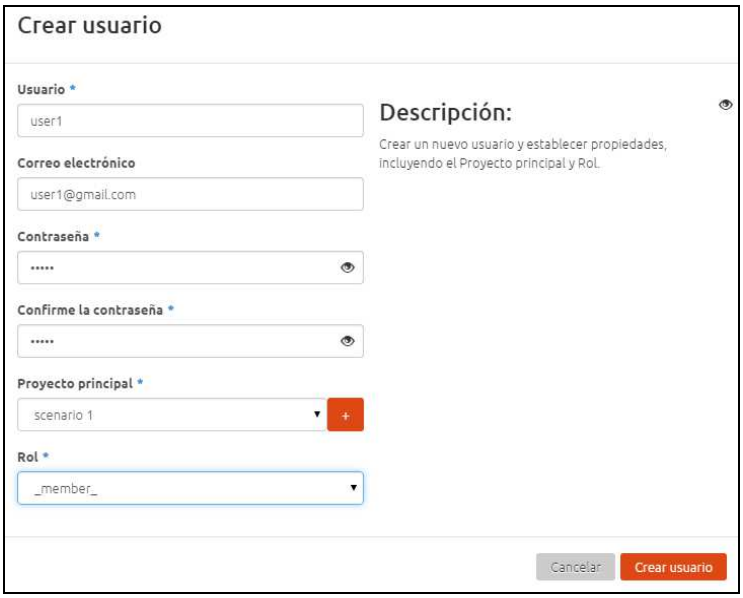

Figura A.4.6.1 Creació d'un usuari

Posteriorment, passarem a crear un projecte, com administradors ens dirigim al menú esquerra a la pestanya projects i clicarem a "create project". Concretament, comentarem la pestanya de 'Quota'.

Hi ha moltes opcions per limitar els recursos als usuaris, a nosaltres només ens interessa els que serveixen per administrar recursos que intervenen en la execució de l'SDR, com són: els el número de vCPUs disponibles, el número d'instàncies que es pot crear dins d'un projecte i la memòria RAM.
"Archivos inyectados" i "contenido del fichero inyectado" fan referència al CloudInit, on podem executar un número de templates amb un màxim de comandes per plantilla.
Les altres opcions són d'administració de xarxes, els noms són bastant intuïtius excepte "grupos de seguridad" que es referix el número de firewalls que el usuari pot crear.

