Clúster, Cloud i DevOps

Remo Suppi Boldrito

PID_00212468

Universitat Oberta de Catalunya

Índex

Introducción

Els avenços en la tecnologia han portat, d'una banda, al desenvolupament de processadors més ràpids, amb més d'un element de còmput (nuclis o *cores*) en cadascun, de baix cost i amb suport a la virtualització maquinari i, d'una altra, al desenvolupament de xarxes altament eficients. Això, juntament amb el desenvolupament de sistemes operatius com GNU/Linux, ha afavorit un canvi radical en la utilització de sistemes de processadors de nuclis múltiples i altament interconnectats, en lloc d'un únic sistema d'alta velocitat (com els sistemes vectorials o els sistemes de processament simètric SMP). A més, aquests sistemes han tingut una corba de desplegament molt ràpida, ja que la relació preu/prestacions ha estat, i ho és cada dia més, molt favorable tant per al responsable dels sistemes TIC d'una organització com per als usuaris finals en diferents aspectes: prestacions, utilitat, fiabilitat, facilitat i eficiència. D'altra banda, les creixents necessitats de còmput (i d'emmagatzematge) s'han convertit en un element tractor d'aquesta tecnologia i el seu desenvolupament, vinculats a la provisió dinàmica de serveis, contractació d'infraestructura i utilització per ús -fins i tot en minuts- (i no per compra o per lloguer), la qual cosa ha generat una evolució total i important en la forma de dissenyar, gestionar i administrar aquests centres de còmput. No menys importants han estat els desenvolupaments, a més dels sistemes operatius, per a suportar totes aquestes tecnologies, en llenguatges de desenvolupament, API i entorns (*frameworks*) perquè els desenvolupadors puguin utilitzar la potencialitat de l'arquitectura subjacent per al desenvolupament de les seves aplicacions, perquè els administradors i gestors puguin desplegar, oferir i gestionar serveis contractats i valorats per minuts d'ús o bytes d'E/S, per exemple, i perquè els usuaris finals puguin fer un ús ràpid i eficient dels sistemes de còmput sense preocupar-se gens d'allò que existeix per sota, on està situat i com s'executa.

Per aquest motiu, en aquest apartat, es desenvolupen tres aspectes bàsics que són part de la mateixa idea, en diferents aspectes, quan es desitja oferir la infraestructura de còmput d'altes prestacions. O una plataforma com a servei o la seva utilització per un usuari final en el desenvolupament d'una aplicació, que permeti utilitzar totes aquestes infraestructures de còmput distribuïdes i d'altes prestacions. En sistemes de còmput d'altes prestacions (*High Performance Computing*-HPC), podem distingir dues grans configuracions:

1) Sistemes fortament acoblats (*tightly coupled systems*): són sistemes on la memòria és compartida per tots els processadors (*shared memory systems*) i la memòria de tots aquests "es veu" (per part del programador) com una única memòria.

2) Sistemes feblement acoblats (*loosely coupled systems*): no comparteixen memòria (cada processador posseeix la seva) i es comuniquen mitjançant missatges passats a través d'una xarxa (*message passing systems*).

En el primer cas, són coneguts com a *sistemes paral·lels de còmput* (*parallel processing system*) i en el segon, com a *sistemes distribuïts de còmput* (*distributed computing systems*).

En l'actualitat, la gran majoria de sistemes (bàsicament per la seva relació preu-prestacions) són del segon tipus i es coneixen com a clústers on els sistemes de còmput estan interconnectats per una xarxa de gran amplada de banda i tots treballen en estreta col·laboració, i l'usuari final els veu com un únic equip. És important indicar que cadascun dels sistemes que integra el clúster al seu torn pot tenir més d'un processador amb més d'un *core* en cadascun, per la qual cosa el programador haurà de tenir en compte aquestes característiques quan desenvolupi les seves aplicacions (llenguatge, memòria compartida/distribuïda, nivells de caché, etc.) i així podrà aprofitar tota la potencialitat de l'arquitectura agregada. El tipus més comú de clúster és l'anomenat Beowulf, que és un clúster implementat amb múltiples sistemes de còmput (generalment similars, però poden ser heterogenis) i interconnectats per una xarxa d'àrea local (generalment Ethernet, però hi ha xarxes més eficients com Infiniband o Myrinet). Alguns autors el denominen *MPP* (*massively parallel processing*) quan és un clúster, però disposen de xarxes especialitzades d'interconnexió (mentre que els clústers utilitzen maquinari estàndard a les seves xarxes) i estan formats per un alt nombre de recursos de còmput (1.000 processadors, no més). Avui dia, la majoria de sistemes publicats en el TOP500 (http://www.top500.org/system/177999) són clústers, i en l'última llista publicada (2014) ocupava el primer lloc l'ordinador Tianhe-2 (Xina) amb 3,1 milions de *cores*, 1 petabyte de memòria RAM (1.000 Tbytes) i un consum de 17 MW.

L'altre concepte vinculat als desenvolupaments tecnològics esmentats a les noves formes d'entendre el món de les TIC en l'actualitat és el *cloud computing* (o informàtica/serveis en el núvol), que sorgeix com a concepte d'oferir serveis d'informàtica a través d'Internet en què l'usuari final no té coneixement d'on s'estan executant les seves aplicacions i tampoc necessita ser un expert per a contactar, pujar i executar les seves aplicacions en minuts. Els proveïdors d'aquest tipus de servei tenen recursos (generalment distribuïts a tot el món) i permeten que els seus usuaris contractin i gestionin aquests recursos en línia i sense la major intervenció i en forma gairebé automàtica, la qual cosa permet reduir els costos tenint avantatges (a més no s'ha d'instal·lar-mantenir la infraestructura física –ni l'obra civil) com la fiabilitat, flexibilitat, rapidesa en l'aprovisionament, facilitat d'ús, pagament per ús, contractació d'allò que es necessita, etc. Òbviament, també té els seus desavantatges, com són la dependència d'un proveïdor i la centralització de les aplicacions/emmagatzematge

de dades, servei vinculat a la disponibilitat d'accés a Internet, qüestions de seguretat, ja que les dades sensibles del negoci no resideixen en les instal·lacions de les empreses i poden generar riscos de sostracció/robatori d'informació, confiabilitat dels serveis prestats pel proveïdor (sempre és necessari signar una SLA –contracte de qualitat de servei), tecnologia susceptible al monopoli, serveis estàndard (només els oferts pel proveïdor), escalabilitat o privadesa.

Un tercer concepte vinculat a aquestes tecnologies, però probablement més del costat dels desenvolupadors d'aplicacions i *frameworks*, és el de DevOps. DevOps sorgeix de la unió de les paraules *Development* (desenvolupament) i *Operations* (operacions), i es refereix a una metodologia de desenvolupament de programari que se centra en la comunicació, col·laboració i integració entre desenvolupadors de programari i els professionals d'operacions / administradors a les tecnologies de la informació (TI). DevOps es presenta com una metodologia que dóna resposta a la relació entre el desenvolupament del programari i les operacions TI, tenint com a objectiu que els productes i serveis de programari desenvolupats per una entitat es puguin fer més eficientment, tinguin una alta qualitat i a més siguin segurs i fàcils de mantenir. El terme DevOps és relativament nou i va ser popularitzat mitjançant *DevOps Open Days* (Bèlgica, 2009), i com a metodologia de desenvolupament ha anat guanyant adeptes des de grans a petites companyies, per a fer més i millors productes i serveis i, sobretot, a causa de diferents factors amb una forta presència avui dia com l'ús dels processos i metodologies de desenvolupament àgil, necessitat d'una major taxa de versions, àmplia disponibilitat d'entorns virtualitzats/*cloud*, major automatització de centres de dades i augment de les eines de gestió de configuració.

En aquest mòdul, es veuran diferents formes de crear i programar un sistema de còmput distribuït (clúster/*cloud*), les eines i biblioteques més importants per a complir aquest objectiu i els conceptes i eines vinculat a les metodologies DevOps.

Objectius

En els materials didàctics d'aquest mòdul, trobareu els continguts i les eines procedimentals per a aconseguir els objectius següents:

1. Analitzar les diferents infraestructures i eines per al còmput d'altes prestacions (inclosa la virtualització) (HPC).
2. Configurar i instal·lar un clúster d'HPC i les eines de monitoratge corresponents.
3. Instal·lar i desenvolupar programes d'exemples en les principals API de programació: Posix Threads, OpenMPI, i OpenMP.
4. Instal·lar un clúster específic basat en una distribució *ad hoc* (Rocks).
5. Instal·lar una infraestructura per a prestar serveis en *cloud* (IasS).
6. Eines DevOps per a automatitzar un centre de dades i gestions de la configuració.

1. Clusterització

çLa història dels sistemes informàtics és molt recent (es pot dir que comença en la dècada dels seixanta). Al principi, eren sistemes grans, pesats, cars, de pocs usuaris experts, no accessibles i lents. En la dècada dels setanta, l'evolució va permetre millores substancials dutes a terme per tasques interactives (*interactive jobs*), temps compartit (*time sharing*), terminals i amb una considerable reducció de la grandària. La dècada dels vuitanta es caracteritza per un augment notable de les prestacions (fins avui dia) i una reducció de la grandària en els anomenats *microordinadors*. La seva evolució ha estat a través de les estacions de treball (*workstations*) i els avenços en xarxes (LAN de 10 Mb/s i WAN de 56 kB/s el 1973 a LAN d'1/10 Gb/s i WAN amb ATM, *asynchronous transfer mode* d'1,2 Gb/s en l'actualitat o xarxes d'alt rendiment com Infiniband -96Gb/s- o Myrinet -10Gb/s-), que és un factor fonamental en les aplicacions multimèdia actuals i d'un futur proper. Els sistemes distribuïts, per la seva banda, van començar la seva història en la dècada dels setanta (sistemes de 4 o 8 ordinadors) i el seu salt a la popularitat el van fer en la dècada dels noranta. Si bé la seva administració, instal·lació i manteniment poden tenir una certa complexitat (cada vegada menys) perquè continuen creixent en grandària, les raons bàsiques de la seva popularitat són l'increment de prestacions que presenten en aplicacions intrínsecament distribuïdes (aplicacions que per la seva naturalesa són distribuïdes), la informació compartida per un conjunt d'usuaris, la compartició de recursos, l'alta tolerància a les fallades i la possibilitat d'expansió incremental (capacitat d'agregar més nodes per a augmentar les prestacions i de manera incremental). Un altre aspecte molt important en l'actualitat és la possibilitat, en aquesta evolució, de la virtualització. Les arquitectures cada vegada més eficients, amb sistemes *multicores*, han permès que la virtualització de sistemes es transformi en una realitat amb tots els avantatges (i possibles desavantatges) que això comporta.

1.1. Virtualització

La virtualització és una tècnica que està basada en l'abstracció dels recursos d'un ordinador, anomenada *Hypervisor* o VMM (*Virtual Machine Monitor*) que crea una capa de separació entre el maquinari de la màquina física (*host*) i el sistema operatiu de la màquina virtual (*virtual machine*, *guest*), i és un mitjà per a crear una "versió virtual" d'un dispositiu o recurs, com un servidor, un dispositiu d'emmagatzematge, una xarxa o fins i tot un sistema operatiu, on es divideix el recurs en un o més entorns d'execució. Aquesta capa de programari (VMM) maneja, gestiona i administra els quatre recursos principals d'un ordinador (CPU, memòria, xarxa i emmagatzematge) i els reparteix de mane-

ra dinàmica entre totes les màquines virtuals definides en l'ordinador central. D'aquesta manera, ens permet tenir diversos ordinadors virtuals executant-se sobre el mateix ordinador físic.

La màquina virtual, en general, és un sistema operatiu complet que s'executa com si estigués instal·lat en una plataforma de maquinari autònoma.

Enllaç d'interès

Per a saber més sobre virtualització, podeu visitar http://en.wikipedia.org/wiki/Virtualization.
Es pot consultar una llista completa de programes de virtualització a http://en.wikipedia.org/wiki/Comparison_of_platform_virtual_machines.

1.1.1. Plataformes de virtualització

Els exemples més comuns de plataforma de virtualització amb llicències GPL o similars són Xen, KVM, Qemu (emulació), OpenVz, VirtualBox i Oracle VM (només el servidor); i entre les plataforma propietàries (algunes com *free to use*) VMware ESX/i, Virtual PC/Hyper-V, Parallels i Virtuozzo. Hi ha diferents taxonomies per a classificar la virtualització (i algunes plataformes poden executar-se en més d'una categoria), essent la més coneguda la següent:

1) Virtualització per HW: considerant com a *Full Virtualization* quan la màquina virtual simula tot el HW per a permetre a un sistema *guest* executar-se sense modificacions (sempre que estigui dissenyat per al mateix conjunt d'instruccions). Va ser la primera generació de VM per a processadors x86 i el que fa és una captura d'instruccions que accedeixen de manera prioritària a la CPU i les transforma en una crida a la VM perquè siguin emulades per programari. En aquest tipus de virtualització, poden executar-se Parallels, VirtualBox, Virtual Iron, Oracle VM, Virtual PC, Hyper-V, VMware per exemple.

2) Virtualització assistida per maquinari, on el maquinari dóna suport que facilita la construcció i treball de la VM i millora notablement les prestacions (a partir del 2005/6, Intel i AMD donen aquest suport com VT-x i AMD-V, respectivament). Els principals avantatges, respecte a altres formes de virtualització, és que no s'ha de tocar el sistema operatiu *guest* (com en paravirtualització) i s'obtenen millors prestacions, però com a contrapartida es necessita suport explícit en la CPU, la qual cosa no està disponible en tots els processadors x86/86_64. En aquest tipus de virtualització poden executar-se KVM, VMware Fusion, Hyper-V, Virtual PC, Xen, Parallels, Oracle VM, VirtualBox.

3) Paravirtualització: és una tècnica de virtualització en què la VM no necessàriament simula l'HW, sinó que presenta una API que pot ser utilitzada pel sistema *guest* (per la qual cosa, s'ha de tenir accés al codi font per a reemplaçar les crides d'un tipus per un altre). Aquest tipus de crides a aquesta API es denominen *hypercall* i Xen pot executar-se en aquesta forma.

4) Virtualització parcial: és el que es denomina *Address Space Virtualization*. La màquina virtual simula múltiples instàncies de l'entorn subjacent del maquinari (però no de tot), particularment l'*address space*. Aquest tipus de virtualit-

zació accepta compartir recursos i allotjar processos, però no permet instàncies separades de sistemes operatius *guest* i es troba en desús actualment.

5) Virtualització en un nivell del sistema operatiu: en aquest cas, la virtualització permet tenir múltiples instàncies aïllades i segures del servidor, totes executant-se sobre el mateix servidor físic i on el sistema operatiu *guest* coincidirà amb el sistema operatiu base del servidor (s'utilitza el mateix kernel). Plataformes dins d'aquest tipus de virtualització són FreeBSD jails (el pioner), OpenVZ, Linux-VServer, LXC, Virtuozzo.

La diferència entre instal·lar dos sistemes operatius i virtualitzar dos sistemes operatius és que en el primer cas tots els sistemes operatius que tinguem instal·lats funcionaran de la mateixa manera que si estiguessin instal·lats en diferents ordinadors, i necessitarem un gestor d'arrencada que en encendre l'ordinador ens permeti triar quin sistema operatiu volem utilitzar, però només en podrem tenir funcionant simultàniament un. En canvi, la virtualització permet executar moltes màquines virtuals amb els seus sistemes operatius i canviar de sistema operatiu com si es tractés de qualsevol altre programa; no obstant això, s'han de valorar molt bé les qüestions relacionades amb les prestacions, ja que si l'HW subjacent no és l'adequat, podrem notar moltes diferències en les prestacions entre el sistema operatiu instal·lat en base o el virtualitzat.

Entre els principals avantatges de la virtualització, hi ha:

1) Consolidació de servidors i millora de l'eficiència de la inversió en HW amb reutilització de la infraestructura existent.

2) Ràpid desplegament de nous serveis amb balanç dinàmic de càrrega i reducció dels sobredimensionaments de la infraestructura.

3) Increment d'*uptime* (temps que el sistema està al 100% en la prestació de serveis), increment de la tolerància a fallades (sempre que existeixi redundància física) i eliminació del temps de parada per a manteniment del sistema físic (migració de les màquines virtuals).

4) Manteniment a cost acceptable d'entorns de programari obsolets però necessaris per al negoci.

5) Facilitat de disseny i test de noves infraestructures i entorns de desenvolupament amb un baix impacte en els sistemes de producció i ràpida posada en marxa.

6) Millora de TCO (*Total Cost of Ownership*) i ROI (*Return on Investment*).

7) Menor consum d'energia que en servidors físics equivalents.

Com a desavantatges, podem esmentar:

1) Augmenta la probabilitat de fallades si no es considera redundància / alta disponibilitat (si es consoliden 10 servidors físics en un de potent equivalent, amb servidors virtualitzats, i deixen de funcionar tots els servidors en aquest, deixaran de prestar servei).

2) Rendiment inferior (possible) en funció de la tècnica de virtualització utilitzada i recursos disponibles.

3) Proliferació de serveis i màquines que incrementen les despeses d'administració/gestió (bàsicament, per l'efecte derivat de la facilitat de desplegament es tendeix a tenir-ne més dels necessaris).

4) Infraestructura desaprofitada (possible), ja que és habitual comprar una infraestructura major que la necessària en aquest moment per al possible creixement futur immediat.

5) Poden existir problemes de portabilitat, maquinari específic no suportat i compromís a llarg termini amb la infraestructura adquirida.

6) La presa de decisions en la selecció del sistema amfitrió pot ser complicada o condicionant.

Com és possible observar, els desavantatges es poden resoldre amb una planificació i presa de decisions adequada, i aquesta tecnologia és habitual en la prestació de serveis i totalment imprescindible en entorns de *cloud computing* (que veurem en aquest apartat també).

1.2. Beowulf

Beowulf [3, 1, 2, 4] és una arquitectura multiordenador que pot ser utilitzada per a aplicacions paral·leles/distribuïdes. El sistema consisteix bàsicament en un servidor i un o més clients connectats (generalment) a través d'Ethernet i sense la utilització de cap maquinari específic. Per a explotar aquesta capacitat de còmput, és necessari que els programadors tinguin un model de programació distribuït que, si bé és possible mitjançant UNIX (Sockets, RPC), pot implicar un esforç considerable, ja que són models de programació a nivell de *systems calls* i llenguatge C, per exemple; però aquesta forma de treball es pot considerar de baix nivell. Un model més avançat en aquesta línia de treball són els **Posix Threads**, que permeten explotar sistemes de memòria compartida i *multicores* de manera simple i fàcil. La capa de programari (interfície de programació d'aplicacions, API) aportada per sistemes com ***Parallel Virtual Machine*** (PVM) i ***Message Passing Interface*** (MPI) facilita notablement l'abstracció del sistema i permet programar aplicacions paral·leles/distribuïdes de manera senzilla i simple. Una de les formes bàsiques de treball és la de mestre-treballadors (*master-workers*), en què existeix un servidor (mestre) que distribueix la tasca que faran els treballadors. En grans sistemes (per exemple, de

1.024 nodes) hi ha més d'un mestre i nodes dedicats a tasques especials, com per exemple entrada/sortida o monitoratge. Una altra opció no menys interessant, sobretot amb l'auge de processadors *multicores*, és **OpenMP**, que és una API per a la programació multiprocés de memòria compartida. Aquesta capa de programari permet afegir concurrència als programes sobre la base del model d'execució *fork-join* i es compon d'un conjunt de directives de compilador, rutines de biblioteca i variables d'entorn, que influencien el comportament en temps d'execució i proporcionen als programadors una interfície simple i flexible per al desenvolupament d'aplicacions paral·leles i arquitectures de CPU que puguin executar més d'un fil d'execució simultani (*hyperthreading*) o que disposin de més d'un nucli per processador accedint a la mateixa memòria compartida.

