Università degli Studi di Bologna "Alma Mater Studiorum"

FACOLTÀ DI SCIENZE MATEMATICHE, FISICHE E NATURALI
Corso di laurea in Informatica

Tesi di laurea triennale

ShockVM

Progetto e realizzazione di un cluster eterogeneo di macchine virtuali

Candidato:
Massimo Gengarelli
Matricola 0000274802

Relatore:
Prof. Renzo Davoli

Anno Accademico 2010–2011
Seconda sessione

Indice

1 Introduzione **7**
1.1 Confronto con altre tecnologie 8
1.2 Struttura del documento 10

2 Presentazione del progetto **11**
2.1 ShockVM Cluster 11
2.1.1 Lista delle macchine 12
2.1.2 Il server virtuale 13
2.1.3 Il router virtuale 13
2.2 ShockVM Live 15
2.2.1 Creare una macchina virtuale 15
2.2.2 Importare una macchina virtuale 17
2.2.3 Esportare una macchina virtuale 18
2.2.4 Visualizzare una macchina virtuale 19
2.2.5 Condividere una macchina virtuale 21

3 Dettagli implementativi **22**
3.1 ShockVM Cluster 22
3.1.1 Virtual Distributed Ethernet 22
3.1.2 Il server LDAP 23
3.1.3 Le “home” e il server NFS 26
3.1.4 Il server DHCP 27
3.1.5 Il server DNS 28
3.1.6 Crediti 30
3.2 ShockVM Live 31
3.2.1 Google Web Toolkit 31
3.2.2 XML 32
3.2.3 Configurazione del framework 33
3.2.4 Gestione degli utenti 33
3.2.5 Creazione di una macchina virtuale 35
3.2.6 Visualizzazione di una macchina 37
3.2.7 Esportazione di una macchina 40
3.2.8 Condivisione di una macchina 43
3.2.9 Il sistema di notifiche 45
3.2.10 Crediti 49

4 Conclusioni ed osservazioni **50**
4.1 Possibili sviluppi 50
4.2 Ringraziamenti personali 52

Elenco delle figure

1 Il Wizard per la creazione delle nuove macchine 15
2 La nuova macchina compare nel menù principale 16
3 Descrizione completa della macchina 16
4 Wizard per importare una macchina virtuale 17
5 Le notifiche di importazione 17
6 Wizard per esportare una macchina virtuale 18
7 Le notifiche di esportazione 18
8 Boot di Debian 19
9 LiveCD di Ubuntu che mostra il client fare uso dei servizi del cluster virtuale. In questo esempio si nota l'indirizzo IP assegnato dinamicamente ed il server DNS 20
10 È anche possibile terminare l'esecuzione di una macchina virtuale tramite il pannello principale 20
11 Completamento automatico dei nomi utente 21
12 Il proprietario della macchina compare fra parentesi 21

Memorandum

"There's that line from Newton about standing on the soulders of the giants. We're all standing on Dennis' shoulders."
– Brian Kernighan
remembering Dennis Ritchie (Sept 9, 1941 – Oct 12, 2011)

1 Introduzione

ShockVM ("**S**hockVM's an **H**eterogeneous **O**perating **C**luster of **K**VM's **V**irtual **M**achines") è un progetto suddiviso in due parti atte a dimostrare due tesi differenti, seppure una strettamente dipendente dall'altra.

La prima parte, identificata nel resto del documento con il nome "**ShockVM Cluster**", consiste nel dimostrare la possibilità di ottenere una cooperazione totale fra macchine virtuali appartenenti a sistemi operativi ed architetture differenti, ovvero un **insieme "eterogeneo" di macchine**; ciò che si è ottenuto è un cluster *completamente virtuale* molto simile a quello *fisico* utilizzato quotidianamente degli studenti di questo corso di laurea. Dal cluster vengono erogati differenti servizi che saranno presentati con maggiore dettaglio nel corso del documento: un server LDAP[1] è responsabile di mantenere una "directory" di utenti e gruppi, così da ottenere un unico set di credenziali d'accesso per qualsiasi macchina del cluster, garantendo quindi che ogni utente sia provvisto di un'unica coppia utente-password funzionante su qualsiasi macchina; un server DNS[2] è installato e configurato in modo tale da permettere sia la risoluzione dei nomi delle macchine in indirizzi IP, sia una facilitazione amministrativa nel qual caso si voglia trasferire un servizio da una macchina all'altra: sono infatti presenti nomi sia per le singole macchine che per i singoli servizi; un server DHCP[3] è incaricato di gestire il "pool" di indirizzi IPv4 disponibili, assegnando IP statici a determinate macchine (quelle del cluster) e IP dinamici ad altre (quelle create con ShockVM Live); un server NFS[4] rende disponibile l'accesso ad un singolo disco contenente tutte le home degli utenti da parte di qualsiasi macchina: così facendo si garantisce agli utenti stessi la presenza dei medesimi dati su qualsiasi macchina del cluster; gli altri servizi comprendono un server per effettuare compilazioni distribuite (DistCC), un DBMS, due server web,... la presenza di questi servizi è giustificata per il solo motivo di rendere palese la dimostrazione della tesi stessa: è possibile avere un cluster perfettamente operativo anche se costruito su macchine di natura molto differente, questo grazie alle astrazioni presentate dai sistemi operativi. Le macchine verranno presentate con maggiore dettaglio nelle sezioni successive, l'insieme di architetture presenta macchine x86, x86_64, arm little endian, mips little endian e PowerPC mentre i sistemi operativi utilizzati sono principalmente GNU/Linux (ArchLinux, Debian e Ubuntu) e BSD (FreeBSD). Una macchina "speciale" viene utilizzata per permettere agli utenti di connettersi al cluster il quale non è — per motivi di tutela dell'ambito universitario — direttamente accessibile tramite Internet.