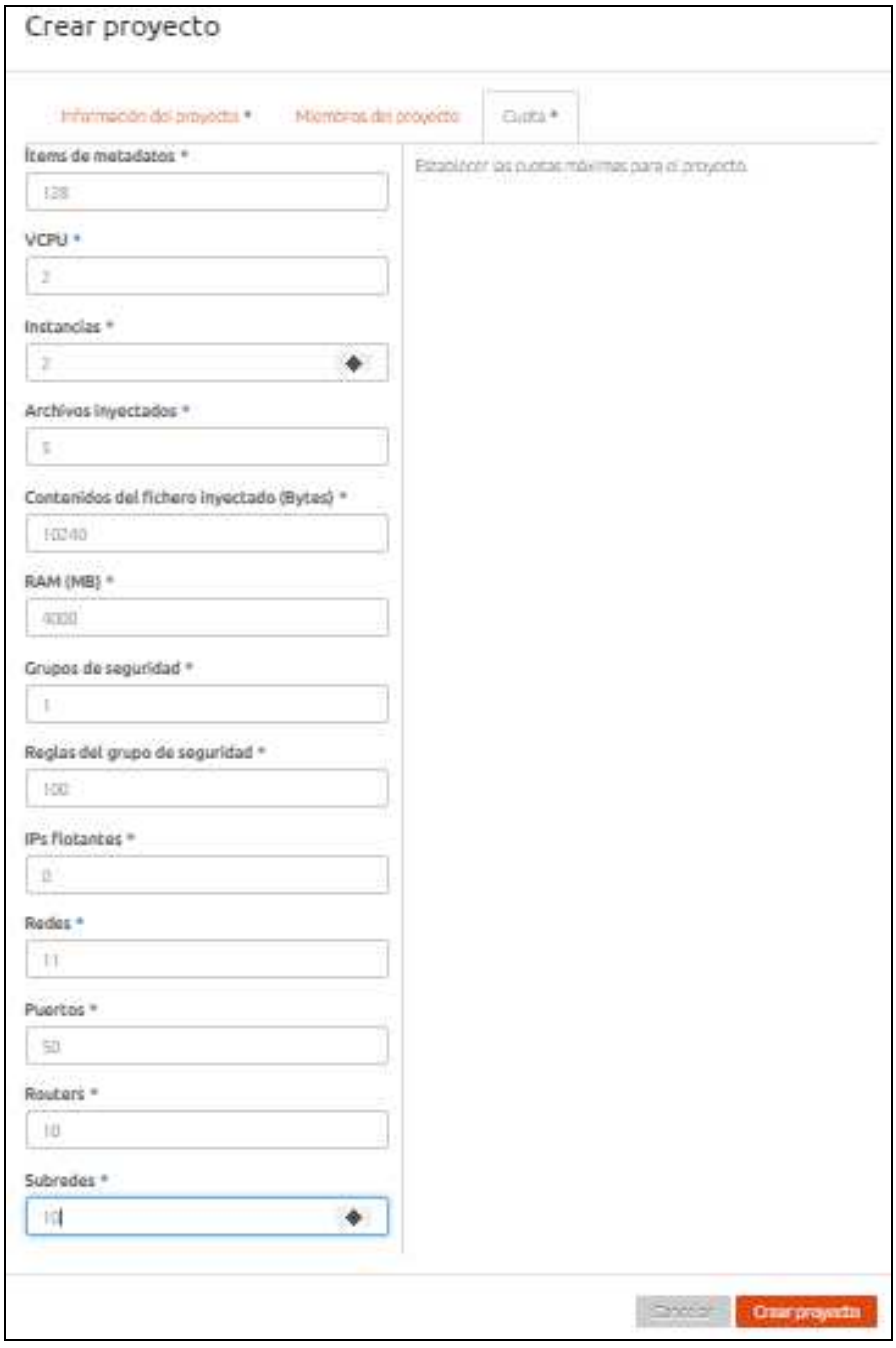

Figura A.4.6.2 Definir noves quotas

Annex 4.7. Llançament d'una instància

La creacio d'instàncies es fa a la pestanya de 'instàncies' dins del panell de l'esquerra.
Un cop cliquem a la "launch instance" se'ns obrirà la següent finestra. Dins de 'Details' hem d'especificar el flavor i la imatge.