Hi ha dos conceptes que poden donar lloc a dubtes i que són Cluster Beowulf i COW (*Cluster of Workstations*). Una de les principals diferències és que Beowulf "es veu" com una única màquina on s'accedeix als nodes remotament, ja que no disposen de terminal (ni de teclat), mentre que un COW és una agrupació d'ordinadors que poden ser utilitzats tant pels usuaris del COW com per altres usuaris en forma interactiva, a través de la pantalla i el teclat. Cal considerar que Beowulf no és un programari que transforma el codi de l'usuari en distribuït ni afecta el nucli del sistema operatiu (com per exemple, Mosix). Simplement, és una forma d'agrupació (un clúster) de màquines que executen GNU/Linux i actuen com un superordinador. Òbviament, hi ha una gran quantitat d'eines que permeten obtenir una configuració més fàcil, biblioteques o modificacions al nucli per a obtenir millors prestacions, però és possible construir un clúster Beowulf a partir d'un GNU/Linux estàndard i de programari convencional. La construcció d'un clúster Beowulf de dos nodes, per exemple, es pot dur a terme simplement amb les dues màquines connectades per Ethernet mitjançant un concentrador (*hub*), una distribució de GNU/Linux estàndard (Debian), el sistema d'arxius compartit (NFS) i tenint habilitats els serveis de xarxa, com per exemple `ssh`. En aquestes condicions, es pot argumentar que es disposa d'un clúster simple de dos nodes. En canvi, en un COW no necessitem tenir coneixements sobre l'arquitectura subjacent (CPU, xarxa) necessària per a fer una aplicació distribuïda en Beowulf però sí que és necessari tenir un sistema operatiu (per exemple, Mosix) que permeti aquest tipus de compartició de recursos i distribució de tasques (a més d'una API específica per a programar l'aplicació que és necessària en els dos sistemes). El primer Beowulf es va construir el 1994 i el 1998 comencen a aparèixer sistemes Beowulf/Linux en les llistes del Top500 (Avalon en la posició 314 amb 68 *cores* i 19,3 Gflops, juny del 98).

1.2.1. Com es configuren els nodes?

Primer de tot s'ha de modificar l'arxiu */etc/hosts* perquè contingui la línia de `localhost` (amb 127.0.0.1) i la resta de noms a les IP internes dels restants nodes (aquest arxiu haurà de ser igual en tots els nodes). Per exemple:

```
    127.0.0.1 localhost
    192.168.0.254   lucix-server
I afegir les IP dels nodes (i per a tots els nodes), per exemple:
    192.168.0.1 lucix1
    192.168.0.2 lucix2
...
S'ha de crear un usuari (nteum) en tots els nodes, crear un grup i afegir aquest usuari al grup:
    groupadd beowulf
    adduser nteum beowulf
    echo umask 007 >> /home/nteum/.bash_profile
```

Així, qualsevol arxiu creat per l'usuari nteum o qualsevol dins del grup serà modificable pel grup beowulf. S'ha de crear un servidor de NFS (i els altres nodes seran clients d'aquest NFS) i exportar el directori */home*, i així tots els clients veuran el directori *$HOME* dels usuaris. Per a això, sobre el servidor s'edita el */etc/exports* i s'agrega una línia com

```
/home lucix*(rw,sync,no_root_esquaix)
```

Sobre cada node client es munta agregant en el */etc/fstab* perquè en l'arrencada el node munti el directori com a `192.168.0.254:/home /home nfs defaults 0 0`, també és aconsellable tenir en el servidor un directori */soft* (on s'instal·larà tot el programari perquè el vegin tots els nodes, i així ens evitem instal·lar aquest en cada node), i agregar-los a l'*export* com

```
/soft lucix*(rw,async,no_root_squash)
```

i en cada node agreguem en el *fstab* `192.168.0.254:/soft /soft nfs defaults 0 0`. També haurem de configurar el */etc/resolv.conf* i el `iptables` (per a fer un NAT) sobre el servidor si aquest té accés a Internet i volem que els nodes també hi tinguin accés (és molt útil sobretot a l'hora d'actualitzar el sistema operatiu).

A continuació, verifiquem que els serveis estan funcionant (tingueu en compte que l'ordre `chkconfig` pot no estar instal·lada en totes les distribucions) (en Debian, cal fer `apt-get install chkconfig`):

```
    chkconfig --list sshd
    chkconfig --list nfs
```

Que han d'estar a "on" en el nivell de treball que estiguem (generalment el nivell 3, però es pot esbrinar amb l'ordre `runlevel`) i, si no, s'hauran de fer les modificacions necessàries perquè aquests *daemons* s'iniciïn en aquest nivell (p. ex. `chkconfig name_service on; service name_service start`). Si bé estarem en un xarxa privada, per a treballar de manera segura és important treballar amb `ssh` i mai amb `rsh` o `rlogin`, per la qual cosa hauríem de generar les claus per a interconnectar en mode segur les màquines-usuari nteum sense `passwd`. Per a això, modifiquem (traiem el comentari `#`), de */etc/ssh/sshd_config* en les següents línies:

```
    RSAAuthentication yes
    AuthorizedKeysFile .ssh/authorized_keys
```

Reiniciem el servei (`service sshd restart`) i ens connectem amb l'usuari que desitgem i generem les claus:

```
    ssh-keygen -t rsa
```

En el directori *$HOME/.ssh* s'hauran creat els arxius `id_rsa` i `id_rsa.pub` en referència a la clau privada i a la pública respectivament. Manualment, podem copiar `id_rsa.pub` en un arxiu anomenat `authorized_keys` en el mateix directori (ja que en estar compartit per NFS, serà el mateix directori que veuran tots els nodes i verifiquem els permisos: 600 per al directori *.ssh*, *authorized_keys* i *id_rsa* i 644 per a *id_rsa.pub*). És convenient també instal·lar NIS sobre el clúster, i així evitem haver de definir cada usuari en els nodes i un servidor DHCP per a definir tots els paràmetres de xarxa que cada node sol·liciti durant l'etapa de *boot*. Per a això, consulteu l'apartat de servidors i el de xarxa de l'assignatura Administració GNU/Linux, on s'expliquen amb detall aquestes configuracions.

A partir d'aquí, ja tenim un clúster Beowulf per a executar aplicacions amb interfície a MPI (com es veurà en els següents subapartats) per a aplicacions distribuïdes (també poden ser aplicacions amb interfície a PVM, però és recomanable MPI per qüestions d'eficiència i prestacions). Hi ha sobre diferents distribucions (Debian, FC incloses) l'aplicació `system-config-cluster`, que permet configurar un clúster sobre la base d'una eina gràfica o `clusterssh` o `dish`, que permeten gestionar i executar ordres en un conjunt de màquines de manera simultània (ordres útils quan necessitem fer operacions sobre tots els nodes del clúster, per exemple, apagar-los, reiniciar-los o fer còpies d'un arxiu a tots aquests).

Enllaç d'interès

Informació addicional: https://wiki.debian.org/highperformancecomputing

1.3. Beneficis del còmput distribuït

Quins són els beneficis del còmput en paral·lel? Veurem això amb un exemple [3]. Considerem un programa per a sumar nombres (per exemple, 4+5+6+...) anomenat `sumdis.c`:

```
#include <stdio.h>

int main (int argc, char** argv){
long inicial, final, resultat, tmp;
    if (argc < 2) {
   printf (" Ús: %s Núm. inicial Núm. final\n",argv[0]);
     return (4); }
    else {
   inicial = atoll(argv[1]);
   final = atoll(argv[2]);
   resultat = 0;}
for (tmp = inicial; tmp <= final; tmp++){resultat += tmp; };
```

```
    printf("%lu\n", resultat);
    return 0;
}
```

Ho compilem amb `gcc -o sumdis sumdis.c` i si mirem l'execució d'aquest programa, per exemple, amb

```
 time ./sumdis 1 1000000
500000500000

    real 0m0.005s
    user 0m0.000s
    sys     0m0.004s
```

es podrà observar que el temps en una màquina Debian 7.5 sobre una màquina virtual (Virtualbox) amb processador i7 és de 5 mil·lèsimes de segon. Si, en canvi, fem des d'1 a 16 milions, el temps real puja fins a 0,050 s, és a dir, 10 vegades més, per la qual cosa, si es consideren 1.600 milions, el temps serà de prop de 4 segons.

La idea bàsica del còmput distribuït és repartir el treball. Així, si disposem d'un clúster de quatre màquines (lucix1–lucix4) amb un servidor i on l'arxiu executable es comparteix per NFS, és interessant dividir l'execució mitjançant ssh de manera que el primer sumi d'1 a 400.000.000, el segon, de 400.000.001 a 800.000.000, el tercer, de 800.000.001 a 1.200.000.000, i el quart, d'1.200.000.001 a 1.600.000.000. Les següents ordres mostren una possibilitat. Considerem que el sistema té el directori */home* compartit per NFS i l'usuari **adminp**, que té adequadament configurades les claus per a executar el codi sense *password* sobre els nodes, executarà:

```
    mkfifo out1     Crea una cua fifo en /home/adminp
    ./distr.sh & time cat out1 | awk '{total + = $1 } END {printf "%lf", total}'
```

S'executa l'ordre `distr.sh`; es recol·lecten els resultats i se sumen mentre es mesura el temps d'execució. El *shell script* `distr.sh` pot ser una cosa com:

```
ssh lucix1 /home/nteum/sumdis 1 400000000 > /home/nteum/out1 < /dev/null &
ssh lucix2 /home/nteum/sumdis 400000001 800000000 > /home/nteum/out1 < /dev/null &
ssh lucix3 /home/nteum/sumdis 800000001 1200000000 > /home/nteum/out1 < /dev/null &
ssh lucix4 /home/nteum/sumdis 1200000001 1600000000 > /home/nteum/out1 < /dev/null &
```

Podrem observar que el temps es redueix notablement (aproximadament en un valor proper a 4) i no exactament de manera lineal, però molt propera. Òbviament, aquest exemple és molt simple i només vàlid per a finalitats demostratives. Els programadors utilitzen biblioteques que els permeten portar a terme el temps d'execució, la creació i comunicació de processos en un sistema distribuït (per exemple MPI o OpenMP).

1.3.1. Com cal programar per a aprofitar la concurrència?

Hi ha diverses maneres d'expressar la concurrència en un programa. Les tres més comunes són les següents:

1) Utilitzant fils (o processos) en el mateix processador (multiprogramació amb superposició del còmput i l'E/S).

2) Utilitzant fils (o processos) en sistemes *multicore*.

3) Utilitzant processos en diferents processadors que es comuniquen mitjançant missatges (MPS, *Message Passing System*).

Aquests mètodes poden ser implementats sobre diferents configuracions de maquinari (memòria compartida o missatges) i, si bé tots dos mètodes tenen els seus avantatges i desavantatges, els principals problemes de la memòria compartida són les limitacions en l'escalabilitat (ja que tots els *cores*/processadors utilitzen la mateixa memòria i el nombre d'aquests en el sistema està limitat per l'amplada de banda de la memòria) i, en els sistemes de pas de missatges, la latència i velocitat dels missatges a la xarxa. El programador haurà d'avaluar quin tipus de prestacions necessita, les característiques de l'aplicació subjacent i el problema que es desitja solucionar. No obstant això, amb els avenços de les tecnologies de *multicores* i de xarxa, aquests sistemes han crescut en popularitat (i en quantitat). Les API més comunes avui dia són Posix Threads i OpenMP per a memòria compartida i MPI (en les seves versions OpenMPI o Mpich) per a pas de missatges. Com hem esmentat anteriorment, hi ha una altra biblioteca molt difosa per a passos de missatges, anomenada PVM, però la versatilitat i prestacions que s'obtenen amb MPI l'ha deixat relegada a aplicacions petites o per a aprendre a programar en sistemes distribuïts. Aquestes biblioteques, a més, no limiten la possibilitat d'utilitzar fils (encara que en un nivell local) i tenir concurrència entre processament i entrada/sortida.

Per a portar a terme una aplicació paral·lela/distribuïda, es pot partir de la versió sèrie o mirar l'estructura física del problema i determinar quines parts poden ser concurrents (independents). Les parts concurrents seran candidates a reescriure's com a codi paral·lel. A més, s'ha de considerar si és possible reemplaçar les funcions algebraiques per les seves versions paral·lelitzades (per exemple, ScaLapack *Scalable Linear Algebra Package* (es poden provar en Debian els diferents programes de prova que es troben en el paquet scalapack-mpi-test, per exemple i directament, instal·lar les biblioteques libscalapack-mpi-dev). També és convenient esbrinar si hi ha alguna aplicació similar paral·lela que pugui orientar-nos sobre la mode de construcció de l'aplicació paral·lel*.

*http://www.mpich.org/

Paral·lelitzar un programa no és una tasca fàcil, ja que s'ha de tenir en compte la **llei d'Amdahl**, que afirma que l'increment de velocitat (*speedup*) està limitat per la fracció de codi (f), que pot ser paral·lelitzat de la següent manera:

$$\text{speedup} = \frac{1}{1-f}$$

Aquesta llei implica que amb una aplicació seqüencial $f = 0$ i l'speedup = 1, mentre que amb tot el codi paral·lel $f = 1$ i l'speedup es fa infinit. Si considerem valors possibles, un 90% ($f = 0{,}9$) del codi paral·lel significa un speedup igual a 10, però amb $f = 0{,}99$ l'speedup és igual a 100. Aquesta limitació es pot evitar amb algorismes escalables i diferents models de programació d'aplicació (paradigmes):

1) Mestre-treballador: el mestre inicia a tots els treballadors i coordina el seu treball i el d'entrada/sortida.

2) *Single Process Multiple Data* (SPMD): el mateix programa que s'executa amb diferents conjunts de dades.

3) Funcional: diversos programes que porten a terme una funció diferent en l'aplicació.

En resum, podem concloure:

1) Proliferació de màquines multitasca (multiusuari) connectades per xarxa amb serveis distribuïts (NFS i NIS).

2) Són sistemes heterogenis amb sistemes operatius de tipus NOS (*Networked Operating System*), que ofereixen una sèrie de serveis distribuïts i remots.

3) La programació d'aplicacions distribuïdes es pot efectuar en diferents nivells:

a) Utilitzant un model client-servidor i programant a baix nivell (*sockets*) o fent servir memòria compartida a baix nivell (Posix Threads).

b) El mateix model, però amb API d'"alt" nivell (OpenMP, MPI).

c) Utilitzant altres models de programació com, per exemple, programació orientada a objectes distribuïts (RMI, CORBA, Agents, etc.).

1.4. Memòria compartida. Models de fils (*threading*)

Normalment, en una arquitectura client-servidor, els clients sol·liciten als servidors determinats serveis i esperen que aquests els contestin amb la major eficàcia possible. Per a sistemes distribuïts amb servidors amb una càrrega molt

alta (per exemple, sistemes d'arxius de xarxa, bases de dades centralitzades o distribuïdes), el disseny del servidor es converteix en una qüestió crítica per a determinar el rendiment general del sistema distribuït. Un aspecte crucial en aquest sentit és trobar la manera òptima de manejar l'E/S, tenint en compte el tipus de servei que ofereix, el temps de resposta esperat i la càrrega de clients. No hi ha un disseny predeterminat per a cada servei, i escollir el correcte dependrà dels objectius i restriccions del servei i de les necessitats dels clients.

Les preguntes que hem de contestar abans de triar un determinat disseny són les següents: quant temps es triga en un procés de sol·licitud del client?, quantes d'aquestes sol·licituds és probable que arribin durant aquest temps?, quant temps pot esperar el client?, quant afecta aquesta càrrega del servidor les prestacions del sistema distribuït? A més, amb l'avenç de la tecnologia de processadors ens trobem que disposem de sistemes *multicore* (múltiples nuclis d'execució) que poden executar seccions de codi independents. Si es dissenyen els programes en forma de múltiples seqüències d'execució i el sistema operatiu ho suporta (i GNU/Linux és un d'aquests), l'execució dels programes es reduirà notablement i s'incrementaran en forma (gairebé) lineal les prestacions en funció dels *cores* de l'arquitectura.

1.4.1. Multifils (*multithreading*)

Segons les últimes tecnologies en programació per a aquest tipus d'aplicacions (i així ho demostra l'experiència), els dissenys més adequats són aquells que utilitzen models de multifils (*multithreading models*), en els quals el servidor té una organització interna de processos paral·lels o fils cooperants i concurrents.

> Un fil (*thread*) és una seqüència d'execució (fil d'execució) d'un programa, és a dir, diferents parts o rutines d'un programa que s'executen concurrentment en un únic processador i accediran a les dades compartides al mateix temps.

Quins avantatges aporta això respecte a un programa seqüencial? Considerem que un programa té tres rutines A, B i C. En un programa seqüencial, la rutina C no s'executarà fins que s'hagin executat A i B. Si, en canvi, A, B i C són fils, les tres rutines s'executaran concurrentment i, si en aquestes hi ha E/S, tindrem concurrència d'execució amb E/S del mateix programa (procés), cosa que millorarà notablement les prestacions d'aquest programa. Generalment, els fils estan continguts dins d'un procés, i diferents fils d'un mateix procés poden compartir alguns recursos, mentre que diferents processos no. L'execució de múltiples fils en paral·lel necessita el suport del sistema operatiu i en els processadors moderns hi ha optimitzacions del processador per a suportar models multifil (*multithreading*) a més de les arquitectures *multicores* on hi ha múltiples nuclis i on cadascun pot executar un *thread*.

Generalment, hi ha quatre models de disseny per fils (en ordre de complexitat creixent):

1) Un fil i un client: en aquest cas, el servidor entra en un bucle sense fi, escoltant per un port i davant la petició d'un client s'executen els serveis en el mateix fil. Altres clients hauran d'esperar que acabi el primer. És fàcil d'implementar però només atén un client alhora.

2) Un fil i diversos clients amb selecció: en aquest cas, el servidor utilitza un sol fil, però pot acceptar múltiples clients i multiplexar el temps de CPU entre aquests. Es necessita una gestió més complexa dels punts de comunicació (*sockets*), però permet crear serveis més eficients, encara que presenta problemes quan els serveis necessiten una alta càrrega de CPU.

3) Un fil per client: és, probablement, el model més popular. El servidor espera per peticions i crea un fil de servei per a atendre cada nova petició dels clients. Això genera simplicitat en el servei i una alta disponibilitat, però el sistema no escala amb el nombre de clients i pot saturar el sistema molt ràpidament, ja que el temps de CPU dedicat davant una gran càrrega de clients es redueix notablement i la gestió del sistema operatiu pot ser molt complexa.

4) Servidor amb fils en granja (*worker threads*): aquest mètode és més complex però millora l'escalabilitat dels anteriors. Hi ha un nombre fix de fils treballadors (*workers*) als quals el fil principal distribueix el treball dels clients. El problema d'aquest mètode és l'elecció del nombre de treballadors: amb un nombre elevat, cauran les prestacions del sistema per saturació; amb un nombre massa baix, el servei serà deficient (els clients hauran d'esperar). Normalment, serà necessari sintonitzar l'aplicació per a treballar amb un determinat entorn distribuït.

Hi ha diferents formes d'expressar en un nivell de programació amb fils: paral·lelisme en un nivell de tasques o paral·lelisme a través de les dades. Triar el model adequat minimitza el temps necessari per a modificar, depurar i sintonitzar el codi. La solució a aquesta disjuntiva és descriure l'aplicació en termes de dos models basats en un treball en concret:

- Tasques paral·leles amb fils independents que poden atendre tasques independents de l'aplicació. Aquestes tasques independents seran encapsulades en fils que s'executaran asincrònicament i s'hauran d'utilitzar biblioteques com Posix Threads (Linux/Unix) o Win32 Thread API (Windows), que han estat dissenyades per a suportar concurrència en un nivell de tasca.

- Model de dades paral·leles per a calcular llaços intensius; és a dir, la mateixa operació ha de repetir-se un nombre elevat de vegades (per exemple, comparar una paraula amb les paraules d'un diccionari). Per a aquest cas, és possible encarregar la tasca al compilador de l'aplicació o, si no és possible, que el programador descrigui el paral·lelisme utilitzant l'en-

torn OpenMP, que és una API que permet escriure aplicacions eficients sota aquest tipus de models.