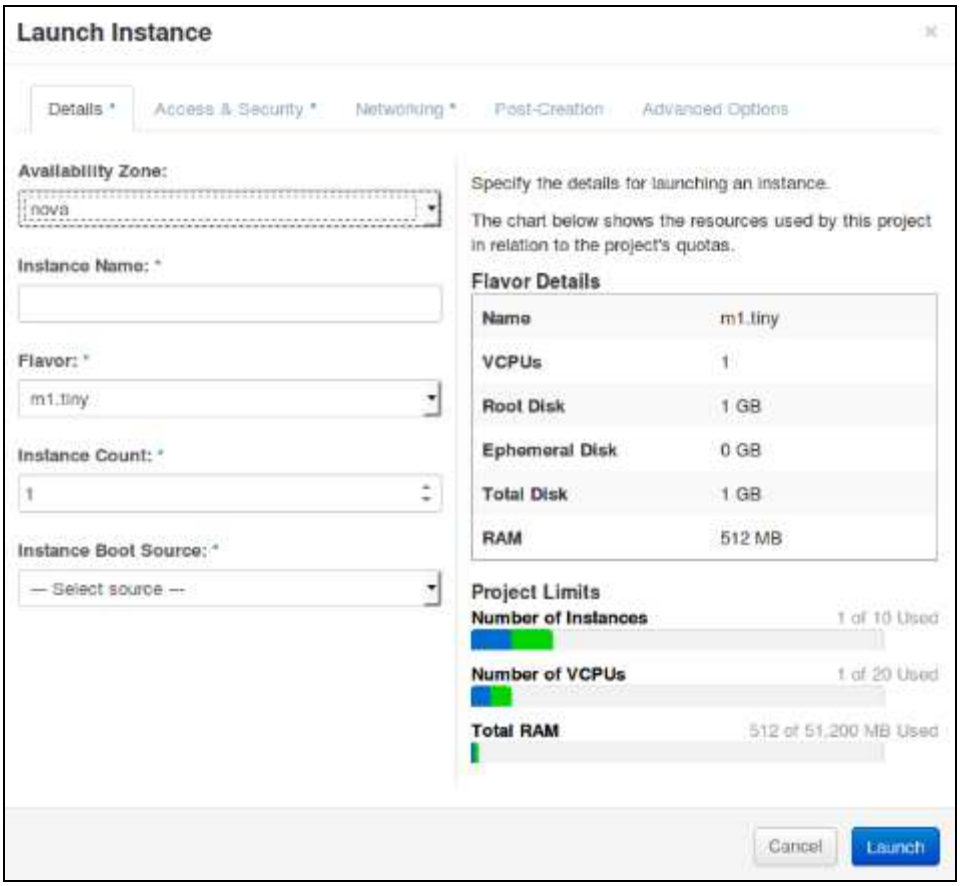

Figura A.4.7.1 Creació d'una instància

Dins de la pestanya Networking, escollirem la xarxa virtual on volem que es connecti la instància.

Figura A.4.7.2 Afegir una xarxa virtual a una instància

Finalment, a la pestanya ‘Post-creation’ escriurem el codi CloudInit que volem executar quan arrenquem la instància. Si deixem aquest camp en blanc, la instància arrencarà sense executar cap comanda un cop s’hagi carregat tot el sistema.

Figura A.4.7.3 Creació d’una instància opcions opcionals

Annex 4.8. Creació d'una xarxa virtual i un router

En el panell de l'esquerra trobem una opció anomenada "red" que ens permet administrar les opcions de xarxa.

Per crear una xarxa ens dirigim a 'redes' i seguidament cliquem a "crear red"

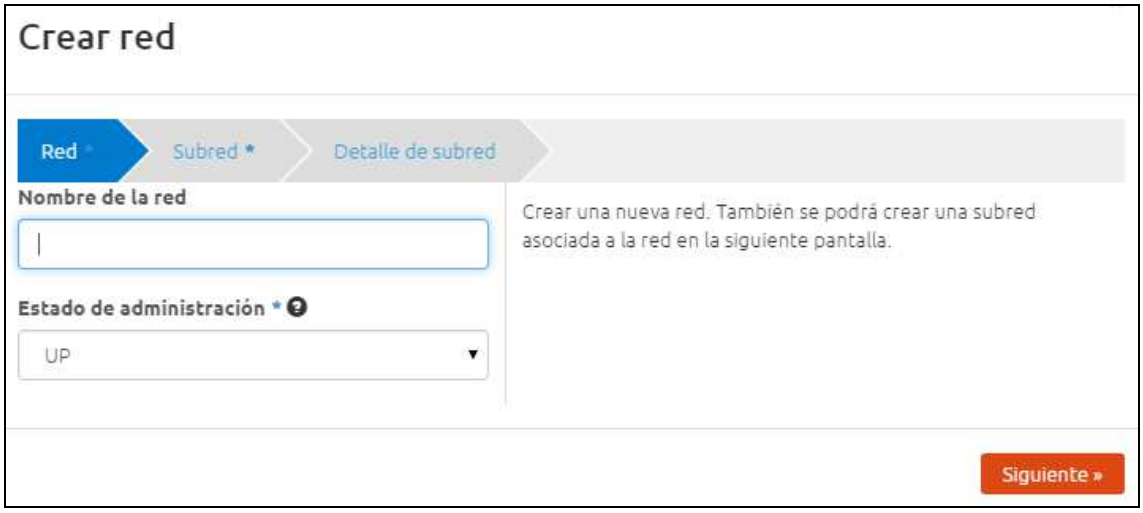

Figura A.4.8.1 Creació d'una xarxa virtual

A la primera finestra només tenim que donar un nom a la xarxa. La següent pestanya tenim que definir la subxarxa:

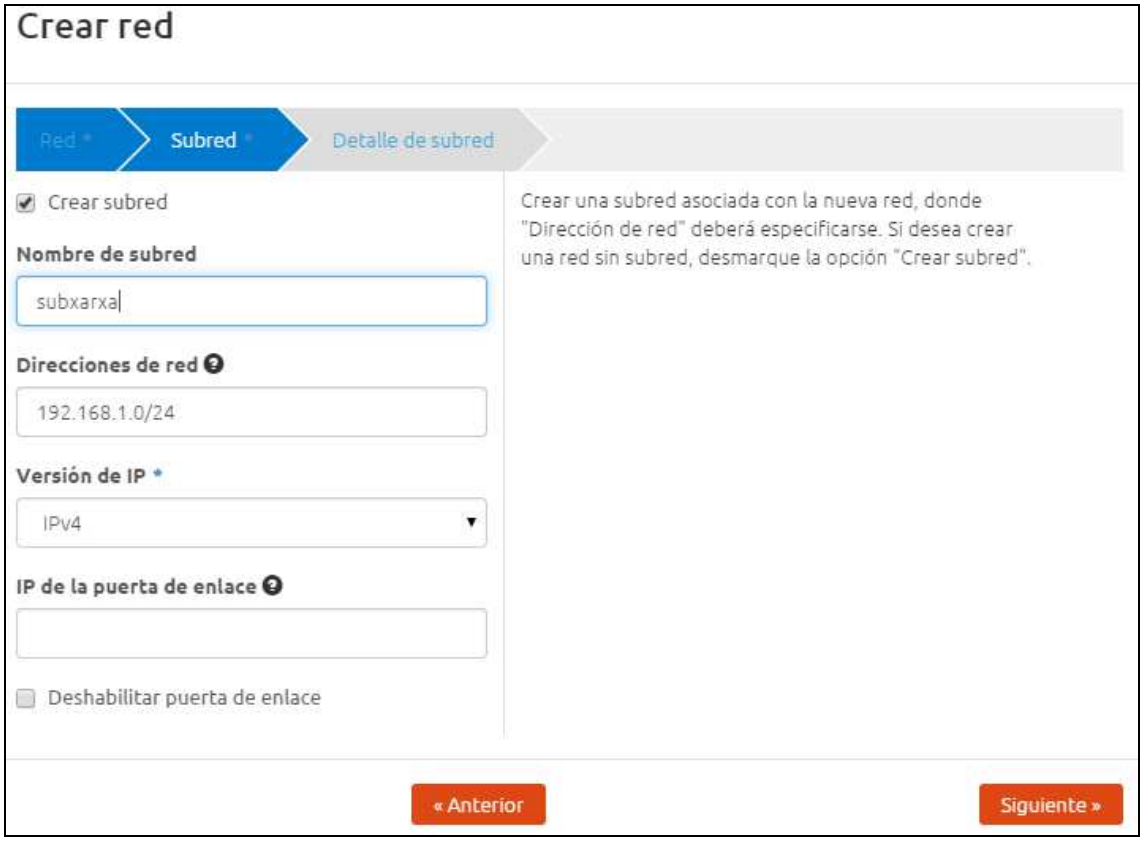

Figura A.4.8.2 Creació d'una subxarxa

Es necessari escollir un nom i el rang de IPs que volem que tingui la subxarxa.

En l'última pestanya habilitarem el DCHP per tal de que les instàncies obtinguin una IP i puguin sortir a la xarxa exterior. Hi han 3 camps opcional:

- **Pools de asiganción**: si volem que tots les instàncies siguin members d'un balancejador de càrrega podem fer-ho especificant el nom del pool.
- **Servidor DNS**: com el seu nom indica podem definir el servidor DNS de les instàncies.
- **Rutas de host**: a vegades és necessari enrutar el tràfic de les instàncies, amb aquesta opció podem enrutar totes de les instàncies de la subxarxa.