Una aplicació d'informació personal (*Personal Information Manager*) és un bon exemple d'una aplicació que conté concurrència en un àmbit de tasques (per exemple, accés a la base de dades, llibreta d'adreces, calendari, etc.). Això podria ser en pseudocodi:

```
Function addressBook;
Function inBox;
Function calendar;
Program PIM     {
    CreateThread (addressBook);
    CreateThread (inBox);
    CreateThread (calendar);   }
```

Podem observar que hi ha tres execucions concurrents sense relació. Un altre exemple d'operacions amb paral·lelisme de dades podria ser un corrector d'ortografia, que en pseudocodi seria: `Function SpellCheck {loop (word = 1, words_in_file) compareToDictionary (word);}`

S'ha de tenir en compte que tots dos models (fils paral·lels i dades paral·leles) poden existir en una mateixa aplicació. A continuació, es mostrarà el codi d'un productor de dades i un consumidor de dades basat en Posix Threads. Per a compilar sobre Linux, per exemple, s'ha d'utilitzar `gcc -o pc pc.c -lptread`.

```
#include <pthread.h>
#include <stdio.h>
#include <unistd.h>
#include <stdlib.h>
#define QUEUESIZE 10
#define LOOP 20

void *producer (void *args);
void *consumer (void *args);
typedef struct { /* Estructura del buffer compartit i descriptors de threads */
int buf[QUEUESIZE]; long head, tail; int full, empty;
pthread_mutex_t *mut; pthread_cond_t *notFull, *notEmpty;
    } queue;
queue *queueInit (void);   /* Prototip de funció: inicialització del buffer */
void queueDelete (queue *q); /* Prototip de funció: esborrament del buffer*/
void queueAdd (queue *q, int in); /* Prototip de funció: inserir element en el buffer  */
void queueDel (queue *q, int *out); /* Prototip de funció: treure element del buffer  */

int main () {
queue *fifo; pthread_t pro, con; fifo = queueInit ();
if (fifo ==  NULL) { fprintf (stderr, " Error en crear buffer.\n");  exit (1);   }
pthread_create (&pro, NULL, producer, fifo);   /* Creació del thread productor */
pthread_create (&con, NULL, consumer, fifo); /* Creació del thread consumidor*/
pthread_join (pro, NULL);   /* main () espera fins que s'acabin els dos threads */
pthread_join (con, NULL);
queueDelete (fifo);       /* Eliminació del buffer compartit */
return 0; }               /* Fi */

void *producer (void *q) { /*Funció del productor */
queue *fifo; int i;
fifo = (queue *)q;
for (i = 0; i < LOOP; i++) {    /* Inserit en el buffer elements=LOOP*/
pthread_mutex_lock (fifo->mut);   /* Semàfor per a entrar a inserir */
```

```
while (fifo->full) {
printf ("Productor: queue FULL.\n");
pthread_cond_wait (fifo->notFull, fifo->mut);  }
/* Bloqueig del productor si el buffer està ple, i allibera el semàfor mut
    perquè hi pugui entrar el consumidor. Continuarà quan el consumidor executi
    pthread_cond_signal (fifo->notFull);*/
         queueAdd (fifo, i); /* Inserit element en el buffer */
                   pthread_mutex_unlock (fifo->mut); /* Allibero el semàfor */
         pthread_cond_signal (fifo->notEmpty);/*Desbloqueig consumidor si està bloquejat*/
         usleep (100000);  /* Dormo 100 mseg per permetre que el consumidor s'activi */
}
return (NULL); }

void *consumer (void *q) { /*Funció del consumidor */
queue *fifo; int i, d;
fifo = (queue *)q;
for (i = 0; i < LOOP; i++) { /* Traiem del buffer elements=LOOP*/
pthread_mutex_lock (fifo->mut); /* Semàfor per a entrar a treure */
while (fifo->empty) {
printf (" Consumidor: queue EMPTY.\n");
pthread_cond_wait (fifo->notEmpty, fifo->mut); }
    /* Bloqueig del consumidor si el buffer està buit, alliberant el semàfor mut
    perquè pugui entrar el productor. Continuarà quan el consumidor executi
    pthread_cond_signal (fifo->notEmpty);*/
queueDel (fifo, &d); /* Trec element del buffer  */
pthread_mutex_unlock (fifo->mut); /* Allibero el semàfor */
pthread_cond_signal (fifo->notFull); /*Desbloquejo productor si està bloquejat*/
printf (" Consumidor: Rebut %d.\n", d);
usleep(200000);/* Dormo 200 mseg per a permetre que el productor s'activi */
}
return (NULL); }

queue *queueInit (void) {
queue *q;
q = (queue *)malloc (sizeof (queue)); /* Creació del buffer */
if (q == NULL) return (NULL);
q->empty = 1;  q->full = 0;  q->head = 0;  q->tail = 0;
q->mut = (pthread_mutex_t *) malloc (sizeof (pthread_mutex_t));
pthread_mutex_init (q->mut, NULL); /* Creació del semàfor */
q->notFull = (pthread_cond_t *) malloc (sizeof (pthread_cond_t));
pthread_cond_init (q->notFull, NULL); /* Creació de la variable condicional notFull*/
q->notEmpty = (pthread_cond_t *) malloc (sizeof (pthread_cond_t));
pthread_cond_init (q->notEmpty, NULL);   /* Creació de la variable condicional notEmpty*/
return (q); }

void queueDelete (queue *q) {
pthread_mutex_destroy (q->mut);  free (q->mut);
pthread_cond_destroy (q->notFull); free (q->notFull);
pthread_cond_destroy (q->notEmpty); free (q->notEmpty);
free (q); }

void queueAdd (queue *q, int in) {
q->buf[q->tail] = in; q->tail++;
if (q->tail == QUEUESIZE) q->tail = 0;
if (q->tail == q->head) q->full = 1;
q->empty = 0;
return; }

void queueDel (queue *q, int *out){
*out = q->buf[q->head]; q->head++;
if (q->head == QUEUESIZE) q->head = 0;
if (q->head == q->tail) q->empty = 1;
q->full = 0;
return; }
```

1.5. OpenMP

L'OpenMP (*Open-Multi Processing*) és una interfície de programació d'aplicacions (API) amb suport multiplataforma per a la programació en C/C++ i Fortran

de processos amb ús de memòria compartida sobre plataformes Linux/Unix (i també Windows). Aquesta infraestructura es compon d'un conjunt de directives del compilador, rutines de la biblioteca i variables d'entorn que permeten aprofitar recursos compartits en memòria i en temps d'execució. Definit conjuntament per un grup dels principals fabricants de maquinari i programari, OpenMP permet utilitzar un model escalable i portàtil de programació, que proporciona als usuaris una interfície simple i flexible per al desenvolupament, sobre plataformes paral·leles, d'aplicacions d'escriptori fins a aplicacions d'altes prestacions sobre superordinadors. Una aplicació construïda amb el model híbrid de la programació paral·lela pot executar-se en un ordinador utilitzant tant OpenMP com Message Passing Interface (MPI) [5].

OpenMP és una implementació multifil, mitjançant la qual un fil mestre divideix la tasques sobre un conjunt de fils treballadors. Aquests fils s'executen de manera simultània i l'entorn d'execució porta a terme l'assignació d'aquests als diferents processadors de l'arquitectura. La secció del codi que està dissenyada per a funcionar en paral·lel està marcada amb una directiva de preprocessament que crearà els fils abans que la secció s'executi. Cada fil tindrà un identificador (`id`) que s'obtindrà a partir d'una funció (en C/C++ `omp_get_thread_num()`) i, després de l'execució paral·lela, els fils s'uniran de nou en la seva execució sobre el fil mestre, que continuarà amb l'execució del programa. Per defecte, cada fil executarà una secció paral·lela de codi independent però es poden declarar seccions de "treball compartit" per a dividir una tasca entre els fils, de manera que cada fil executi part del codi assignat. D'aquesta manera, és possible tenir en un programa OpenMP paral·lelisme de dades i paral·lelisme de tasques convivint.

Els principals elements d'OpenMP són les sentències per a la creació de fils, la distribució de càrrega de treball, la gestió de dades d'entorn, la sincronització de fils i les rutines en un nivell d'usuari. OpenMP utilitza en C/C++ les directives de preprocessament conegudes com a *pragma* (`#pragma omp <resta del pragma>`) per a diferents construccions. Així, per exemple, `omp parallel` s'utilitza per a crear fils addicionals per a executar el treball indicat en la sentència paral·lela on el procés original és el fil mestre (`id=0`). El conegut programa que imprimeix "Hello, world" utilitzant OpenMP i multifils és*:

***Compileu amb** `gcc -fopenmp -o hello hello.c` **(en algunes distribucions -Debian està instal·lat per defecte- s'ha de tenir instal·lada la biblioteca GCC OpenMP (GOMP)** `apt-get install libgomp1`.

```
#include <stdio.h>
#include <omp.h>

int main(int argc, char* argv[]){
   #pragma omp parallel
   printf("Hello, world.\n");
   return 0;}
```

On executarà un *thread* per a cada *core* disponible en l'arquitectura. Per a especificar *work-sharing constructs*, s'utilitza:

- **omp for o omp do**: reparteix les iteracions d'un llaç en múltiples fils.
- **sections**: assigna blocs de codi independents consecutius a diferents fils.

- **single**: especifica que el bloc de codi serà executat per un sol fil amb una sincronització (*barrier*) al final.
- **master**: similar a `single`, però el codi del bloc serà executat pel fil mestre i no hi ha *barrier* implicat al final.

Per exemple, per a inicialitzar els valors d'un *array* en paral·lel utilitzant fils per a fer una porció del treball (compileu, per exemple, amb `gcc -fopenmp -o init2 init2.c`):

```
#include <stdio.h>
#include <omp.h>
#define N 1000000
int main(int argc, char *argv[]) {
   float a[N]; long i;
    #pragma omp parallel for
   for (i=0;i<N;i++)  a[i]= 2*i;
return 0;
}
```

Si executem amb el *pragma* i després el comentem i calculem el temps d'execució (`time ./init2`), veiem que el temps d'execució passa de 0.003 s a 0.007 s, la qual cosa mostra la utilització del *dualcore* del processador.

Atès que OpenMP és un model de memòria compartida, moltes variables en el codi són visibles per a tots els fils per defecte. Tanmateix, de vegades és necessari tenir variables privades i passar valors entre blocs seqüencials del codi i blocs paral·lels, per la qual cosa és necessari definir atributs a les dades (*data clauses*) que permetin diferents situacions.

- **shared**: les dades a la regió paral·lela són compartides i accessibles per tots els fils de manera simultània.

- **private**: les dades a la regió paral·lela són privades per a cada fil, i cadascun en té una còpia sobre una variable temporal.

- **default**: permet al programador definir com seran les dades dins de la regió paral·lela (`shared`, `private` o `none`).

Un altre aspecte interessant d'OpenMP són les directives de sincronització:

- **critical section**: el codi emmarcat serà executat per fils, però només un per vegada (no hi haurà execució simultània) i es manté l'exclusió mútua.
- **atomic**: similar a `critical section`, però avisa el compilador perquè faci servir instruccions de maquinari especials de sincronització i així obtenir millors prestacions.
- **ordered**: el bloc és executat en l'ordre d'un programa seqüencial.
- **barrier**: cada fil espera que els restants hagin acabat la seva execució (implica sincronització de tots els fils al final del codi).
- **nowait**: especifica que els fils que acabin el treball assignat poden continuar.

A més, OpenMP proveeix de sentències per a la planificació (*scheduling*) del tipus `schedule(type, chunk)` (on el tipus pot ser `static`, `dynamic` o `guided`) o proporciona control sobre les sentències `if`, la qual cosa permetrà definir si es paral·lelitza o no en funció de si l'expressió és veritable o no. OpenMP també proporciona un conjunt de funcions de biblioteca, com per exemple:

- **omp_set_num_threads**: defineix el nombre de fils que cal fer servir a la següent regió paral·lela.
- **omp_get_num_threads**: obté el nombre de fils que s'estan usant en una regió paral·lela.
- **omp_get_max_threads**: obté la màxima quantitat possible de fils.
- **omp_get_thread_num**: retorna el número del fil.
- **omp_get_num_procs**: retorna el màxim nombre de processadors que es poden assignar al programa.
- **omp_in_parallel**: retorna un valor diferent de zero si s'executa dins d'una regió paral·lela.

Veurem a continuació alguns exemples simples (compileu amb la instrucció `gcc -fopenmp -o out_file input_file.c`):

```
/* Programa simple multihreading amb OpenMP */
#include <omp.h>
int main()  {
     int iam =0, np = 1;

     #pragma omp parallel private(iam, np)
     #if defined (_OPENMP)
         np = omp_get_num_threads();
         iam = omp_get_thread_num();
     #endif

     printf("Hello from thread %d out of %d \n", iam, np);
     }
 }

 /* Programa simple amb threads imbricats amb OpenMP */
 #include <omp.h>
 #include <stdio.h>
     main(){
         int x=0,nt,tid,ris;

         omp_set_nested(2);
         ris=omp_get_nested();
         if (ris) printf("Paral·lelisme imbricat actiu %d\n", ris);
         omp_set_num_threads(25);
         #pragma omp parallel private (nt,tid) {
             tid = omp_get_thread_num();
             printf("Thread %d\n",tid);
             nt = omp_get_num_threads();
             if (omp_get_thread_num()==1)
             printf("Nombre de Threads: %d\n",nt);
             }
}

 /* Programa simple d'integració amb OpenMP */
 #include <omp.h>
 #include <stdio.h>
 #define N 100
     main() {
     double local, pi=0.0, w; long i;
     w = 1.0 / N;
```

```
        #pragma omp parallel private(i, local)
            {
            #pragma omp single
            pi = 0.0;
            #pragma omp for reduction(+: pi)
            for (i = 0; i < N; i++) {
              local = (i + 0.5)*w;
              pi = pi + 4.0/(1.0 + local*local);
              printf ("Pi: %f\n",pi);
                }
            }
        }

/* Programa simple de reducció amb OpenMP */
#include <omp.h>
#include <stdio.h>
#define NUM_THREADS 2
    main () {
    int i; double ZZ, res=0.0;
        omp_set_num_threads(NUM_THREADS);
     #pragma omp parallel for reduction(+:res) private(ZZ)
        for (i=0; i< 1000; i++) {
       ZZ = i*i;
       res = res + ZZ;
       printf("ZZ: %f, res: %f\n", ZZ, res);}
              }
        }
```

Nota

Es recomana veure també els exemples que es poden trobar en la referència [6].

1.6. MPI, *Message Passing Interface*

La definició de l'API de MPI [9, 10] ha estat el treball resultant del MPI Forum (MPIF), que és un consorci de més de 40 organitzacions. MPI té influències de diferents arquitectures, llenguatges i treballs al món del paral·lelisme, com per exemple: WRC (Ibm), Intel NX/2, Express, nCUBE, Vertex, p4, Parmac i contribucions de ZipCode, Chimp, PVM, Chamaleon, PICL.

El principal objectiu de MPIF va ser dissenyar una API, sense relació particular amb cap compilador ni biblioteca, que permetés la comunicació eficient (*memory-to-memory copy*), còmput i comunicació concurrent i descàrrega de comunicació, sempre que hi hagués un coprocessador de comunicacions. A més, es demanava que suportés el desenvolupament en ambients heterogenis, amb interfície C i F77 (incloent C++, F90), on la comunicació fos fiable i les fallades, resoltes pel sistema. L'API també havia de tenir interfície per a diferents entorns, disposar d'una implementació adaptable a diferents plataformes amb canvis insignificants i que no interferís amb el sistema operatiu (*thread-safety*). Aquesta API va ser dissenyada especialment per a programadors que utilitzessin el *Message Passing Paradigm* (MPP) en C i F77, per a aprofitar la seva característica més rellevant: la portabilitat. El MPP es pot executar sobre màquines multiprocessador, xarxes d'estacions de treball i fins i tot sobre màquines de memòria compartida. La primera versió de l'estàndard va ser MPI-1 (que si bé hi ha molts desenvolupaments sobre aquesta versió, es considera en EOL), la versió MPI-2 va incorporar un conjunt de millores, com ara creació de processos dinàmics, *one-sided communication,* entrada/sortida paral·lela entre d'altres, i finalment l'última versió, MPI-3 (considerada com una revisió major), inclou *nonblocking collective operations, one-sided operations* i suport per a Fortran 2008.[7]

Molts aspectes han estat dissenyats per a aprofitar els avantatges del maquinari de comunicacions sobre SPC (*scalable parallel computers*), i l'estàndard ha estat acceptat de manera majoritària pels fabricants de maquinari en paral·lel i distribuït (SGI, SUN, Cray, HPConvex, IBM, etc.). Hi ha versions lliures (per exemple, Mpich, LAM/MPI i OpenMPI) que són totalment compatibles amb les implementacions comercials efectuades pels fabricants de maquinari i inclouen comunicacions punt a punt, operacions col·lectives i grups de processos, context de comunicacions i topologia, suport per a F77 i C, i un entorn de control, administració i *profiling*. [11, 12, 10]

Tanmateix, hi ha també alguns punts que poden presentar alguns problemes en determinades arquitectures, com són la memòria compartida, l'execució remota, les eines de construcció de programes, la depuració, el control de fils, l'administració de tasques i les funcions d'entrada/sortida concurrents (la major part d'aquests problemes de falta d'eines estan resolts a partir de la versió 2 de l'API –MPI2). Un dels problemes de MPI1, ja que no té creació dinàmica de processos, és que només suporta models de programació MIMD (*Multiple Instruction Multiple Data*) i comunicant-se via les anomenades MPI. A partir de MPI-2 i amb els avantatges de la creació dinàmica de processos, ja es poden implementar diferents paradigmes de programació com ara *master-worker/farmer-tasks*, *divide & conquer*, paral·lelisme especulatiu, etc. (o almenys sense tanta complexitat i major eficiència en la utilització dels recursos).

Per a la instal·lació de MPI es recomana utilitzar la distribució (en alguns casos la compilació pot ser complexa a causa de les dependències d'altres paquets que pot necessitar). Debian inclou la versió OpenMPI (sobre Debian 7.5 és versió 2, però es poden baixar les fonts i compilar-les) i Mpich2 (Mpich3 disponible en http://www.mpich.org/downloads/). La millor elecció serà OpenMPI, ja que combina les tecnologies i els recursos d'altres projectes diversos (FT-MPI, LA-MPI, LAM/MPI i PACX-MPI) i suporta totalment l'estàndard MPI-2 (i des de la versió 1.75, suporta la versió MPI3). Entre altres característiques d'OpenMPI tenim: és conforme a MPI-2/3, *thread safety & concurrency*, creació dinàmica de processos, alt rendiment i gestió de treballs tolerants a fallades, instrumentació en temps d'execució, *job schedulers*, etc. Per a això s'han d'instal·lar els paquets `openmpi-dev`, `openmpi-bin`, `openmpi-common` i `openmpi-doc`. A més, Debian 7.5 inclou una altra implementació de MPI anomenada LAM (paquets `lam*`). S'ha de considerar que si bé les implementacions són equivalents des del punt de vista de MPI, tenen diferències quant a la gestió i procediments de compilació/execució/gestió.

1.6.1. Configuració d'un conjunt de màquines per a fer un clúster adaptat a OpenMPI

Per a la configuració d'un conjunt de màquines per a fer un clúster adaptat a OpenMPI [14], s'han de seguir els següents passos:

1) Cal tenir les màquines "visibles" (per exemple, mitjançant un `ping`) a través de TCP/IP (IP pública/privada).

2) És recomanable que totes les màquines tinguin la mateixa arquitectura de processador, així és més fàcil distribuir el codi, amb versions similars de Linux (si pot ser, amb el mateix tipus de distribució).

3) Es recomana tenir NIS o, si no, s'ha de generar un mateix usuari (per exemple, mpiuser) a totes les màquines i el mateix directori `$HOME` muntat per NFS.

4) Nosaltres anomenarem les màquines `slave1`, `slave2`, etc. (ja que després resultarà més fàcil fer les configuracions), però es poden anomenar les màquines com cadascú prefereixi.

5) Una de les màquines serà el mestre i les restants, *slaveX*.

6) S'ha d'instal·lar en tots els nodes (suposem que tenim la distribució Debian): `openmpi-bin`, `openmpi-common`, `openmpi-dev`. Cal verificar que en totes les distribucions es treballa amb la mateixa versió d'OpenMPI.

7) En Debian, els executables estan en */usr/bin* però si estan en un *path* diferent, haurà d'agregar-se a mpiuser i també verificar que `LD_LIBRARY_PATH` apunta a */usr/lib*.

8) En cada node esclau ha d'instal·lar-se el *SSH server* (instal·leu el paquet `openssh-server`), i sobre el mestre, el client (paquet `openssh-client`).

9) S'han de crear les claus públiques i privades fent `ssh-keygen -t dsa` i copiar a cada node amb `ssh-copy-id` per a aquest usuari (només s'ha de fer en un node, ja que, com tindrem el directori *$HOME* compartit per NFS per a tots els nodes, amb una còpia n'hi ha prou).

10) Si no es comparteix el directori, cal assegurar que cada esclau coneix que l'usuari *mpiuser* es pot connectar sense *passwd*, per exemple fent: `ssh slave1`.

11) S'ha de configurar la llista de les màquines sobre les quals s'executarà el programa, per exemple */home/mpiuser/.mpi_hostfile* i amb el següent contingut:

```
# The Hostfile for Open MPI
# The master node, slots=2 is used because it is a dual-processor machine.
    localhost slots=2
# The following slave nodes are single processor machines:
    slave1
    slave2
    slave3
```

12) OpenMPI permet utilitzar diferents llenguatges, però aquí utilitzarem C. Per a això, cal executar sobre el mestre `mpicc testprogram.c`. Si es desitja veure què incorpora `mpicc`, es pot fer `mpicc -showme`.

13) Per a executar en local, podríem fer `mpirun -np 2 ./myprogram` i per a executar sobre els nodes remots (per exemple 5 processos), `mpirun -np 2 -hostfile ./mpi_hostfile ./myprogram`.

És important notar que `np` és el nombre de processos o processadors en què s'executarà el programa i es pot posar el nombre que es desitgi, ja que OpenMPI intentarà distribuir els processos de manera equilibrada entre totes les màquines. Si hi ha més processos que processadors, OpenMPI/Mpich utilitzarà les característiques d'intercanvi de tasques de GNU/Linux per a simular l'execució paral·lela. A continuació, es veuran dos exemples: Srtest és un programa simple per a establir comunicacions entre processos punt a punt, i cpi calcula el valor del nombre π de manera distribuïda (per integració).

```
/* Srtest Progam */
#include "mpi.h"
#include <stdio.h>
#include <string.h>
#define BUFLEN 512

int main(int argc, char *argv[]){
    int myid, numprocs, next, namelen;
    char buffer[BUFLEN], processor_name[MPI_MAX_PROCESSOR_NAME];
    MPI_Status status;
    MPI_Init(&argc,&argv);  /* Ha de posar-se abans d'altres crides MPI, sempre */
    MPI_Comm_size(MPI_COMM_WORLD,&numprocs);
    MPI_Comm_rank(MPI_COMM_WORLD,&myid);    /*Integra el procés en un grup de comunicacions*/
    MPI_Get_processor_name(processor_name,&namelen); /*Obté el nom del processador*/

    fprintf(stderr,"Procés %d sobre %s\n", myid, processor_name);
    fprintf(stderr,"Procés %d de %d\n", myid, numprocs);
    strcpy(buffer,"hello there");
    if (myid == numprocs-1) next = 0;
    else next = myid+1;

    if (myid == 0) { /*Si és l'inicial, envia string de buffer*/
        printf("%d sending '%s' \n",myid,buffer);fflush(stdout);
    MPI_Send(buffer, strlen(buffer)+1, MPI_CHAR, next, 99, MPI_COMM_WORLD);
        /*Blocking Send, 1:buffer, 2:size, 3:tipus, 4:destinació, 5:tag, 6:context*/
    printf("%d receiving \n",myid);fflush(stdout);
    MPI_Recv(buffer, BUFLEN, MPI_CHAR, MPI_ANY_SOURCE, 99, MPI_COMM_WORLD,&status);
    printf("%d received '%s' \n",myid,buffer);fflush(stdout);
       /* mpdprintf(001,"%d receiving \n",myid); */
        }
    else {
       printf("%d receiving  \n",myid);fflush(stdout);
   MPI_Recv(buffer, BUFLEN, MPI_CHAR, MPI_ANY_SOURCE, 99, MPI_COMM_WORLD,&status);
        /* Blocking Recv, 1:buffer, 2:size, 3:tipus, 4:font, 5:tag, 6:context, 7:status*/
   printf("%d received '%s' \n",myid,buffer);fflush(stdout);
   /* mpdprintf(001,"%d receiving \n",myid); */
   MPI_Send(buffer, strlen(buffer)+1, MPI_CHAR, next, 99, MPI_COMM_WORLD);
   printf("%d sent '%s' \n",myid,buffer);fflush(stdout);
        }
    MPI_Barrier(MPI_COMM_WORLD); /*Sincronitza tots els processos*/
    MPI_Finalize(); /*Allibera els recursos i acaba*/
    return (0);
    }


/* CPI Program */
#include "mpi.h"
#include <stdio.h>
#include <math.h>
double f( double );
double f( double a) { return (4.0 / (1.0 + a*a)); }
int main( int argc, char *argv[] ) {
    int done = 0, n, myid, numprocs, i;
    double PI25DT = 3.14159265358979323846264338327950288419716939937510;
    double mypi, pi, h, sum, x; double startwtime = 0.0, endwtime;
    int namelen; char processor_name[MPI_MAX_PROCESSOR_NAME];
    MPI_Init(&argc,&argv);
```

```
    MPI_Comm_size(MPI_COMM_WORLD,&numprocs); /*Indica el nombre de processos en el grup*/
    MPI_Comm_rank(MPI_COMM_WORLD,&myid); /*Id del procés*/
    MPI_Get_processor_name(processor_name,&namelen); /*Nom del procés*/
    fprintf(stderr, "Procés %d sobre %s\n", myid, processor_name);
    n = 0;
    while (!done) {
        if (myid ==0) { /*Si és el primer...*/
            if (n ==0) n = 100; else n = 0;
            startwtime = MPI_Wtime();} /* Time Clock */
        MPI_Bcast(&n, 1, MPI_INT, 0, MPI_COMM_WORLD); /*Broadcast a la resta*/
            /*Envia des del 4 arg. a tots els processos del grup. Els restants que no són 0
            copiaran el buffer des de 4 o arg -procés 0-*/
            /*1:buffer, 2:size, 3:tipus, 5:grup */
        if (n == 0) done = 1;
        else {h = 1.0 / (double) n;
            sum = 0.0;
            for (i = myid + 1; i <= n; i += numprocs) {
                x = h * ((double)i - 0.5); sum += f(x); }
            mypi = h * sum;
            MPI_Reduce(&mypi, &pi, 1, MPI_DOUBLE, MPI_SUM, 0,MPI_COMM_WORLD);
                /* Combina els elements del Send Buffer de cada procés del grup usant
                l'operació MPI_SUM i retorna el resultat en el Recv Buffer. Ha de ser cridada
                per tots els processos del grup fent servir els mateixos arguments*/

                /*1:sendbuffer, 2:recvbuffer, 3:size, 4:tipus, 5:oper, 6:root, 7:context*/
            if (myid == 0){ /*només el P0 imprimeix el resultat*/
                printf("Pi és aproximadament %.16f, l'error és %.16f\n", pi,fabs(pi - PI25DT));
                endwtime = MPI_Wtime();
                printf("Temps d'execució = %f\n", endwtime-startwtime); }
            }
    }
    MPI_Finalize(); /*Allibera recursos i acaba*/
    return 0;
}
```

Per a visualitzar l'execució d'un codi paral·lel/distribuït en MPI, hi ha una aplicació anomenada XMPI (en Debian xmpi) que permet "veure", durant l'execució, l'estat de l'execució de les aplicacions sobre MPI, però està vinculada al paquet LAM/MPI. Per a visualitzar i analitzar codi sobre OpenMPI, s'hauria de (i és recomanable) baixar i compilar el codi de MPE* o TAU**, que són eines de *profiling* molt potents i que no requereixen gaire treball ni dedicació per a engegar-les.

***http://www.mcs.anl.gov/research/projects/perfvis/software/mpe/**
****http://www.cs.uoregon.edu/research/tau/home.php**

1.7. Rocks Cluster

Rocks Cluster és una distribució de Linux per a clústers d'ordinadors d'alt rendiment. Les versions actuals de Rocks Cluster estan basades en CentOS (CentOS 6.5 el juliol del 2014) i, com a instal·lador, Anaconda amb certes modificacions, que simplifica la instal·lació "en massa" en molts ordinadors. Rocks Cluster inclou moltes eines (com ara MPI) que no formen part de CentOS però són els components que transformen un grup d'ordinadors en un clúster. Les instal·lacions poden personalitzar-se amb paquets de programari addicionals anomenats *rolls*. Els *rolls* estenen el sistema integrant automàticament els mecanismes de gestió i empaquetament usats pel programari bàsic, i simplifiquen àmpliament la instal·lació i configuració d'un gran nombre d'ordinadors. S'ha creat una gran quantitat de *rolls*, com per exemple SGE *roll*, Cóndor *roll*, Xen *roll*, el Java *roll*, Ganglia *roll*, etc.*** Rocks Cluster és una dis-

*****http://www.rocksclusters.org/wordpress/?page_id=4**

tribució molt emprada en l'àmbit de clústers, per la seva facilitat d'instal·lació i incorporació de nous nodes i per la gran quantitat de programes per al manteniment i monitoratge del clúster.

Enllaços d'interès

Per a una llista detallada de les eines incloses en Rocks Cluster, consulteu http://www.rocksclusters.org/roll-documentation/base/5.5/.

Les principals característiques de Rocks Cluster són:

1) Facilitat d'instal·lació, ja que només és necessari completar la instal·lació d'un node anomenat *node mestre* (*frontend*), la resta s'instal·la amb Avalache, que és un programa P2P que ho fa de manera automàtica i evita haver d'instal·lar els nodes un a un.

2) Disponibilitat de programes (conjunt molt ampli de programes que no és necessari compilar i transportar) i facilitat de manteniment (només s'ha de mantenir el node mestre).

3) Disseny modular i eficient, pensat per a minimitzar el trànsit de xarxa i utilitzar el disc dur propi de cada node per a només compartir la informació mínima i imprescindible.

La instal·lació es pot seguir pas a pas des del lloc web de la distribució [15] i els autors garanteixen que no es triga més d'una hora per a una instal·lació bàsica. Rocks Cluster permet, en l'etapa d'instal·lació, diferents mòduls de programari, els *rolls*, que contenen tot el necessari per a fer la instal·lació i la configuració de sistema amb aquestes noves addicions de manera automàtica i, a més, decidir en el *frontend* com serà la instal·lació en els nodes esclaus, quins *rolls* estaran actius i quina arquitectura s'usarà. Per al manteniment, inclou un sistema de còpia de seguretat de l'estat, anomenat *Roll Restore*. Aquest *roll* guarda els arxius de configuració i *scripts* (i fins i tot, es poden afegir els arxius que es vulguin).

1.7.1. Guia ràpida d'instal·lació

Aquest és un resum breu de la instal·lació proposada en [15] per a la versió 6.1 i es parteix de la base que el *frontend* té (mínim) 30 GB de disc dur, 1 GB de RAM, arquitectura x86-64 i 2 interfícies de xarxa (eth0 per a la comunicació amb Internet i eth1 per a la xarxa interna); per als nodes 30 GB de disc dur, 512 MB de RAM i 1 interfície de xarxa (xarxa interna). Després d'obtenir els discos *Kernel/Boot Roll, Base Roll, US Roll CD1/2* (o en defecte d'això, DVD equivalent), inserim *kernel boot* i seguim els següents passos:

Enllaços d'interès

És possible descarregar els discos des de: http://www.rocksclusters.org/wordpress/?page_id=80

1) Arrenquem el *frontend* i veurem una pantalla en la qual introduïm `build` i ens preguntarà la configuració de la xarxa (IPV4 o IPV6).

2) El següent pas que s'ha de fer és seleccionar els `rolls` (per exemple, seleccionant els "CD/DVD-based Roll" i anem introduint els següents *rolls*) i marcar en les successives pantalles quins són els que volem.

3) La següent pantalla ens demanarà informació sobre el clúster (és important definir bé el nom de la màquina, *Fully-Qualified Host Name*, ja que en cas contrari fallarà la connexió amb diferents serveis) i també informació per a la xarxa privada que connectarà el *frontend* amb els nodes i la xarxa pública (per exemple, la que connectarà el *frontend* amb Internet) així com DNS i passarel·les.

4) A continuació, se sol·licitarà la contrasenya per al *root*, la configuració del servei de temps i la partició del disc (es recomana escollir "auto").

5) Després de formatar els discos, sol·licitarà els CD de *rolls* indicats i instal·larà els paquets i farà un *reboot* del *frontend*.

6) Per a instal·lar els nodes, s'ha d'entrar com a *root* en el *frontend* i executar `insert-ethers`, que capturarà les peticions de DHCP dels nodes i els agregarà a la base de dades del *frontend*, i seleccionar l'opció "*Compute*" (per defecte, consulteu la documentació per a les altres opcions).

7) Posem en marxa el primer node i en el *boot order* de la BIOS generalment es tindrà CD, PXE (Network Boot), Hard Disk (si l'ordinador no suporta PXE, llavors feu el *boot* del node amb el *Kernel Roll CD*). En el *frontend* es veurà la petició i el sistema l'agregarà com a `compute-0-0` i començarà la descàrrega i instal·lació dels arxius. Si la instal·lació falla, s'hauran de reiniciar els serveis httpd, mysqld i autofs en el *frontend*.

8) A partir d'aquest punt, es pot monitorar la instal·lació executant la instrucció `rocks-console`, per exemple amb `rocks-console compute-0-0`. Després que s'hagi instal·lat tot, es pot sortir de `insert-ethers` prement la tecla F8. Si es disposa de dos *racks*, després d'instal·lat el primer es pot començar el segon fent `insert-ethers -cabinet=1`, els quals rebran els noms `compute-1-0`, `compute-1-1`, etc.

9) A partir de la informació generada en consola per `rocks list host`, generarem la informació per a l'arxiu `machines.conf`, que tindrà un aspecte com:

```
nteum slot=2
compute-0-0 slots=2
compute-0-1 slots=2
compute-0-2 slots=2
compute-0-3 slots=2
```

i que després haurem d'executar amb

```
mpirun -np 10 -hostfile ./machines.conf ./mpi_program_to_execute.
```

1.8. FAI

FAI és una eina d'instal·lació automatitzada per a desplegar Linux en un clúster o en un conjunt de màquines en forma desatesa. És equivalent a *Kickstart*

de RH, o Alice de SuSE. FAI pot instal·lar Debian, Ubuntu i RPM de distribucions Linux. Els seus avantatges són que mitjançant aquesta eina es pot desplegar Linux sobre un node (físic o virtual) simplement fent que arrenqui per PXE i quedi preparat per a treballar i totalment configurat (quan es diu un, es podria dir 100 amb el mateix esforç per part de l'administrador) i sense cap interacció pel mig. Per tant, és un mètode escalable per a la instal·lació i actualització d'un clúster Beowulf o una xarxa d'estacions de treball sense supervisió i amb poc esforç. FAI utilitza la distribució Debian, una col·lecció d'*scripts* (la majoria en Perl) i *cfengine* i *Preseeding d-i* per al procés d'instal·lació i canvis en els arxius de configuració.

És una eina pensada per a administradors de sistemes que han d'instal·lar Debian en desenes o centenars d'ordinadors i pot utilitzar-se a més com a eina d'instal·lació de propòsit general per a instal·lar (o actualitzar) un clúster de còmput HPC, de servidors web, o un *pool* de bases de dades i configurar la gestió de la xarxa i els recursos en forma desatesa. Aquesta eina permet tractar (mitjançant un concepte que incorpora anomenat *classes*) amb un maquinari diferent i diferents requisits d'instal·lació en funció de la màquina i característiques de què es disposi en la seva preconfiguració, la qual cosa permet fer desplegaments massius eficients i sense intervenció d'administrador (una vegada que l'eina hagi estat configurada de manera adequada). [29, 30, 31]

1.8.1. Guia ràpida d'instal·lació

En aquest apartat veurem una guia de la instal·lació bàsica sobre màquines virtuals (en Virtualbox) per a desplegar altres màquines virtuals, però es pot adaptar/ampliar a les necessitats de l'entorn amb mínimes intervencions i és una eina que no té *daemons*/DB i només consisteix en *scripts* (consulteu les referències indicades).

Instal·lació del servidor. Si bé hi ha un paquet anomenat `fai-quickstart` és millor fer-ho per parts, ja que molts dels paquets ja estan instal·lats, i si no, es poden instal·lar quan és necessari (vegeu més informació en http://fai-project.org/fai-guide/_anchor_id_inst_xreflabel_inst_installing_fai.html).

Descàrrega dels paquets: en Debian Wheezy hi ha tots els paquets. És millor baixar-se l'última versió de http://fai-project.org/download/wheezy/ i encara que s'han de descarregar tots els paquets `fai-alguna-cosa-.deb` només utilitzarem `fai-server` i `fai-doc` en aquest exemple.

Instal·lació de la clau (*key*):

```
wget -O - http://fai-project.org/download/074BCDE4.asc | sudo
apt-key add -
```

Instal·lació dels paquets: dins del directori on es trobin i com a *root* cal executar

```
dkpg -i fai-server_x.y_all.deb; dkpg -i fai-doc_x.y_all.deb
```

reemplaçant la "x.y"per a la versió descarregada.

Configurar el DHCP del servidor: el *boot* dels nodes i la transferència del sistema operatiu l'efectuarem per una petició PXE i qui primer ha d'actuar és el `dhcp` per a assignar al node els paràmetres de xarxa i l'arxiu de *boot*, per la qual cosa haurem de modificar */etc/dhcp/dhcpd.conf* agregant les primitives **next-server** i **filename** a les zones de la nostra xarxa:

```
...
authoritative;
...
subnet 192.168.168.0 netmask 255.255.255.0 {
  ...
  next-server 192.168.168.254;
  filename "fai/pxelinux.0";
  ...
}
```

Configuració del servei TFTP: la transferència dels arxius del SO es farà pel servei TFTP (*Trivial File Transfer Protocol*), per la qual cosa haurem de comprovar si el `tftp-hpa` està instal·lat i ben configurat amb `netstat -anp|grep :69`. Haurà de donar una cosa similar a `udp 0 0 0.0.0.0:69 0.0.0.0:*` Si no, caldrà verificar si està instal·lat (`dpkg -l | grep tftp`) i, en cas que no estigui instal·lat, executar `apt-get install tftpd-hpa`. A continuació, verificarem que la configuració és l'adequada */etc/default/tftpd-hpa* (s'ha de recordar reiniciar el servei si es modifica):

```
TFTP_USERNAME="tftp"
TFTP_DIRECTORY="/srv/tftp"
TFTP_ADDRESS="0.0.0.0:69"
TFTP_OPTIONS="--secure"
```

Crear la configuració de FAI: executant com a *root* (els arxius de configuració es trobaran en */srv/fai/config* i hi ha regles per a modificar-los –vegeu la documentació):

```
fai-setup
echo '/srv/fai/config 192.168.168.0/24(async,ro,no_subtree_check,no_root_squash)' >> /etc/exports
/etc/init.d/nfs-kernel-server restart

Amb un exportfs, veurem una cosa com:
    /home          192.168.168.0/24
    ...
    /srv/fai/nfsroot 192.168.168.0/24
    /srv/fai/config 192.168.168.0/24
```

Generar una configuració FAI-client: per a tots els clients (després, podrem fer configuracions diferents):

```
fai-chboot -IBv -u nfs://192.168.168.254/srv/fai/config default
```

Haurem de modificar l'arxiu generat perquè treballi amb NFSv3; amb la versió del kernel utilitzat en Wheezy (3.2.x) només podem utilitzar V3 (si es desitja utilitzar NFSv4, haurem de canviar a un kernel 3.12.x que estan disponibles en repositoris com BackPorts http://backports.debian.org/). L'arxiu */srv/tftp/fai/pxelinux.cfg/default* ha de quedar com (s'ha modificat la variable *root*):

```
default fai-generated

label fai-generated
kernel vmlinuz-3.2.0-4-amd64
append initrd=initrd.img-3.2.0-4-amd64 ip=dhcp  root=192.168.168.254:/srv/fai/nfsroot:vers=3
aufs FAI_FLAGS=verbose,sshd,reboot FAI_CONFIG_SRC=nfs://192.168.168.254/srv/fai/config FAI_ACTION=install
```

Copiar els mínims fitxers de configuració:

```
cp -a /usr/share/doc/fai-doc/examples/simple/* /srv/fai/config/
```

Configuració del node: la nostra prova de concepte la farem per a un node en VirtualBox i primer és necessari instal·lar les expansions de la versió que tenim instal·lada. Per exemple, si tenim VirtualBox 4.3.10, haurem d'anar a *https://www.virtualbox.org/wiki/downloads* i instal·lar *VirtualBox 4.3.10 Oracle VM VirtualBox Extension Pack All supported platforms* (això ens permetrà tenir un dispositiu que faci *boot* per a PXE) i podrem verificar que estan instal·lades en el menú de VirtualBox – >*File* – >*Preferences* – >*Extensions*. Després haurem de crear una nova màquina virtual amb un disc buit i seleccionar-la en *System* l'opció de *boot order network* en primer lloc. Arrencarem la màquina i veurem que rep IP (verifiquem els possibles errors en */var/log/messages*|*syslog*) i que comença baixant el **initrd-img** i després el **kernel**.