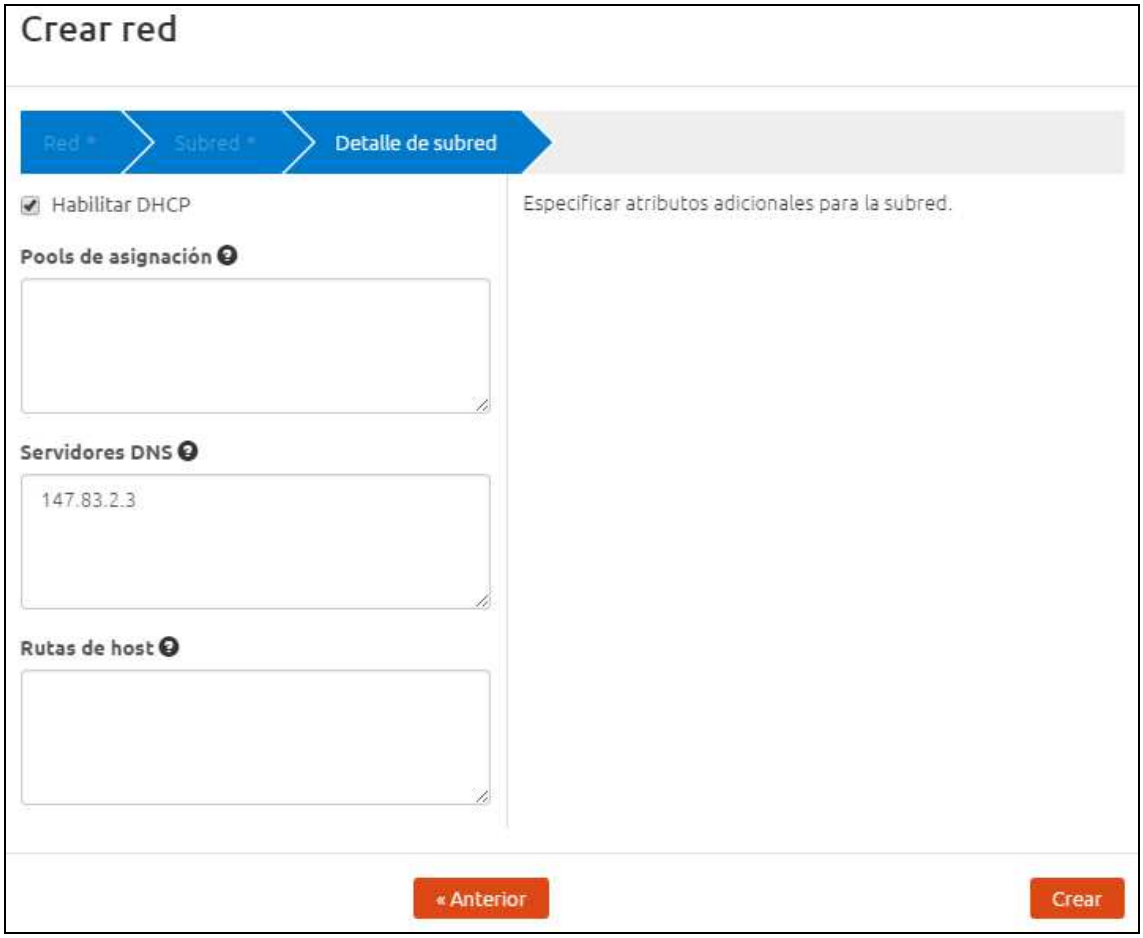

Figura A.4.8.3 Creació d'una xarxa atributs opcional

Un cop hem creat una xarxa virtual necessitem connectar-la amb la xarxa exterior. La xarxa exterior és una xarxa que hem definit nosaltres i que només l'administrador pot gestionar. Per tant, necessitem connectar la xarxa que acabem de crear amb la xarxa externa a través d'un router.

El router el podem crear en el mateix panel dins de router: red->router->crear router. Només necessitarem triar un nom per el router.

Per mitjar de les interfícies podem unir dues xarxes tal com mostra la figura següent:

Figura A.4.8.4 Introducció de noves interfaces al router

Tenim que realitzar aquesta acció per les dues xarxes. On indicarem en cada cas la xarxa que volem que connecti el router i una IP de la xarxa de la mateixa.

Annex 4.9. Template ‘Creaciò d’un projecte’

```
heat_template_version: 2013-05-23

description: >
 HOT template to create a new neutron network to run master and slave ALOE applications.

parameters:
 image:
   type: string
   description: Name of image to use for servers
 flavor:
   type: string
   description: Flavor to use for servers
 públic_net:
   type: string
   description: >
     ID or name of públic network for which floating IP addresses will be allocated
 private_net_name:
   type: string
   description: Name of private network to be created
 private_net_cidr:
   type: string
   description: Private network address (CIDR notation)
 private_net_gateway:
   type: string
   description: Private network gateway address
 private_net_pool_start:
   type: string
   description: Start of private network IP address allocation pool
 private_net_pool_end:
   type: string
   description: End of private network IP address allocation pool

resources:
 private_net:
   type: OS::Neutron::Net
   properties:
     name: { get_param: private_net_name }

 private_subnet:
   type: OS::Neutron::Subnet
   properties:
     network_id: { get_resource: private_net }
     cidr: { get_param: private_net_cidr }
     gateway_ip: { get_param: private_net_gateway }
     allocation_pools:
       - start: { get_param: private_net_pool_start }
         end: { get_param: private_net_pool_end }

 router:
   type: OS::Neutron::Router
   properties:
     external_gateway_info:
       network: { get_param: públic_net }

 router_interface:
   type: OS::Neutron::RouterInterface
   properties:
```

```
    router_id: { get_resource: router }
    subnet_id: { get_resource: private_subnet }

 master:
  type: OS::Nova::Server
  properties:
    name: Master
    image: { get_param: image }
    flavor: { get_param: flavor }
    networks:
      - port: { get_resource: master_port }
    user_data:
       str_replace:
          template: |
             #! /usr/bin/env bash
             cd /home/node/Aloe1.6
             runcf

 master_port:
  type: OS::Neutron::Port
  properties:
    network_id: { get_resource: private_net }
    fixed_ips:
      - subnet_id: { get_resource: private_subnet }

 master_floating_ip:
  type: OS::Neutron::FloatingIP
  properties:
    floating_network: { get_param: públic_net }
    port_id: { get_resource: server1_port }

 slave:
  type: OS::Nova::Server
  properties:
    name: Slave
    image: { get_param: image }
    flavor: { get_param: flavor }
    networks:
      - port: { get_resource: slave_port }
    user_data:
       str_replace:
          template: |
             #! /usr/bin/env bash
             cd /home/node/Aloe1.6
             runcf

 server2_port:
  type: OS::Neutron::Port
  properties:
    network_id: { get_resource: private_net }
    fixed_ips:
      - subnet_id: { get_resource: private_subnet }

 slave_floating_ip:
  type: OS::Neutron::FloatingIP
  properties:
    floating_network: { get_param: públic_net }
    port_id: { get_resource: slave_port }

outputs:
```

```
master_private_ip:
  description: IP address of master in private network
  value: { get_attr: [ master, first_address ] }
master_públic_ip:
  description: Floating IP address of master in públic network
  value: { get_attr: [ master_floating_ip, floating_ip_address ] }
slave_private_ip:
  description: IP address of master in private network
  value: { get_attr: [ slave, first_address ] }
slave_públic_ip:
  description: Floating IP address of slave in públic network
  value: { get_attr: [ slave_floating_ip, floating_ip_address ] }
```