Els detalls de la configuració es poden consultar en http://fai-project.org/fai-guide/_anchor_id_config_xreflabel_config_installation_details.html

1.9. *Logs*

Linux manté un conjunt de registres anomenats *system logs* o *logs* simplement, que permeten analitzar què ha passat i quan en el sistema, a partir dels esdeveniments que recull del propi *kernel* o de les diferents aplicacions i gairebé totes les distribucions es troben en */var/log*. Probablement, els dos arxius més importants (i que amb més o menys presència estan en totes les distribucions) són */var/log/messages* i */var/log/syslog*, els que tenen registres (ASCII) d'esdeveniments com ara errors del sistema, (re)inicis/apagats, errors d'aplicacions,

advertiments, etc. Existeix una ordre, `dmesg`, que permet a més veure els missatges d'inici (en algunes distribucions són els que apareixen en la consola o amb la tecla Esc en la manera gràfica d'arrencada, o Crtl+F8 en altres distribucions) per a visualitzar els passos seguits durant l'arrencada (tenint en compte que pot ser molt extens, es recomana executar `dmesg | more`).

Per la qual cosa, el sistema ens permetrà analitzar què ha passat i esbrinar les causes que han produït aquest registre i on/quan. El sistema inclòs en Debian és `rsyslog` (http://www.rsyslog.com/) amb un servei (a través del *daemon* **rsyslogd**) que reemplaça a l'original *syslog*. És un sistema molt potent i versàtil i la seva configuració és través de l'arxiu */etc/rsyslog.conf* o arxius en el directori */etc/rsyslog.d*. Podeu mirar la documentació sobre la seva configuració (`man rsyslog.conf` o a la pàgina indicada), però en resum inclou una sèrie de directives, filtres, *templates* i regles que permeten canviar el comportament dels *logs* del sistema. Per exemple, les regles són de tipus **recurs.nivell acció** però es pot combinar més d'un recurs separat per ',', o diferents nivells, o tots '*' o negar-ho '!', i si posem '=' davant del nivell, indica només aquest nivell i no tots els superiors, que és el que ocorre quan només s'indica un nivell (els nivells poden ser de menor a major importància *debug, info, notice, warning, err, crit, alert, emerg*). Per exemple, `*.warning /var/log/messages` indicarà que tots els missatges de *warning,err,crit,alert,emerg* de qualsevol recurs vagin a parar a aquest arxiu. Es pot incloure també un '-' davant del nom del fitxer, que indica que no se sincronitzi el fitxer després de cada escriptura per a millorar les prestacions del sistema i reduir la càrrega.

Un aspecte interessant dels *logs* és que els podem centralitzar des de les diferents màquines de la nostra infraestructura en un determinat servidor, per a no haver de connectar-nos si volem "veure"què està ocorrent en els *logs* en aquestes màquines (sobretot si el nombre de màquines és elevat). Per a això, en cada client hem de modificar l'arxiu */etc/rsyslog.conf* (on la IP serà la del servidor que recollirà els *logs*):

```
# Provides TCP forwarding.
*.*                             @@192.168.168.254:514
```

En el servidor, haurem de treure el comentari (#) dels mòduls de recepció per TCP en */etc/rsyslog.conf*:

```
# Provides TCP syslog reception
$ModLoad imtcp
$InputTCPServerRun 514
```

Després de fer les modificacions de cada arxiu, és important no oblidar fer un `/etc/init.d/rsyslog restart`. A partir d'aquest moment, podem mirar el */var/log/syslog* del servidor, per exemple, i veurem els registres marcats amb el nom (o IP) de la màquina on s'ha generat el *log*.

Existeix un paquet anomenat `syslog-ng` que presenta característiques avançades (per ex., xifratge o filtres basats en continguts) o altres d'equivalents amb `rsyslog`. Trobareu més informació en http://www.balabit.com/network-security/syslog-ng.

1.9.1. Octopussy

Octopussy és una eina gràfica que permet analitzar els *logs* de diferents màquines i s'integra amb *syslog* o equivalents reemplaçant-los. Entre les seves principals característiques, suporta LDAP, alertes per correu, missatges IM (Jabber), s'integra amb Nagios i Zabbix, permet generar reports i enviar-los per correu, FTP i scp, fer mapes de l'arquitectura per a mostrar estats i incorpora una gran quantitat de serveis predefinits per a registrar els *logs*.[32]

La seva instal·lació (que pot resultar una mica complicada) comença per afegir el repositori *non-free /etc/apt/sources.list* per a descarregar paquets addicionals (libmail-sender-perl) que necessitarà aquest programari:

```
deb http://ftp.es.debian.org/debian/ wheezy non-free
deb-src http://ftp.es.debian.org/debian/ wheezy non-free
```

Després, haurem d'executar `apt-get update`.

Posteriorment, haurem de descarregar el paquet Debian des de l'adreça web http://sourceforge.net/projects/syslog-analyzer/files/ (per exemple el paquet octopussy_x.i.x_all.deb on la x.y.z és la versió –10.0.14 el juliol de 2014) i instal·lar-lo.

```
dpkg -i octopussy_1.0.x_all.deb
apt-get -f install
```

Després haurem de modificar l'arxiu *etc/rsyslog.conf*:

```
$ModLoad imuxsock # provides support for local system logging
$ModLoad imklog   # provides kernel logging support (previously done by rklogd)
#$ModLoad immark  # provides --MARK-- message capability
# provides UDP syslog reception
$ModLoad imudp
$UDPServerRun 514
# provides TCP syslog reception
$ModLoad imtcp
$InputTCPServerRun 514
```

I reiniciar el rsyslog `/etc/init.d/rsyslog restart`.

Després s'hauran de generar els certificats per a *Octopussy Web Server* (també es pot utilitzar l'OpenCA per a generar els certificats web propis, tal com es va explicar en l'apartat de servidors):

```
openssl genrsa > /etc/octopussy/server.key
openssl req -new -x509 -nodes -sha1 -days 365 -key /etc/octopussy/server.key > /etc/octopussy/server.crt
```

En l'últim, no s'ha d'introduir *passwd* i cal recordar en *Common Name* incloure el nom.domini de la nostra màquina.

Configurar l'arxiu de la configuració d'Apache */etc/octopussy/apache2.conf* pel valor correcte:

```
SSLCertificateFile    /etc/octopussy/server.crt
SSLCertificateKeyFile /etc/octopussy/server.key
```

Reiniciar el servidor (propi) de web que després es connectarà amb el nostre Apache per SSL i port 8888.

```
/etc/init.d/octopussy web-stop
/etc/init.d/octopussy web-start
```

I comprobar que funciona en l'URL https://sysdw.nteum.org:8888/index.asp, acceptar el certificat i introduir com a Usuari/Passwd admin/admin. A partir de la interfície general, s'haurà de configurar els *Devices, Service, Alerts*, etc. per a definir el comportament que volem que tingui l'aplicació.

Si volem deshabilitar i tornar al `rsyslog` original (i donat que té serveis propis que s'engeguen durant l'arrencada), haurem d'executar `update-rc.d octopussy remove` i renombrar els arxius de configuració en */etc/rsyslog.d* de l'aplicació (per exemple,

```
mv /etc/rsyslog.d/xyz-ocotopussy.conf /etc/rsyslog.d/xyz-octopussy.conf.org
```

reemplaçant xyz pel que correspongui) i reiniciant *rsyslog* amb `/etc/init.d/rsylog restart`

1.9.2. Eines de monitoratge addicionals

A més de les eines que s'han vist en l'apartat de monitoratge (com Ganglia, Nagios|Icinga, MRTG o Zabbix), hi ha altres eines que poden ajudar en la gestió i administració d'un clúster com són Cacti, XyMon i Collectd (també disponibles en Debian). En aquest apartat, dedicarem una petita referència a Cacti i XYMon, ja que per la seva visió o capacitat d'integració permeten tenir informació a l'instant sobre com estan ocorrent les execucions en el clúster.

Cacti. Cacti [16] és una solució per a la visualització d'estadístiques de xarxa i va ser dissenyada per a aprofitar el poder d'emmagatzematge i la funcionalitat de generar gràfiques que posseeix RRDtool (similar a Ganglia). Aquesta eina,

desenvolupada en PHP, proveeix de diferents formes de visualització, gràfics avançats i disposa d'una interfície d'usuari fàcil d'usar, que la fan interessant tant per a xarxes LAN com per a xarxes complexes amb centenars de dispositius.

La seva instal·lació és simple i requereix prèviament tenir instal·lat MySQL. Després, fent `apt-get install cacti` s'instal·larà el paquet i al final ens demanarà si volem instal·lar la interfície amb la base de dades, i ens sol·licitarà el nom d'usuari i contrasenya d'aquesta i la contrasenya de l'usuari admin de la interfície de Cacti (si no li hem donat, contindrà la que defineix per defecte, que és usuari admin i *passwd* admin). A continuació instal·larà la configuració en el lloc d'Apache2 (*/etc/apache2/conf.d/cacti.conf*) i tornarà a arrencar els servidors i podrem connectar-nos a l'URL *http://sysdw.nteum.org/cacti/*. La primera tasca que cal portar a terme és fer els ajustos pertinents de la instal·lació (ens mostrarà una *checklist*), després s'ha de canviar la contrasenya i ens mostrarà a continuació una pantalla amb les pestanyes de gràfics i la de consola, en la qual trobarem les diferents opcions per a configurar i engegar el monitoratge més adequat per al nostre lloc. Seguint el procediment indicat en *http://docs.cacti.net/manual:088:2_basics.1_first_graph*, serà molt fàcil incorporar els gràfics més adequats per a monitorar la nostra instal·lació. És interessant incorporar a Cacti un *plugin* anomenat *weathermap* (http://www.network-weathermap.com/) que ens permetrà veure un diagrama de la infraestructura en temps real i els elements monitorats amb dades dinàmiques sobre el seu estat*.

***http://www.network-weathermap.com/manual/0.97b/pages/cacti-plugin.html**

Xymon. Xymon [17] és un monitor de xarxa/serveis (sota llicència GPL) que s'executa sobre màquines GNU/Linux. L'aplicació es va inspirar en la versió *opensource* de l'eina Big Brother i es va dir Hobbit però, com que aquesta era una marca registrada, finalment va canviar a Xymon. Xymon ofereix un monitoratge gràfic de les màquines i serveis amb una visualització adequada i jeràrquica òptima per a quan es necessita tenir en una sola visualització l'estat de la infraestructura (i sobretot quan s'han de monitorar centenars de nodes). També es pot entrar en els detalls i mirar qüestions específiques dels nodes, gràfics i estadístiques, però entrant en nivells interiors de detalls. El monitoratge dels nodes requereix un client que serà l'encarregat d'enviar la informació al *host* (equivalent a NRPE en Nagios). La seva instal·lació en Debian és simple: `apt-get install xymon`. Després, podrem modificar l'arxiu */etc/hobbit/bb-host* per a agregar una línia com a *group server* `158.109.65.67 sysdw.nteum.org ssh http://sysdw.nteum.org` i modificar el fitxer */etc/apache2/conf.d/hobbit* per a canviar la línia `Allow from localhost` per `Allow from all`, reiniciar els dos serveis, `service hobbit restart`, i `service apache2 restart` i ho tindrem disponible en l'URL: *http://sysdw.nteum.org/hobbit/* (no us descuideu la barra final). És molt fàcil agregar nous *hosts* (es declaren en l'arxiu anterior), i instal·lar el client en els nodes tampoc té cap dificultat.

2. *Cloud*

Les infraestructures *cloud* es poden classificar en 3 + 1 grans grups en funció dels serveis que presten i a qui els presten:

1) Públiques: els serveis es troben en servidors externs i les aplicacions dels clients es barregen en els servidors i amb altres infraestructures. L'avantatge més clar és la capacitat de processament i emmagatzematge sense instal·lar màquines localment (no hi ha inversió inicial ni manteniment) i es paga per ús. Té un retorn de la inversió ràpid i pot resultar difícil integrar aquests serveis amb altres de propis.

2) Privades: les plataformes es troben dins de les instal·lacions de l'empresa/institució i són una bona opció per a aquells que necessiten una alta protecció de dades (aquests continuen dins de la pròpia empresa). És més fàcil integrar aquests serveis amb altres de propis, però hi ha inversió inicial en infraestructura física, sistemes de virtualització, amplada de banda, seguretat i despesa de manteniment, la qual cosa implica un retorn més lent de la inversió. No obstant això, entre tenir infraestructura i tenir-ne *cloud* i virtualizada, aquesta última és millor pels avantatges de major eficiència, millor gestió i control i major aïllament entre projectes i rapidesa en la provisió.

3) Híbrides: combinen els models de núvols públics i privats i permeten mantenir el control de les aplicacions principals a la vegada que s'aprofita el *cloud computing* en els llocs on tingui sentit amb una inversió inicial moderada i, alhora, permet comptar amb els serveis que es necessitin sota demanda.

4) Comunitat: canalitzen necessitats agrupades i les sinergies d'un conjunt o sector d'empreses per a oferir serveis de *cloud* a aquests grups d'usuaris.

A més, hi ha diferents capes sota les quals es poden contractar aquests serveis:

1) *Infraestructure as a Service* (IaaS): es contracta capacitat de procés i d'emmagatzematge que permeten desplegar aplicacions pròpies que per motius d'inversió, infraestructura, cost o falta de coneixements no es volen instal·lar en la pròpia empresa (exemples d'aquest tipus són EC2/S3 d'Amazon i Azure de Microsoft).

2) *Platform as a Service* (PaaS): es proporciona a més un servidor d'aplicacions (on s'executaran les nostres aplicacions) i una base de dades, on es podran instal·lar les aplicacions i executar-les, les quals s'hauran de desenvolupar d'acord amb unes indicacions del proveïdor (per exemple, Google App Engine).

3) *Programari as a Service* (SaaS): comunament s'identifica amb *cloud*, on l'usuari final paga un lloguer per a l'ús de programari sense adquirir-lo en propietat, instal·lar-lo, configurar-lo i mantenir-lo (en són exemples Adobe Creative Cloud, Google Docs o Office365).

4) *Business Process as a Service* (BPaaS): és la capa més nova (i se sustenta damunt de les altres 3), on el model vertical (o horitzontal) d'un procés de negoci es pot oferir sobre una infraestructura *cloud*.

Si bé els avantatges són evidents i molts actors d'aquesta tecnologia la basen principalment en l'abaratiment dels costos de servei, comencen a haver-hi opinions en contra (http://deepvalue.net/ec2-is-380-more-expensive-than-internal-cluster/) que consideren que un *cloud* públic potser no és l'opció més adequada per a determinat tipus de serveis/infraestructura. Entre altres aspectes, els negatius poden ser la fiabilitat en la prestació de servei, seguretat i privadesa de les dades/informació en els servidors externs, *lock-in* de dades en la infraestructura sense possibilitat d'extreure-les, estabilitat del proveïdor (com a empresa/negoci) i posició de força al mercat no subjecte a la variabilitat del mercat, colls d'ampolla en les transferències (empresa/proveïdor), rendiment no predictible, temps de resposta a incidents/accidents, problemes derivats de la falta de maduresa de la tecnologia/infraestructura/gestió, acords de serveis (SLA) pensats més per al proveïdor que per a l'usuari, etc. No obstant això, és una tecnologia que té els seus avantatges i que amb l'adequada planificació i presa de decisions, valorant tots els factors que influeixen en el negoci i escapant de conceptes superficials (tots el tenen, tots l'utilitzen, baix cost, expansió il·limitada, etc.), pot ser una elecció adequada per als objectius de l'empresa, el seu negoci i la prestació de serveis en IT que necessita o que forma part de les seves finalitats empresarials.[18]

És molt àmplia la llista de proveïdors de serveis *cloud* en les diferents capes, però d'acord amb la informació de Synergy Research Group el 2014*, els líders al mercat d'infraestructures *cloud* són:

***https://www.srgresearch.com/articles/amazon-salesforce-and-ibm-lead-cloud-infrastructure-service-segments**

1) *IasS*: Amazon amb gairebé el 50% i IBM i Rackspace amb valors inferiors al 10% cadascun i Google en valors inferiors.

2) *PaaS*: Salesforce (20%) seguides molt de prop per Amazon, Google i Microsoft.

3) *Private/Hibrid*: IBM (15%), Orange i Fujitsu amb valors propers al 5% cadascun.

4) *Business Process as a Service* (BPaaS): Hi ha diversos operadors, per exemple IBM, Citrix i VMware entre d'altres, però no hi ha dades definides de quota de mercat.

Això significa un canvi apreciable en relació amb les dades del 2012, en què es pot observar una variabilitat de l'oferta molt alta**.