Annex 4.10. Template ‘Autoscaling’

```
heat_template_version: 2013-05-23
description: AutoScaling
parameters:
 image:
   type: string
   description: Image used for servers
 flavor:
   type: string
   description: flavor used by the web servers
 database_flavor:
   type: string
   description: flavor used by the db server
 subnet_id:
   type: string
   description: subnet on which the load balancer will be located
 external_network_id:
   type: string
   description: UUID of a Neutron external network

resources:
 master:
   type: OS::Nova::Server
   properties:
     flavor: {get_param: flavor}
     image: {get_param: image}
     user_data_format: RAW
     user_data:
       str_replace:
         template: |
           #!/bin/bash -v
            cd DADES/ALOE/aloe
            runfc
         params:

 slaves:
   type: OS::Heat::AutoScalingGroup
   properties:
     min_size: 1
     max_size: 3
     resource:
       type: lb_server.yaml
       properties:
         flavor: {get_param: flavor}
         image: {get_param: image}
         pool_id: {get_resource: pool}
         metadata: {"metering.stack": {get_param: "OS::stack_id"}}
         user_data:
           str_replace:
             template: |
               #!/bin/bash -v
               cd DADES/ALOE/aloe
               runfc
               systemctl restart httpd.service
             params:
               $db_name: {get_param: database_name}

 slave_scaleup_policy:
```

```
    type: OS::Heat::ScalingPolicy
    properties:
      adjustment_type: change_in_capacity
      auto_scaling_group_id: {get_resource: asg}
      cooldown: 60
      scaling_adjustment: 1
  slave_scaledown_policy:
    type: OS::Heat::ScalingPolicy
    properties:
      adjustment_type: change_in_capacity
      auto_scaling_group_id: {get_resource: asg}
      cooldown: 60
      scaling_adjustment: -1
  cpu_alarm_high:
    type: OS::Ceilometer::Alarm
    properties:
      description: Scale-up if the average CPU > 50% for 1 minute
      meter_name: cpu_util
      statistic: avg
      period: 60
      evaluation_periods: 1
      threshold: 50
      alarm_actions:
        - {get_attr: slave_scaleup_policy, alarm_url]}
      matching_metadata: {'metadata.user_metadata.stack': {get_param: "OS::stack_id"}}
      comparison_operator: gt
  cpu_alarm_low:
    type: OS::Ceilometer::Alarm
    properties:
      description: Scale-down if the average CPU < 15% for 10 minutes
      meter_name: cpu_util
      statistic: avg
      period: 600
      evaluation_periods: 1
      threshold: 15
      alarm_actions:
        - {get_attr: [slave_scaledown_policy, alarm_url]}
      matching_metadata: {'metadata.user_metadata.stack': {get_param: "OS::stack_id"}}
      comparison_operator: lt
  monitor:
    type: OS::Neutron::HealthMonitor
    properties:
      type: TCP
      delay: 5
      max_retries: 5
      timeout: 5
  pool:
    type: OS::Neutron::Pool
    properties:
      protocol: HTTP
      monitors: [{get_resource: monitor}]
      subnet_id: {get_param: subnet_id}
      lb_method: ROUND_ROBIN
      vip:
        protocol_port: 80
  lb:
    type: OS::Neutron::LoadBalancer
    properties:
      protocol_port: 80
      pool_id: {get_resource: pool}
```

```
 # assign a floating ip address to the load balancer
 # pool.
 lb_floating:
   type: OS::Neutron::FloatingIP
   properties:
     floating_network_id: {get_param: external_network_id}
     port_id: {get_attr: [pool, vip, port_id]}

outputs:
 scale_up_url:
   value: {get_attr: [slave_scaleup_policy, alarm_url]}
 scale_dn_url:
   value: {get_attr: [slave_scaledown_policy, alarm_url]}
 pool_ip_address:
   value: {get_attr: [pool, vip, address]}
 ceilometer_query:
   value:
     str_replace:
       template: >
         ceilometer statistics -m cpu_util
         -q metadata.user_metadata.stack=stackval -p 600 -a avg
       params:
         stackval: { get_param: "OS::stack_id" }
```

Lb_server. xml

```
heat_template_version: 2013-05-23
description: A load-balancer server
parameters:
  image:
    type: string
    description: Image used for servers
  flavor:
    type: string
    description: flavor used by the servers
  pool_id:
    type: string
    description: Pool to contact
  user_data:
    type: string
    description: Server user_data
  metadata:
    type: json
  network:
    type: string
    description: Network used by the server

resources:
  server:
    type: OS::Nova::Server
    properties:
      flavor: {get_param: flavor}
      image: {get_param: image}
      metadata: {get_param: metadata}
      user_data: {get_param: user_data}
      user_data_format: RAW
      networks: [{network: {get_param: network} }]
  member:
```

```
type: OS::Neutron::PoolMember
properties:
  pool_id: {get_param: pool_id}
  address: {get_attr: [server, first_address]}
  protocol_port: 80
```