****https://www.srgresearch.com/articles/amazons-cloud-iaas-and-paas-investments-pay**

Quant a plataformes per a desplegar *clouds* amb llicències GPL, Apache i BSD (o similars), podem comptar entre les més referenciades (i per ordre alfabètic):

1) AppScale: és una plataforma que permet als usuaris desenvolupar i executar/emmagatzemar les pròpies aplicacions basades en Google AppEngine i pot funcionar com a servei o en local. Es pot executar sobre AWS EC2, Rackspace, Google Compute Engine, Eucalyptus, Openstack, CloudStack, així com sobre KVM i VirtualBox i suporta Python, Java, Go i plataformes PHP Google AppEngine. http://www.appscale.com/

2) Cloud Foundry: és una plataforma *open source* PaaS desenvolupada per VMware escrita bàsicament en Ruby and Go. Pot funcionar com a servei i també en local. http://cloudfoundry.org/

3) Apache CloudStack: dissenyada per a gestionar grans xarxes de màquines virtuals com IaaS. Aquesta plataforma inclou totes les característiques necessàries per al desplegament d'un IaaS: CO (*compute orchestration*), *Network-as-a-Service*, gestió d'usuaris/comptes, API nativa, *resource accounting* i UI millorada (*User Interface*). Suporta els *hypervisors* més comuns, gestió del *cloud* via web o CLI i una API compatible amb AWS EC2/S3 que permeten desenvolupar *clouds* híbrids. http://cloudstack.apache.org/

4) Eucalyptus: plataforma que permet construir *clouds* privats compatibles amb AWS. Aquest programari permet aprofitar els recursos de còmput, xarxa i emmagatzematge per a oferir autoservei i desplegament de recursos de *cloud* privat. Es caracteritza per la simplicitat en la instal·lació i estabilitat en l'entorn amb una alta eficiència en l'ús dels recursos. https://www.eucalyptus.com/

5) Nimbus: és una plataforma que una vegada instal·lada sobre un clúster, proporciona IasS per a la construcció de *clouds* privats o de comunitat i pot ser configurada per a suportar diferents virtualitzacions, sistemes de cues o Amazon EC2. http://www.nimbusproject.org/

6) OpenNebula: és una plataforma per a gestionar tots els recursos d'un centre de dades i permetre la construcció de IasS privats, públics i híbrids. Proveeix d'una gran quantitat de serveis, prestacions i adaptacions que han permès que sigui una de les plataformes més difoses en l'actualitat. http://opennebula.org/

7) OpenQRM: és una plataforma per al desplegament de *clouds* sobre un centre de dades heterogènies. Permet la construcció de *clouds* privats, públics i híbrids amb IaaS. Combina la gestió del la CPU/emmagatzematge/xarxa per a oferir serveis sobre màquines virtualitzades i permet la integració amb recursos remots o altres *clouds*. http://www.openqrm-enterprise.com/community.html

8) OpenShift: aposta important de la companyia RH i és una plataforma (versió Origin) que permet prestar servei *cloud* en modalitat PasS. OpenShift suporta l'execució de binaris que són aplicacions web tal com s'executen en RHEL, per la qual cosa permet un gran nombre de llenguatges i *frameworks*. https://www.openshift.com/products/origin

9) OpenStack és una arquitectura programari que permet el desplegament de *cloud* en la modalitat de IaaS. Es gestiona mitjançant una consola de control via web que permet la provisió i control de tots els subsistemes i l'aprovisionament dels recursos. El projecte iniciat (2010) per Rackspace i NASA és actualment gestionat per OpenStack Foundation i hi ha més de 200 companyies adherides al projecte, entre les quals es troben les grans proveïdores de serveis *cloud* públics i desenvolupadores de SW/HW (ATT, AMD, Canonical, Cisco, Dell, EMC, Ericsson, HP, IBM, Intel, NEC, Oracle, RH, SUSE Linux, VMware, Yahoo, entre d'altres). És una altra de les plataformes més referenciades. http://www.openstack.org/

10) PetiteCloud: és una plataforma programari que permet el desplegament de *clouds* privats (petits) i no orientats a dades. Aquesta plataforma pot ser utilitzada sola o en unió amb altres plataformes *cloud* i es caracteritza per la seva estabilitat/fiabilitat i facilitat d'instal·lació. http://www.petitecloud.org

11) oVirt: si bé no es pot considerar una plataforma *cloud*, oVirt és una aplicació de gestió d'entorns virtualitzats. Això significa que es pot utilitzar per a gestionar els nodes HW, l'emmagatzematge o la xarxa i desplegar i monitorar les màquines virtuals que s'estan executant al centre de dades. Forma part de RH Enterprise Virtualization i és desenvolupada per aquesta companyia amb llicència Apache. http://www.ovirt.org/

2.1. Opennebula

Considerant les opinions de [19],[20], la nostra prova de concepte la farem sobre OpenNebula. Atès que la versió en Debian Wheezy és la 3.4 i el web dels desenvolupadors és el 4.6.2 (http://opennebula.org/), hem decidit instal·lar els paquets Debian amb KVM com a *hypervisor* de la nova versió*. Aquesta instal·lació és una prova de concepte (funcional però mínima), però útil per a després fer un desplegament sobre una arquitectura real, on d'una banda executarem els serveis d'OpenNebula i la seva interfície gràfica (anomenada Sunstone) i, d'una altra banda, un hypervisor (*host*) que executarà les màquines virtuals. OpenNebula assumeix dos rols separats: *frontend* i *nodes*. El ***frontend*** és el que executa els serveis/gestió web i els ***nodes*** executen la màquina virtual, i si bé en la nostra instal·lació de proves executarem el *frontend* i *nodes* a la mateixa màquina, es recomana executar les màquines virtuals en altres *hosts* que tinguin les extensions de virtualització (en el nostre cas, KVM). Per a verificar si disposem d'aquestes extensions en HW podem executar `grep -E 'svm|vmx' /proc/cpuinfo` i si ens dóna sortida és gairebé segur que el sistema suporta aquestes extensions. Tingueu en compte que això ho haurem de fer sobre un sistema operatiu base (*bare metal*) i no sobre un de ja virtualitzat, és a dir, no podem instal·lar OpenNebula sobre una màquina virtualizada per VirtualBox, per exemple (hi ha *hypervisors* que permeten aquesta instal·lació com VMWare ESXi però VirtualBox, no). Els paquets que conformen la instal·lació són:

***http://docs.opennebula.org /4.6/design_and_installation /quick_starts /qs_ubuntu_kvm.html**

1) opennebula-common: arxius comuns.

2) libopennebula-ruby: biblioteques de ruby.

3) opennebula-node: paquet que prepara un node on estarà la VM.

4) opennebula-sunstone: OpenNebula Sunstone *Web Interface*.

5) opennebula-tools: *Command line interface*.

6) opennebula-gate: permet la comunicació entre VMs i OpenNebula.

7) opennebula-flow: gestiona els serveis i l'"elasticitat".

8) opennebula: OpenNebula *daemon*.

Instal·lació del *frontend* (com a *root*): Instal·lar el repositori:

```
wget -q -O- http://downloads.opennebula.org/repo/Debian/repo.key
| apt-key add -
```

i després els agreguem a la llista

```
echo "deb http://downloads.opennebula.org/repo/Debian/7 stable
opennebula"> /etc/apt/sources.list.d/opennebula.list
```

Instal·lar els paquets: fem `apt-get update` i després `apt-get install opennebula opennebula-sunstone` (si el node està en un *host* separat, hauran de compartir un directori per NFS, per la qual cosa és necessari també instal·lar el `nfs-kernel-server`).

Serveis: haurem de tenir dos serveis en marxa que són OpenNebula *daemon* (oned) i la interfície gràfica (sunstone), la qual només està configurada per seguretat per a atendre el *localhost* (si es desitja canviar, cal editar */etc/one/sunstone-server.conf* i canviar **:host: 127.0.0.1** per **:host: 0.0.0.0** i reiniciar el servidor `/etc/init.d/opennebula-sunstone restart`).

Configurar el NFS (si estem en un únic servidor amb *frontend+Node*, aquesta part no és necessària): s'ha d'agregar a */etc/exports* del *frontend* `/var/lib/one/ *(rw,sync,no_subtree_check,root_esquash)` i reiniciar el servei (amb l'ordre `service nfs-kernel-server restart`).

Configurar la clau pública de SSH: OpenNebula necessita accedir per SSH als nodes com l'usuari **oneadmin** i sense *passwd* (des de cada node a un altre, inclòs el *frontend*), per la qual cosa ens canviem com a usuari oneadmin (`su - oneadmin`) i executem

```
cp ~/.ssh/id_rsa.pub ~/.ssh/authorized_keys
```

i agreguem el següent text a *~/.ssh/config* (perquè no demani confirmació de `known_hosts`):

```
cat << EOT > ~/.ssh/config
Host *
    StrictHostKeyChecking no
    UserKnownHostsFile /dev/null
EOT
chmod 600 ~/.ssh/config
```

Instal·lació dels nodes: repetim el pas de configurar el repositori i executem `apt-get install opennebula-node nfs-common bridge-utils` (en el nostre cas no és necessari, ja que és la mateixa màquina i només hem d'instal·lar `opennebula-node` i `bridge-utils`). Hem de verificar que podem accedir com a oneadmin a cada node i copiem les claus de `ssh`.

Configuració de la xarxa: (hem de fer un *backup* dels arxius que modificarem prèviament) generalment tindrem eth0 i la connectarem a un *bridge* (el nom del *bridge* haurà de ser el mateix en tots els nodes) i en */etc/network/interfícies* agreguem (cal posar les IP que corresponguin):

```
auto lo
iface lo inet loopback
auto br0
iface br0 inet static
        address 192.168.0.10
        network 192.168.0.0
        netmask 255.255.255.0
        broadcast 192.168.0.255
        gateway 192.168.0.1
        bridge_ports eth0
        bridge_fd 9
        bridge_hello 2
        bridge_maxage 12
        bridge_stp off
```

Si tenim servei de DHCP, l'hem de reemplaçar per:

```
auto lo
iface lo inet loopback
auto br0
iface br0 inet dhcp
        bridge_ports eth0
        bridge_fd 9
        bridge_hello 2
        bridge_maxage 12
        bridge_stp off
```

I reiniciem la xarxa `/etc/init.d/networking restart`

Configurem el NFS sobre els nodes (no és necessari en el nostre cas): agreguem a */etc/fstab*

```
192.168.1.1:/var/lib/one/ /var/lib/one/ nfs soft,intr,rsize=8192,wsize=8192,noauto
```

on 192.168.1.1 és la IP del *frontend*; després muntem el directori `mount /var/lib/one/`.

Configurem Qemu: l'usuari **oneadmin** ha de poder manejar libvirt com a *root*, per la qual cosa executem el següent.

```
cat << EOT > /etc/libvirt/qemu.conf
user  = "oneadmin"
group = "oneadmin"
dynamic_ownership = 0
EOT
```

Reiniciem libvirt amb `service libvirt-bin restart`

Ús bàsic: En el *frontend*, podem connectar-nos a la interfície web en l'URL *http://frontend:9869*. L'usuari és **oneadmin** i el *passwd* està en el seu *HOME* a */.one/one_auth*, que es genera de manera aleatòria (*/var/lib/one/.one/one_auth*). Per a interactuar amb OpenNebula, s'ha de fer des de l'usuari **oneadmin** i des del *frontend* (per a connectar-se, cal fer simplement `su - oneadmin`).

Agregar un *host*: és el primer que cal fer per després executar VM; es pot fer des de la interfície gràfica o des de CLI fent com oneadmin `onehost create localhost -i kvm -v kvm -n dummy` (cal posar el nom del *host* correcte en lloc de *localhost*).

Si hi ha errors, probablement són de `ssh` (hem de verificar que igual que en el cas de oneadmin, es pot connectar als altres *hosts* sense *passwd*) i els errors els veurem en */var/log/one/oned.log*.

El segon pas és agregar la xarxa, una imatge i un *template* abans de llançar una VM:

Xarxa: (es pot fer des de la interfície gràfica també), creem l'arxiu *mynetwork.one* amb el següent contingut:

```
NAME = "private"
TYPE = FIXED
BRIDGE = br0
LEASES = [ IP=192.168.0.100 ]
LEASES = [ IP=192.168.0.101 ]
LEASES = [ IP=192.168.0.102 ]
```

On les IP hauran d'estar lliures a la xarxa en què estem fent el desplegament i executem `onevnet create mynetwork.one`.

En el cas de la imatge, aquesta es pot fer des de CLI però és més simple fer-la des de la interfície web. Seleccionem **Marketplace** i després **ttylinux-kvm** (és una imatge petita de prova) i després li diem **Import**. Podrem verificar que la imatge està en l'apartat corresponent, però l'estat és LOCK fins que l'hagi baixat i, quan finalitzi, veurem que canvia a READY. Després podrem crear un *template* seleccionant aquesta imatge, la xarxa, i allà podrem posar la

clau pública del *~/.ssh/id_dsa.pub* en l'apartat **Context**, i des d'aquest apartat (**Templates**) podrem dir-li que creï una instància d'aquest *template* (o si no, en **VirtualMachines** crear-ne una utilitzant aquest *template*). Veurem que la VM passa de PENDING – > PROLOG – > RUNNING (si falla en podrem veure en els *logs* la causa) i després ens podrem connectar o bé per `ssh` a la IP de la VM o mitjançant **VNC** en la interfície web.

Com s'ha pogut comprovar, la instal·lació està prou automatitzada però s'ha de tenir en compte que és una infraestructura complexa i que s'ha de dedicar temps i anàlisi a determinar les causes per les quals es produeixen els errors i solucionar-los. Existeix una gran quantitat de documentació i llocs a Internet, però recomanem començar per les fonts [21] i [22].

En aquest apartat, s'ha vist una prova de concepte funcional però OpenNebula és molt extens i flexible, i permet gran quantitat d'opcions/extensions. Mitjançant OpenNebula C12G Labs (http://c12g.com/), es podran trobar i descarregar imatges de màquines virtuals creades per OpenNebula* i llistes per a posar-les en funcionament (són les que es veuen des del MarketPlace de la interfície web), però en descarregar-les sobre el nostre servidor les podrem utilitzar sempre que les necessitem i no s'hauran de descarregar cada vegada (no s'han de descomprimir ni s'ha de fer res, s'han d'utilitzar tal com estan). Una de les extensions interessants és OpenNebula Zones (anomenada **ozones**), que ens permet una administració centralitzada de múltiples instàncies d'OpenNebula (zones), gestionant diferents dominis administratius. El mòdul és gestionat per l'*oZones administrator*, que és qui administra els permisos a les diferents zones dels usuaris individuals**. La seva configuració pot trobar-se en el web d'OpenNebula***.

***http://marketplace.c12g.com/appliance**

****http://archives.opennebula.org/documentation:arxivis:rel3.0:ozones**
*****http://docs.opennebula.org/4.4/advanced_administration/multiple_zone_and_virtual_data_centers/zonesmngt.html**

3. DevOps

Tal com hem comentat a l'inici d'aquest apartat, Devops es pot considerar, en paraules dels experts, un moviment tant en l'aspecte professional com en l'aspecte cultural del món TI. Si bé no hi ha totes les respostes encara, un administrador trobarà diferents "comunitats" (i ell mateix formarà part d'alguna), que tindran noves necessitats de desenvolupament i producció de serveis/productes dins del món TI i amb les premisses de "més ràpid", "més eficient", "de major qualitat" i totalment adaptable als "diferents entorns". És a dir, el grup de treball haurà de trencar les barreres entre els departaments d'una empresa i permetre que un producte passi ràpidament des del departament de recerca al de disseny, després al de desenvolupament + producció, després al de test + qualitat i finalment, a vendes, i amb les necessitats d'eines que permetin desplegar totes aquestes activitats en cadascuna de les fases i portar el control de tots pels responsables de cadascun dels àmbits, inclosos els funcionals i els de l'organització / directius. És per això que un administrador necessitarà eines de gestió, control, desplegament, automatització i configuració. Una llista (curta) de les eines que podríem considerar d'acord amb els nostres objectius de llicència GPL-BSD-Apache o similars (és interessant el lloc http://devs.info/, ja que permet accedir a la major part d'eines/entorns/documentació per a programadors i una llista més detallada -que inclou SW propietari- es pot trobar a [27]):

1) **Linux**: Ubuntu|Debian^{v}, Fedora^{v}|CentOS|SL

2) **IasS**: Cloud Foundry, OpenNebula^{v}, OpenStack

3) **Virtualització**: KVM^{v}, Xen, VirtualBox^{v}, Vagrant^{v}

4) Contenidors: LXC^{v}, Docker^{v}

5) Instal·lació SO: Kickstart i Cobbler (rh), Preseed|Fai^{v} (deb), Rocks^{v}

6) Gestió de la configuració: Puppet^{v}, Chef^{v}, CFEngine, SaltStack, Juju, bcfg2, mcollective, fpm (effing)

7) Servidors web i acceleradors: Apache^{v}, nginx^{v}, varnish, squid^{v}

8) BD: MySQL^{v}|MariaDB^{v}, PostgreSQL^{v}, OpenLDAP^{v}, MongoDB, Redis

9) Entorns: Lamp, Lamr, AppServ, Xampp, Mamp

10) Gestió de versions: Git^{v}, Subversion^{v}, Mercurial^{v}

11) Monitoratge/supervisió: Nagios^{v}, Icinga, Ganglia^{v}, Cacti^{v}, Monin^{v}, MRTG^{v}, XYmon^{v}

12) Misc: pdsh, ppsh, pssh, GNUparallel, nfsroot, Multihost SSH Wrapper, lldpd, Benchmarks^{v}, Biblioteques^{v}

13) Supervisió: Monit^{v}, runit, Supervisor, Godrb, BluePill-rb, Upstart, Systemd^{v}

14) Security: OpenVas^{v}, Tripwire^{v}, Snort^{v}

15) Desenvolupament i test: Jenkins, Maven, Ant, Gradle, CruiseControl, Hudson

16) Desplegament i *workflow*: Capistrano

17) Servidors d'aplicacions: JBoss, Tomcat, Jetty, Glassfish,

18) Gestió de *logs*: Rsyslog^{v}, Octopussy^{v}, Logstash

Excepte les que són molt orientades a desenvolupament d'aplicacions i serveis, en les dues assignatures se n'ha vist (o es veuran dins d'aquest apartat) gran part (marcades amb ^{v}) i sens dubte, amb els coneixements obtinguts, l'alumne podrà ràpidament desplegar totes aquelles que necessiti i que no s'han tractat en aquests cursos.

A continuació, veurem algunes eines (molt útils en entorns DevOps) més orientades a generar automatitzacions en les instal·lacions o generar entorns aïllats de desenvolupaments/test/execució que permeten de manera simple i fàcil (i sense pèrdues de prestacions/rendiment) disposar d'eines i entorns adequats a les nostres necessitats.

3.1. Linux Containers, LXC

LXC (Linux Containers) és un mètode de virtualització en un nivell del sistema operatiu per a executar múltiples sistemes Linux aïllats (anomenats *contenidors*) sobre un únic *host*. El *kernel* de Linux utilitza `cgroups` per a poder aïllar els recursos (CPU, memòria, E/S, *network*, etc.), la qual cosa no requereix iniciar cap màquina virtual. *Cgroups* també proveeix aïllament dels espais de noms per a aïllar per complet l'aplicació del sistema operatiu, inclosos arbre de processos, xarxa, ID d'usuaris i sistemes d'arxius muntats. Mitjançant una API molt potent i eines simples, permet crear i gestionar contenidors de sistema o aplicacions. LXC utilitza diferents mòduls del *kernel* (ipc, uts, mount, pid, network, user) i d'aplicacions (Apparmor, SELinux profiles, Seccomp policies, Chroots -pivot_root- i Control groups -cgroups-) per a crear i gestionar els contenidors. Es pot considerar que LXC és a mig camí d'un "potent" `chroot` i una màquina virtual, i ofereix un entorn molt proper a un Linux estàndard però sense necessitat de tenir un kernel separat. Això és més eficient que utilitzar virtualització amb un `hypervisor` (KVM, VirtualBox) i més ràpid de (re)iniciar, sobretot si s'estan fent desenvolupaments i és necessari fer-ho freqüentment, i el seu impacte en el rendiment és molt baix (el contenidor no ocupa recursos) i tots es dedicaran als processos que s'estiguin executant. L'únic inconvenient de la utilització de LXC és que només es poden executar

sistemes que suporten el mateix *kernel* que el seu amfitrió, és a dir, no podrem executar un contenidor BSD en un sistema Debian.

La instal·lació es fa mitjançant l'ordre `apt-get install lxc lxctl` i es poden instal·lar altres paquets addicionals que són opcionals (`bridge-utils libvirt-bin debootstrap`), no obstant això, si volem que els contenidors tinguin accés a la xarxa, amb IP pròpia és convenient instal·lar `apt-get install bridge-utils`. Per a preparar el *host*, primer hem de muntar el directori `cgroup` afegint a */etc/fstab* la següent línia `cgroup /sys/fs/cgroup cgroup defaults 0 0` i verificant que podem fer `mount -a` i el sistema *cgroups* apareixerà muntat quan executem l'ordre `mount`. Podeu veure detalls i possibles solucions a errors a [23, 24, 25]. A partir d'aquest moment, podem verificar la instal·lació amb `lxc-checkconfig`, que donarà una sortida similar a:

```
Found kernel config file /boot/config-3.2.0-4-amd64
--- Namespaces ---
 Namespaces: enabled
 Utsname namespace: enabled
 Ipc namespace: enabled
 Pid namespace: enabled
 User namespace: enabled
 Network namespace: enabled
 Multiple /dev/pts instances: enabled

--- Control groups ---
 Cgroup: enabled
 Cgroup clone_children flag: enabled
 Cgroup device: enabled
 Cgroup sched: enabled
 Cgroup cpu account: enabled
 Cgroup memory controller: enabled
 Cgroup cpuset: enabled

--- Misc ---
 Veth pair device: enabled
 Macvlan: enabled
 Vlan: enabled
 File capabilities: enabled
```

Si sorgeixen opcions deshabilitades, s'ha de mirar la causa i solucionar-ho (encara que algunes poden estar-ho). Si bé la distribució inclou un contenidor Debian (*/usr/share/lxc/templates*), és recomanable obtenir-ne un des de l'adreça de https://github.com/simonvanderveldt/lxc-debian-wheezy-template, que dóna solució a alguns problemes que té l'original quan s'instal·la sobre Debian Wheezy. Per a això, podem fer:

```
wget https://github.com/simonvanderveldt/lxc-debian-wheezy-template/raw/master/lxc-debian-wheezy-robvdhoeven
     -O /usr/share/lxc/templates/lxc-debianW
host# chown root:root /usr/share/lxc/templates/lxc-debianW
host# chmod +x /usr/share/lxc/templates/lxc-debianW
```

A continuació, es crea el primer contenidor com a `lxc-create -n mycont -t debianW` (en aquest cas, l'anomenem **mycont** i utilitzem el *template* de

debianW, però també pot ser Debian, que és el que inclou la distribució). Si desitgem configurar la xarxa, primer hem de configurar el *bridge* modificant (en el *host*) */etc/network/interfícies* amb el següent:

```
# The loopback network interface
auto lo br0
iface lo inet loopback

# The primary network interface
iface eth0 inet manual          # la posem en manual per evitar problemes
iface br0 inet static           # inicialitzem el bridge
address 192.168.1.60
netmask 255.255.255.0
gateway 192.168.1.1
bridge_ports eth0
bridge_stp off       # disable Spanning Tree Protocol
    bridge_waitport 0    # no delay before a port becomes available
    bridge_fd 0          # no forwarding delay
```

Fem un `ifdown eth0` i després un `ifup br0` i verifiquem amb `ifconfig` i per exemple `ping google.com` que tenim connectivitat. Després, modifiquem l'arxiu de configuració */var/lib/lxc/mycont/config* per a inserir el *bridge*:

```
lxc.utsname = myvm
lxc.network.type = veth
lxc.network.flags = up
lxc.network.link = br0          #ha d'existir en el host
lxc.network.ipv4 = 192.168.1.100/24     # Ip per al contenidor 0.0.0.0 indica dhcp
lxc.network.hwaddr = 00:1E:1E:1a:00:00
```

Les ordres més útils per a gestionar els contenidors són:

1) Per a iniciar l'execució com a *daemon* -en *background*- (el login/*passwd* per defecte és *root*/*root*): `lxc-start -n mycont -d`.

2) Per a connectar-nos al contenidor: `lxc-console -n mycont` "Ctrl+a q" per a sortir de la consola.

3) Per a iniciar el contenidor annexat a la consola: `lxc-start -n mycont` (en *foreground*)

4) Per a parar l'execució del contenidor: `lxc-stop -n myvm`

5) Per a iniciar els contenidors al *boot* de manera automàtica s'haurà de fer un enllaç de l'arxiu de configuració en el directori */etc/lxc/acte/*, per exemple `ln -s /var/lib/lxc/mycont/config /etc/lxc/auto/mycont`

6) Per a muntar sistemes d'arxius externs dins del contenidor, agregem a l'arxiu */var/lib/lxc/mycont/config* la línia

`lxc.mount.entry=/path/in/host/mount_point /var/lib/lxc/mycont/rootfs/mount_moint none bind 0 0`

i reiniciem el contenidor.

S'ha d'anar amb compte quan s'inicia el contenidor sense `-d`, ja que no hi ha manera de sortir (en la versió actual, el 'Ctrl+a q' no funciona). Cal iniciar sempre els contenidors en *background* (amb `-d`) tret que necessiti depurar perquè el contenidor no arrenca. Una altra consideració que cal tenir és que l'ordre `lxc-halt` executarà el `telinit` sobre l'arxiu */run/initctl* si existeix i apagarà el *host*, per la qual cosa cal apagar-lo amb `lxc-stop` i no fer un `shutdown -h now` dins del contenidor, ja que també apagarà el *host*. També existeix un panell gràfic per a gestionar els contenidors via web, http://lxc-webpanel.github.io/install.html (en Debian hi ha problemes amb l'apartat de xarxa; alguns d'aquests els soluciona https://github.com/vaytess/lxc-web-panel/tree/lwp-backup. Per a això, hem de clonar el lloc

```
git clone https://github.com/vaytess/lxc-web-panel.git
```

i després reemplaçar el directori obtingut per */srv/lwp* -renombrar aquest abans- i fer un `/etc/init.d/lwp restart`). També és important notar que la versió disponible en Debian és la 0.8 i en el web del desenvolupador* és la 1.04, que té molts dels errors corregits i la seva compilació és molt simple (vegeu l'arxiu INSTALL dins del paquet) per la qual cosa, si s'hi ha de treballar, es recomana aquesta versió.

***https://linuxcontainers.org /downloads/**

3.2. Docker

Docker és una plataforma oberta per al desenvolupament, empaquetatge i execució d'aplicacions de manera que es puguin posar en producció o compartir més ràpidament separant aquestes de la infraestructura, de manera que sigui menys costós en recursos (bàsicament espai de disc, CPU i engegada) i que estigui tot preparat per al següent desenvolupador amb tot el que es va posar però res de la part d'infraestructura. Això significarà menys temps per a provar i accelerarà el desplegament escurçant de manera significativa el cicle entre que s'escriu el codi i passa a producció. Docker empra una plataforma (contenidor) de virtualització lleugera amb fluxos de treball i eines que l'ajuden a administrar i implementar les aplicacions i proporcionar una manera d'executar gairebé qualsevol aplicació en forma segura en un contenidor aïllat. Aquest aïllament i seguretat permeten executar molts contenidors de manera simultània en el *host* i, atesa la naturalesa (lleugera) dels contenidors, tot això s'executa sense la càrrega addicional d'un *hypervisor* (que seria l'altra manera de gestionar aquestes necessitats compartint la VM), la qual cosa significa que podem obtenir millors prestacions i utilització dels recursos. És a dir, amb una VM cada aplicació virtualizada inclou no només l'aplicació, que poden ser desenes de Mbytes -binaris + biblioteques-, sinó també el sistema operatiu *guest*, que poden ser diversos Gbytes; en canvi, en Docker el que es denomina DE (Docker Engine) només comprèn l'aplicació i les seves dependències, que s'executen com un procés aïllat a l'espai d'usuari del SO *host*, compartint el *kernel* amb altres contenidors. Per tant, té el benefici de l'aïllament i l'assignació de recursos de les màquines virtuals, però és molt més portàtil i eficient

transformant-se en l'entorn perfecte per a donar suport al cicle de vida del desenvolupament de programari, test de plataformes, entorns, etc.[33]

L'entorn està format per dos grans components: **Docker** [34] (és la plataforma de virtualització –contenidor) i **Docker Hub** [35] (una plataforma SasS que permet obtenir i publicar/gestionar contenidors ja configurats). En la seva arquitectura, Docker utilitza una estructura client-servidor en què el client (CLI **Docker**) interactua amb el *daemon* Docker, que fa el treball de la construcció, execució i distribució dels contenidors de Docker. Tant el client com el *daemon* es poden executar en el mateix sistema, o es pot connectar un client a un *daemon* de Docker remot. El client Docker i el servei es comuniquen mitjançant *sockets* o una API RESTful.

Per a la instal·lació, no estan disponibles els paquets sobre Debian Wheezy (sí ho estaran sobre Jessie), i hem d'actualitzar el kernel, ja que Docker necessita una versió 3.8 (o superior). Per a això, carregarem el nou kernel 3.14 (juliol del 2014) des de Debian Backports executant:

```
echo "deb http://ftp.debian.org/debian/ wheezy-backports main non-free contrib"
     > /etc/apt/sources.list.d/backport.list
apt-get update
apt-get -t wheezy-backports install linux-image-amd64 linux-headers-amd64
reboot
```

Una vegada que hem seleccionat el nou *kernel* durant l'arrencada, podem fer:

```
Instal·lem unes dependències:
    apt-get install apt-transport-https
Importem la key d'Ubuntu:
    apt-key adv --keyserver hkp://keyserver.ubuntu.com:80 --recv-keys
    36A1D7869245C8950F966E92D8576A8BA88D21E9
Agreguem el repositori de Docker:
    echo "deb http://get.docker.io/ubuntu docker main" > /etc/apt/sources.list.d/docker.list"
Actualitzem i instal·lem:
    apt-get update
    apt-get install lxc-docker
Ara podem verificar si la instal·lació funciona executant una imatge d'Ubuntu (local)
dins del seu contenidor:
    docker run -i -t ubuntu /bin/bash           (Crtl+D per a sortir del contenidor)
Des de dins, podrem executar more lsb-release i ens indicarà:
        DISTRIB_ID=Ubuntu
        DISTRIB_RELEASE=14.04
        DISTRIB_CODENAME=trusty
        DISTRIB_DESCRIPTION="Ubuntu 14.04 LTS"
```

Però també podem fer `docker run -i -t centos /bin/bash`. Com que la imatge no la té en local, la descarregarà del Docker Hub i l'executarà.

Fent more */etc/os-release*, ens indicarà que és un CentOS Linux, 7 (Core) i ens donarà informació complementària.

Abans de carregar el primer contenidor, ens pot donar un error com *Cannot start container ... mkdir /sys/fs/devices: operation not permitted*. Això és a causa d'una instal·lació prèvia de LXC i només hem de comentar en l'arxiu */etc/fstab* la línia `#cgroup /sys/fs/cgroup cgroup defaults 0 0`, reini-

ciar la màquina i executar novament el contenidor. Les ordres per a iniciar-se (a més de les que hem vist) són:

1) `docker`: mostra totes les opcions.

2) `docker images`: fa una llista de les imatges localment.

3) `docker search` *patró*: busca contenidors/imatges que tinguin aquest patró.

4) `docker pull` *nom*: obté imatges del *hub*. El nom pot ser <username>/<repository> per als particulars.

5) `docker run name cmd`: executarà el contenidor i a dins, l'ordre indicada per `cmd`.

6) `docker ps -l`: permet obtenir l'ID i informació d'una imatge.

7) `docker commit ID name`: actualitza la imatge amb què s'ha instal·lat fins a aquest moment (amb 3-4 nombres de l'ID és suficient).

8) `docker inspect`: mostra les imatges corrent (similar a docker ps).

9) `docker push`: salva la imatge en el *hub* (hem de registrar-nos primer) i està disponible per a altres usuaris/*hosts* que la vulguin instal·lar amb tot el que hem configurat dins.

10) `docker cp`: copia arxius/directoris des del contenidor al *host*.

11) `docker export/import`: exporta/importa el contenidor en un arxiu tar.

12) `docker history`: mostra la història d'una imatge.

13) `docker info`: mostra informació general (imatges, directoris, etc.).

14) `docker kill`: per a l'execució d'un contenidor.

15) `docker restart`: reinicia un contenidor.

16) `docker rm`: elimina un contenidor.

17) `docker rmi`: elimina imatges.

18) `docker start/stop`: inicia/atura un contenidor.

3.3. Puppet

Puppet és una eina de gestió de configuració i automatització en sistemes TI (llicència Apache –abans GPL). El seu funcionament és gestionat mitjançant arxius (anomenats *manifiests*) de descripcions dels recursos del sistema i els seus estats, utilitzant un llenguatge declaratiu propi de l'eina. El llenguatge Puppet pot ser aplicat directament al sistema, o compilat en un catàleg i distribuït al sistema de destinació utilitzant un model client-servidor

(mitjançant una API REST), en què un agent interpel·la els proveïdors específics del sistema per a aplicar el recurs especificat en els *manifiests*. Aquest llenguatge permet una gran abstracció que habilita els administradors per a descriure la configuració en termes d'alt nivell, com ara usuaris, serveis i paquets sense necessitat d'especificar les ordres del sistema operatiu (`apt`, `dpkg`, etc.).[36, 37, 38] L'entorn Puppet té dues versions: Puppet (*Open Source*) i Puppet Enterprise (producte comercial). Les diferències entre aquestes es poden veure a http://puppetlabs.com/puppet/enterprise-vs-open-source. A més s'integren amb aquests entorns altres eines com MCollective (*orchestration framework*), Puppet Dashboard (consola web però abandonada en el seu desenvolupament), PuppetDB *(datawarehouse* per a Puppet), Hiera (eina de cerca de dades de configuració), Facter (eina per a la creació de catàlegs) i Geppetto (IDE -*integrated development environment*- per a Puppet).

3.3.1. Instal·lació

És important començar amb dues màquines que tinguin els seus *hostname* i definició a */etc/hosts* correctament i que siguin accessibles mitjançant la xarxa amb les dades obtingudes del */etc/hosts* (és important que el nom de la màquina -sense domini- d'aquest arxiu coincideixi amb el nom especificat en el *hostname* ja que, si no, hi haurà problemes quan es generin els certificats).

```
Obtenim els repositoris i instal·lem puppetmaster en el servidor:
    wget http://apt.puppetlabs.com/puppetlabs-release-wheezy.deb
    dpkg -i puppetlabs-release-wheezy.deb
    apt-get update
    apt-get install puppetmaster
Puppetlabs recomana executar la següent ordre abans de córrer puppetmaster:
    puppet resource service puppetmaster ensure=running enable=true
    service puppetmaster restart

Sobre el client, s'ha d'instal·lar els repositoris (tres primeres instruccions) i instal·lar puppet
(si el client és Ubuntu, no és necessari instal·lar el repositori):
    apt-get install puppet
Sobre el client, s'ha de modificar /etc/puppet/puppet.conf per a agregar en la secció [main] el
servidor (sysdw.nteum.org en el nostre cas) i reiniciar:
    [main]
    server=sysdw.nteum.org
    /etc/init.d/puppet restart
Després, s'ha d'executar sobre el client:
    puppet agent  --waitforcert 60 --test
Aquesta ordre enviarà una petició de signatura del certificat al servidor
i esperarà que aquest l'hi signi, per la qual cosa, ràpidament sobre el servidor s'ha de fer:
    puppetca --list
Ens respondrà amb alguna cosa com "pucli.nteum.org" (...) i executem:
    puppetca --sign pucli.nteum.org
Veurem informació per part del servidor i també per part del client i l'execució dels
catàlegs que hi hagi pendents.
```

D'aquesta manera, tenim instal·lat el client i el servidor i ara haurem de fer el nostre primer 'manifest'. Per a això, organitzarem l'estructura d'acord amb les recomanacions de PuppetLabs fent un arbre que tindrà la següent estructura a partir de */etc/puppet*:

```
+-- auth.conf
+-- autosign.conf
+-- environments
|   +-- common
|   +-- development
|   |   +-- modules
|   +-- production
|       +-- modules
|           +-- ntp
+-- etckeeper-commit-post
+-- etckeeper-commit-pre
+-- files
|   +-- shadow.sh
+-- fileserver.conf
+-- manifests
|   +-- nodes
|   |   +-- client1.pp
|   |   +-- server.pp
|   +-- site.pp
+-- modules
|   +-- accounts
|   |   +-- manifests
|   |       +-- init.pp
|   |       +-- system.pp
|   +-- elinks
|   |   +-- manifests
|   |       +-- init.pp
|   +-- nmap
|       +-- manifests
|           +-- init.pp
+-- node.rb
+-- puppet.conf
+-- rack
+-- templates
```

Començarem per l'arxiu */etc/puppet/manifests/site.pp*, que com a contingut tindrà `import 'nodes/*.pp'`. En el directori */etc/puppet/manifests/nodes* tindrem dos arxius (server.pp i client1.pp) amb el següent contingut:

```
Arxiu server.pp:
    node 'sysdw.nteum.org' {
    }
Arxiu client1.pp:
   node 'pucli.nteum.org' {
        include nmap
        include elinks
}
```

On definim dos serveis que cal instal·lar en el client pucli.nteum.org (`nmap` i `elinks`). Després, definim els directoris */etc/puppet/modules/elinks/manifests*, i */etc/puppet/modules/nmap/manifests* on hi haurà un arxiu *init.pp* amb la tasca que s'ha de fer:

```
Arxiu /etc/puppet/modules/elinks/manifests/init.pp:
    class elinks {
        case $operatingsystem {
            centos, redhat: {
                package { " elinks":
                ensure => installed,}}
            debian, ubuntu: {
                package { " elinks":
                ensure => installed,}}
```

```
            }
    }
Arxiu /etc/puppet/modules/nmap/manifests/init.pp:
   class nmap {
    case \$operatingsystem {
        centos, redhat: {
            package { "nmap":
            ensure => installed,}}
        debian, ubuntu: {
            package { "nmap":
            ensure => installed,}}
        }
    }
```

En aquests, podem veure la selecció del SO i després les indicacions del paquet que cal instal·lar. Tot això es podria haver posat en l'arxiu inicial (*site.pp*), però es recomana fer-ho així per a millorar la visibilitat/estructura i que puguin aprofitar-se les dades per a diferents instal·lacions. Sobre el servidor, hauríem d'executar `puppet apply -v /etc/puppet/manifests/site.pp`, i sobre el client, per a actualitzar el catàleg (i instal·lar les aplicacions) `puppet agent -v -test` (en cas de no fer-ho, el client s'actualitzarà al cap de 30 minuts per defecte). Es podrà verificar sobre el client que els paquets s'han instal·lat i fins i tot es poden desinstal·lar des de la línia d'ordres i executar l'agent que els tornarà a instal·lar.[37]

Com a següent acció, procurarem crear un usuari i posar-li un *password*. Comencem creant un mòdul */etc/puppet/modules/accounts/manifests* i, dins d'aquest, dos arxius anomenats *init.pp* i *system.pp* amb el següent contingut:

```
Arxiu /etc/puppet/modules/accounts/manifests/system.pp:
    define accounts::system ($comment,$password) {
    user { $title:
       ensure => 'present',
       shell => '/bin/bash',
       managehome => true,}
    }
Arxiu /etc/puppet/modules/accounts/manifests/init.pp:
    class accounts {
    file { '/etc/puppet/templates/shadow.sh':
        ensure => file,
        recurse => true,
        mode => "0777",
        source => "puppet:///files/shadow.sh",}
    @accounts::system { 'demo':
       comment      => 'demo users',
       password     => '*',}
    exec { "demo":
          command => 'echo "demo:123456" | chpasswd',
          provider => 'shell',
          onlyif => " /etc/puppet/templates/shadow.sh demo",}
}
```

En l'arxiu system, hem definit el tipus d'accounts::system per a assegurar que cada usuari té directori HOME i *shell* (en lloc dels valors predeterminats de l'ordre `useradd`), contrasenya i el camp Gecos. Després, en el *init.pp* indiquem el nom de l'usuari que cal crear i el camp Geco. Per a crear aquest usuari sense *passwd*, hem de modificar *client1.pp* perquè quedi:

```
Arxiu client1.pp:
   node 'pucli.nteum.org' {
     #include nmap
     #include elinks
include accounts
    realize (Accounts::System['demo'])
}
```

Com es pot observar, hem comentat el que no volem que s'executi, ja que el nmap i l'elinks es van instal·lar en l'execució anterior. Aquí indiquem que s'inclogui el mòdul *accounts* i que es faci l'acció. Amb això només crearíem usuaris, però no els modificaríem el *password*. Hi ha diferents maneres d'inicialitzar el *password*, ja que només s'ha d'executar una vegada i quan no estigui inicialitzat i aquí en seguirem una, vista a [37]. Per a això, utilitzarem un *script* que haurem d'enviar des del servidor al client, per la qual cosa hem de fer:

```
Arxiu /etc/puppet/fileserver.conf modificar:
    [files]
        path /etc/puppet/files
        allow *
Arxiu /etc/puppet/auth.conf incloure:
    path /files
        auth *
Arxiu /etc/puppet/puppet.conf en la secció [main] incloure:
        pluginsync = true
Tornar a arrencar el servidor: /etc/init.d/puppetmaster restart
```

Ara crearem un *script /etc/puppet/files/shadow.sh* que ens permetrà saber si hem de modificar el *passwd* del */etc/shadow* o no:

```
    #!/bin/bash
    rc=` /bin/grep $1 /etc/shadow | awk -F":" '($2 == " ! ")' | wc -l `
    if [ $rc -eq 0 ]
        then
            exit 1
        else
            exit 0
        fi
```

I ja tenim les modificacions en l'arxiu */etc/puppet/modules/accounts/init.pp* on li diem quin arxiu cal transferir i on (*file*) i després l'exec, que permet canviar el *passwd* si l'*script* ens retorna un 0 o un 1. Després haurem d'executar novament sobre el servidor `puppet apply -v /etc/puppet/manifests/site.pp` i sobre el client per a actualitzar el catàleg `puppet agent -v -test` i verificar si l'usuari s'ha creat i podem accedir-hi. Com a complement a Puppet, hi ha el Puppet Dashboard, però aquest programari està sense manteniment, per la qual cosa no es recomana instal·lar-lo (tret que sigui estrictament necessari –indicacions en https://wiki.debian.org/puppetdashboard). En el seu lloc es pot instal·lar **Foreman** [39, 40], que és una aplicació que s'integra molt bé amb Puppet i permet veure en una consola tota la informació i seguiment, així com fer el desplegament i l'automatització de manera gràfica d'una instal·lació. La instal·lació és molt simple [40, 41]:

```
Inicialitzem el repositori i instal·lem el paquet foreman-installer:
        echo "deb http://deb.theforeman.org/ wheezy stable" > \
            /etc/apt/sources.list.d/foreman.list
        wget -q http://deb.theforeman.org/foreman.asc -O- | apt-key add -
        apt-get update
        apt-get install -i foreman-installer
Executem l'instal·lador amb -i (interactive) per a verificar els setting:
        foreman-installer -i

Contestem (y) a la pregunta i seleccionem les opcions:
1. Configure foreman, 2. Configure foreman_proxy, 3. Configure puppet.
Veurem que el procés pot acabar amb uns errors (Apache), però no hem de preocupar-nos
i hem de reiniciar el servei amb service apache2 restart.

Ens connectem a l'URL https://sysdw.nteum.org/, acceptem el certificat i entrem
amb usuari admin i passwd changeme.
Sobre Infrastructure > Smart Proxy fem clic en New Proxy i seleccionem el nom
i l'URL: https://sysdw.nteum.org:8443.

El següent és fer que s'executi el puppet sobre el servidor (si no està posat
en marxa, s'ha d'engegar service puppet start i pot ser necessari modificar
l'arxiu /etc/default/puppet i executem puppet agent -t).

Sobre Foreman, veurem que el DashBoard canvia i actualitza els valors per a la gestió
integrada amb puppet.
```

3.4. Chef

Chef és una altra de la grans eines de configuració amb funcionalitats similars a Puppet (hi ha un bon article en què s'efectua una comparació sobre el dilema de Puppet o Chef [42]). Aquesta eina utilitza un llenguatge (basat en Ruby) DSL (*domain-specific language*) per a escriure les "receptes" de configuració que seran utilitzades per a configurar i administrar els servidors de la companyia/institució i que a més pot integrar-se amb diferents plataformes (com Rackspace, Amazon EC2, Google i Microsoft Azure) per a, automàticament, gestionar la provisió de recursos de noves màquines. L'administrador comença escrivint les receptes que descriuen com maneja Chef les aplicacions del servidor (com ara Apache, MySQL, o Hadoop) i com seran configurades, i indica quins paquets hauran de ser instal·lats, els serveis que hauran d'executar-se i els arxius que s'hauran de modificar, i a més informarà de tot això en el servidor perquè l'administrador tingui el control del desplegament. Chef es pot executar en mode client/servidor o en mode *standalone* (anomenat Chef-solo). En el mode C/S, el client envia diversos atributs al servidor i el servidor utilitza la plataforma `Solr` per a indexar aquests i proveir l'API corresponent, i utilitza aquests atributs per a configurar el node. Chef-solo permet utilitzar les receptes en nodes que no tinguin accés al servidor i només necessita "la seva" recepta (i les seves dependències), que ha d'estar en el disc físic del node (Chef-solo és una versió amb funcionalitat limitada de Chef-client). Chef, juntament amb Puppet, CFEngine i Bcfg2, és una de les eines més utilitzades per a aquest tipus de funcionalitat i que ja forma part de les mitjanes-grans instal·lacions actuals de GNU/Linux (és interessant veure en els detalls del desplegament de la infraestructura Wikipedia que, excepte *passwords* i certificats, tot està documentat a http://blog.wikimedia.org/2011/09/19/ever-wondered-how-the-wikimedia-servers-are-configured/).

La instal·lació bàsica de Chef-server i Chef-client pot ser una mica més complicada, però comencem igualment amb màquines que tinguin */etc/hosts* i *hostname* ben configurades. En aquest cas, suggerim treballar amb màquines virtuals Ubuntu (14.04 si pot ser), ja que hi ha dependències de la biblioteca Libc (necessita la versió 2.15) i Ruby (1.9.1) que en Debian Wheezy generen problemes (fins i tot utilitzant el repositori experimental). El primer pas és baixar-nos els dos paquets de *http://www.getchef.com/chef/install/* i executar en cadascun `dpkg -i paquet- corresponent.deb` (és a dir, el servidor a la màquina servidora i el client a la màquina client). Després de cert temps en el servidor, podem executar `chef-server-ctl reconfigure`, que reconfigurarà tota l'eina i crearà els certificats. Quan acabi, podrem connectar-nos a l'URL *https://sysdw.nteum.org/* amb usuari i *password* admin/pssw0rd1.

Sobre la interfície gràfica, cal anar a *Client* > *Create* i crear un client amb el nom desitjat (pucli en el nostre cas), marcar la casella Admin i fer *Create Client*. Sobre la següent pantalla es generaran les claus privades i pública per a aquest client i haurem de copiar i salvar la clau privada en un arxiu (per exemple */root/pucli.pem*). Sobre el client, haurem de crear un directori */root/.chef* i des del servidor li copiem la clau privada `scp /root/pucli.pem pucli:/root/.chef/pucli.pem`. Sobre el client fem `knife configuri` i les dades hauran de quedar com:

```
log_level                :info
log_location             STDOUT
node_name                'pucli'
client_key               '/root/.chef/pucli.pem'
validation_client_name   'chef-validator'
validation_key           '/etc/chef/validation.pem'
chef_server_url          'https://sysdw.nteum.org'
cache_type               'BasicFile'
cache_options( :path => '/root/.chef/checksums' )
cookbook_path ['/root/chef-repo/cookbooks' ]
```

Després podrem executar `knife client list` i ens haurà de mostrar els clients, una cosa com:

```
chef-validator
chef-webui
pucli
```

A partir d'aquest punt, estem en condicions de crear la nostra primera recepta (*cookbook*) però donada la complexitat de treballar amb aquest paquet i els coneixements necessaris, recomanem començar treballant en el client com a Chef-solo per a automatitzar l'execució de tasques i passar després a descriure les tasques del servidor i executar-les remotament. Per a això, recomanem seguir la guia a http://gettingstartedwithchef.com/.[44, 43, 45]. Si bé s'han de fer alguns canvis quant als *cookbook* que s'han d'instal·lar, la seqüència és correcta i permet aprendre a treballar i repetir els resultats en tants servidors com sigui necessari (es recomana fer el procediment en un i després sobre una altra

màquina sense cap paquet, instal·lar el Chef-solo, copiar el repositori del primer i executar el Chef-solo per a tenir un altre servidor instal·lat exactament igual que el primer).

3.5. Vagrant

Aquesta eina és útil en entorns DevOps i té un paper diferent del de Docker però orientat cap als mateixos objectius: proporcionar entorns fàcils de configurar, reproduïbles i portàtils amb un únic flux de treball que ajudarà a maximitzar la productivitat i la flexibilitat en el desenvolupament d'aplicacions/serveis. Pot proveir de màquines de diferents proveïdors (VirtualBox, VMware, AWS, o altres) utilitzant *scripts*, Chef o Puppet per a instal·lar i configurar automàticament el programari de la VM. Per als desenvolupadors, Vagrant aïllarà les dependències i configuracions dins d'un únic entorn disponible, consistent, sense sacrificar cap de les eines que el desenvolupador utilitza habitualment i tenint en compte un arxiu anomenat *Vagrantfile*. La resta de desenvolupadors tindrà el mateix entorn encara que treballin des d'uns altres SO o entorns, de manera que s'aconseguirà que tots els membres d'un equip estiguin executant codi en el mateix entorn i amb les mateixes dependències. A més, en l'àmbit TI permet tenir entorns d'un sol ús amb un flux de treball coherent per a desenvolupar i provar *scripts* d'administració de la infraestructura, ja que ràpidament es poden fer proves de *scripts*, *cookbooks* de Chef, mòduls de Puppets i altres que utilitzen la virtualització local, com VirtualBox o VMware. Després, amb la mateixa configuració, pot provar aquests *scripts* en el *cloud* (per exemple, AWS o Rackspace) amb el mateix flux de treball.

En primer lloc, s'ha d'indicar que haurem de treballar sobre un GNU/Linux base (el que es defineix com a *bare metal*, és a dir, sense virtualizar, ja que necessitarem les extensions de HW visibles i si hem virtualitzat amb VirtualBox, per exemple, això no és possible). És recomanable instal·lar la versió de VirtualBox (pot ser la del repositori Debian), verificar que tot funciona i després descarregar l'última versió de Vagrant des de *http://www.vagrantup.com/downloads.html* i instal·lar-la amb `dpkg -i vagrant_x.y.z_x86_64.deb` (on x.y.z serà la versió que hem descarregat).

A partir d'aquest punt, és molt simple executant l'ordre [46] `vagrant init hashicorp/precisi32`, que inicialitzarà/generarà un arxiu anomenat **Vagrantfile** amb les definicions de la VM, i quan executem **vagrant up** descarregarà del repositori *cloud* de Vagrant una imatge d'Ubuntu 12.04-32b i l'engegarà. Per a accedir-hi, simplement hem de fer `vagrant ssh`, i si volem rebutjar-la, `vagrant destroy`. En el *cloud* de Vagrant [47], podem accedir a diferents imatges preconfigurades* que podrem carregar simplement fent `vagrant box add nom`, per exemple `vagrant box add chef/centos-6.5`. Quan s'executi l'ordre `up` veurem que ens dóna una sèrie de missatges, entre els quals ens indicarà el port per a connectar-se a aquesta màquina virtual directament (per exemple, `ssh vagrant@localhost -p 2222` i amb *passwd*

***https://vagrantcloud.com /discover/featured**

vagrant). Per a utilitzar una VM com a base, podem modificar l'arxiu *Vagrantfile* i canviar-ne el contingut:

```
Vagrant.configure("2") do |config|
  config.vm.box = "hashicorp/precise32"
end
```

Si desitgem treballar amb dues màquines de manera simultània, haurem de modificar l'arxiu Vagrantfile amb el següent:

```
Vagrant.configure("2") do |config|
  config.vm.define "centos" do |centos|
    centos.vm.box = "chef/centos-6.5"
  end
  config.vm.define "ubu" do |ubu|
    ubu.vm.box = "hashicorp/precise32"
  end
end
```

Podrem veure que assigna un port SSH a cadascuna per a evitar col·lisions quan fem el `vagrant up`. Des de la VM podríem instal·lar el programari com es fa habitualment, però per a evitar que cada persona faci el mateix hi ha una forma d'aprovisionament que s'executarà quan es faci el `up` de la VM. Per a això, escrivim un *script init.sh* amb el següent contingut:

```
#!/usr/bin/env bash
apt-get update
apt-get install -y apache2
rm -rf /var/www
ln -fs /tmp /var/www
```

En aquest *script* (per a no tenir problemes de permisos per a aquesta prova) hem apuntant el directori */var/www* a */tmp*. A continuació, modifiquem el Vagrantfile (només hem deixat una VM per accelerar els processos de càrrega):

```
Vagrant.configure("2") do |config|
  config.vm.define " ubu" do |ubu|
    ubu.vm.box = "hashicorp/precise32"
    ubu.vm.provision :shell, path: " init.sh"
  end
end
```

Després, haurem de fer `vagrant reload -provision`. Veurem com s'aprovisiona i carrega Apache i després, si entrem a la màquina i creem un fitxer */tmp/index.html* i fem `wget 127.0.0.1` (o la IP interna de la màquina), veurem que accedim a l'arxiu i el baixem (igualment, si fem `ps -edaf | grep apache2` veurem que està funcionant). Finalment, i per a veure-la des del *host*, hem de redireccionar el port 80 per exemple al 4444. Per a això, hem d'agregar en el mateix arxiu (sota ubu.vm.provision) la línia:

```
config.vm.network :forwarded_port, host: 4444, guest: 80.
```

Tornem a fer `vagrant reload -provision` i des del *host* ens connectem a l'URL: *127.0.0.1:4444* i veurem el contingut de l'*index.html*.

En aquestes proves de concepte, hem mostrat alguns aspectes interessants però és una eina molt potent que s'ha d'analitzar amb cura per a configurar les opcions necessàries per al nostre entorn, com per exemple aspectes del *Vagrant Share* o qüestions avançades del *Provisioning* que aquí només hem tractat superficialment.[46]

Activitats

1. Instal·leu i configureu OpenMPI sobre un node; compileu i executeu el programa `cpi.c` i observeu el seu comportament.

2. Instal·leu i configureu OpenMP; compileu i executeu el programa de multiplicació de matrius (https://computing.llnl.gov/tutorials/openMP/exercise.html) en 2 *cores* i obteniu proves de la millora en l'execució.

3. Utilitzant dos nodes (pot ser amb VirtualBox), Cacti i XYmon i monitoreu el seu ús.

4. Utilitzant Rocks i VirtualBox, instal·leu dues màquines per a simular un clúster.

5. Instal·leu Docker i creeu 4 entorns diferents.

6. Instal·leu Puppet i configureu un màquina client.

7. Ídem que en el punt anterior amb Chef.

8. Amb Vagrant creeu dues VM, una amb Centos i una altra amb Ubuntu i aprovisioneu la primera amb Apache i la segona amb MySQL. S'ha de poder accedir a les màquines des del *host* i s'han de poder comunicar entre elles.

Bibliografia

[1] *Beowulf cluster*.
<http://en.wikipedia.org/wiki/Beowulf_cluster>

[2] **S. Pereira**. *Building a simple Beowulf cluster with Ubuntu*.
<http://byobu.info/article/Building_a_simple_Beowulf_cluster_with_Ubuntu/>

[3] **Radajewski, J.; Eadline, D.** *Beowulf: Installation and Administration*. TLDP.
<http://www2.ic.uff.br/ vefr/research/clcomp/Beowulf-Installation-and-Administration-HOWTO.html>

[4] **Swendson, K**. *Beowulf HOWTO* (tlpd).
<http://www.tldp.org/HOWTO/Beowulf-HOWTO/>

[5] **Barney, B.** *OpenMP*. Lawrence Livermore National Laboratory.
<https://computing.llnl.gov/tutorials/openMP/>

[6] *OpenMP Exercise*.
<https://computing.llnl.gov/tutorials/openMP/exercise.html>

[7] *MPI 3*.
<http://www.mpi-forum.org/docs/mpi-3.0/mpi30-report.pdf>

[8] *MPI Examples*.
<http://www.mcs.anl.gov/research/projects/mpi/usingmpi/examples/main.htm>
(Download Section)

[9] *Mpich1 Project*.
<http://www.mcs.anl.gov/research/projects/mpi/>

[10] *MPICH, High-Performance Portable MPI* .
<http://www.mpich.org/>

[11] *LAM/MPI*.
<http://www.lam-mpi.org/ >

[12] *OpenMPI*.
<http://www.open-mpi.org/>

[13] *FAQ OpenMPI*.
<http://www.open-mpi.org/faq/>

[14] **Woodman, L.** *Setting up a Beowulf cluster Using Open MPI on Linux*.
<http://techtinkering.com/articles/?id=32>

[15] *Rocks Cluster. Base Users Guide*.
<http://central6.rocksclusters.org/roll-documentation/base/6.1.1/>

[16] *Cacti*.
<http://www.cacti.net/>

[17] *Xymon*.
<http://xymon.sourceforge.net/>

[18] *Cloud Computing. Retos y Oportunidades*.
<http://www.ontsi.red.es/ontsi/sites/default/files/2-_resumen_ejecutivo_cloud_computing_vf.pdf>

[19] *Eucalyptus, CloudStack, OpenStack and OpenNebula: A Tale of Two Cloud Models*.
<http://opennebula.org/eucalyptus-cloudstack-openstack-and-opennebula-a-tale-of-two-cloud-models/>

[20] *OpenNebula vs. OpenStack: User Needs vs. Vendor Driven*.
<http://opennebula.org/opennebula-vs-openstack-user-needs-vs-vendor-driven/>

[21] *OpenNebula Documentation*.
<http://opennebula.org/documentation/>

[22] *Quickstart: OpenNebula on Ubuntu 14.04 and KVM*.
<http://docs.opennebula.org/4.6/design_and_installation/quick_starts/qs_ubuntu_kvm.html>

[23] *LXC*.
<https://wiki.debian.org/LXC>

[24] *LXC en Ubuntu*.
<https://help.ubuntu.com/lts/serverguide/lxc.html>

[25] *LXC: Step-by-Step Guide*.
<https://www.stgraber.org/2013/12/20/lxc-1-0-blog-post-series/>

[26] **S. Graber**. *LXC 1.0*.
<https://wiki.debian.org/BridgeNetworkConnections>

[27] *A Short List of DevOps Tools*.
<http://newrelic.com/devops/toolset>

[28] *Preseeding d-i en Debian*.
<https://wiki.debian.org/DebianInstaller/Preseed>

[29] *FAI (Fully Automatic Installation) for Debian GNU/Linux*.
<https://wiki.debian.org/FAI>

[30] *FAI (Fully Automatic Installation) project and documentation*.
<http://fai-project.org/>

[31] *El libro del administrador de Debian. Instalación automatizada*.
<http://debian-handbook.info/browse/es-ES/stable/sect.automated-installation.html>

[32] *Octopussy. Open Source Log Management Solution*.
<http://8pussy.org/>

[33] *Understanding Docker*.
<https://docs.docker.com/introduction/understanding-docker/>

[34] *Docker Docs*.
<https://docs.docker.com/>

[35] *Docker HUB*.
<https://registry.hub.docker.com/>

[36] *Puppet Labs Documentation*.
<http://docs.puppetlabs.com/>

[37] *Puppet - Configuration Management Tool*.
<http://puppet-cmt.blogspot.com.es/>

[38] *Learning Puppet*.
<http://docs.puppetlabs.com/learning/ral.html>

[39] *Foreman - A complete lifecycle management tool*.
<http://theforeman.org/>

[40] *Foreman - Quick Start Guide*.
<http://theforeman.org/manuals/1.5/quickstart_guide.html>

[41] *How to Install The Foreman and a Puppet Master on Debian Wheezy*.
<http://midactstech.blogspot.com.es/2014/02/PuppetMasterForeman-Wheezy.html>

[42] *Puppet or Chef: The configuration management dilemma*.
<http://www.infoworld.com/d/data-center/puppet-or-chef-the-configuration-management-dilemma-215279?page=0,0>

[43] *Download Chef: Server and Client*.
<http://www.getchef.com/chef/install/>

[44] **Gale, A.** *Getting started with Chef*.
<http://gettingstartedwithchef.com/>

[45] *Chef Cookbooks*.
<https://supermarket.getchef.com/cookbooks-directory>

[46] *Vagrant: Getting Started*.
<http://docs.vagrantup.com/v2/getting-started/index.html >

[47] *Vagrant Cloud*.
<https://vagrantcloud.com/>

[48] *KVM*.
<https://wiki.debian.org/es/KVM>

[49] *VirtualBox*.
<https://wiki.debian.org/VirtualBox>

[50] *Guía rápida de instalación de Xen*.
<http://wiki.debian.org/Xen>

[51] *The Xen hypervisor*.
<http://www.xen.org/>

[52] *Qemu*.
<http://wiki.qemu.org/Main_Page>

[53] *QEMU. Guía rápida de instalación e integración con KVM y KQemu*.
<http://wiki.debian.org/QEMU>