La seconda parte, identificata con il nome "**ShockVM Live** " tratta la realizzazione di un'applicazione web in grado di permettere ai suoi utenti la creazione, gestione, condivisione e visualizzazione di macchine virtuali che andranno a sfruttare i servizi messi a disposizione da ShockVM Cluster: verranno principalmente utilizzati i servizi DHCP e DNS. Si può pensare a ShockVM Live come ad un *"social network per macchine virtuali"*: il suo scopo è quello di permettere agli utenti di interagire fra loro al fine di poter ottenere macchine

[1] Lightweight Directory Access Protocol
[2] Domain Name System
[3] Dynamic Host Configuration Protocol
[4] Network FileSystem

utilizzabili per qualsiasi tipo di sperimentazione. È anche possibile "esportare" una macchina virtuale, la quale diventerà quindi "importabile" ed utilizzabile da qualsiasi utente. **ShockVM Live** è quasi completamente scritto utilizzando Java e il framework GWT[5] (Google Web Toolkit), presenta alcune parti in Javascript e chiaramente gli elementi stilistici dell'applicazione sono gestiti tramite CSS. ShockVM Live verrà presentato più nel dettaglio nel corso del documento. L'unico requisito fondamentale per poter usufruire dell'applicazione web è di disporre di un browser il cui engine sia in grado di comprendere e visualizzare HTML5: WebKit (Google Chrome, Apple Safari, Chromium ...) e Gecko (Firefox) sono due esempi di engine funzionanti.

1.1 Confronto con altre tecnologie

Per quanto concerne il cluster in realtà non è possibile effettuare un confronto con altre tecnologie a causa della sua stessa natura: si tratta infatti di un'architettura di rete "standard" con servizi che è facile ritrovare in altre installazioni di questo tipo; tuttavia, la peculiarità di questa parte del progetto è la totale assenza di strutture fisiche ("tangibili"): come già spiegato nel paragrafo precedente qualsiasi macchina (router compreso) è parte di un processo di emulazione e persino i "cavi di rete" e gli "switch" sono virtualizzati. La sua architettura è volutamente simile a quella utilizzata quotidianamente dagli studenti di questo stesso corso di laurea, sebbene i processi server utilizzati per implementare i servizi non siano gli stessi utilizzati dal cluster cs.unibo.it[6].

Un'analisi differente invece dev'essere effettuata per quanto riguarda l'applicazione web. Sebbene non esistano ad oggi servizi completi che permettano il livello di collaborazione che è possibile raggiungere tramite ShockVM Live, è vero anche che si può comunque raggiungere il risultato sfruttando differenti tecnologie, proprietarie e non. A seguire un elenco di applicazioni che potrebbero servire allo stesso scopo.

1. **FLOZ** (Free Live OS Zoo) è un progetto di tre anni fa, il cui scopo è quello di permettere agli utenti di utilizzare immagini di macchine virtuali "prefabbricate". FLOZ fu creato da Mattia Gentilini (ex studente di questo stesso corso di laurea) per poter essere utilizzato in parallelo con il progetto OSZoo.org, un archivio Internet di distribuzioni Linux al quale qualsiasi utente può collaborare, di Stefano Marinelli (anche lui ex studente di questo stesso corso di laurea). Insieme, permettono di "provare live" alcune delle immagini che è possibile scaricare da OSZoo.org. Tuttavia, questa soluzione presenta diversi limiti, in particolare:

   (a) non è possibile creare in modo semplice nuove macchine virtuali: una macchina dev'essere creata sul proprio computer personale, caricata negli archivi di OSZoo.org e successivamente si può richiederne l'inclusione in FLOZ tramite e-mail agli amministratori;

[5] Verrà introdotto più avanti nel corso di questo stesso documento

[6] All'interno del cluster virtuale ad esempio si utilizza il protocollo LDAP per la gestione delle directory degli utenti mentre in cs.unibo.it si utilizza NIS. Il risultato ottenuto tuttavia è fondamentalmente lo stesso

(b) le macchine create con FLOZ non dispongono di alcun disco, ovvero qualsiasi modifica alla macchina che si vuole provare non è permanente, nel momento in cui si effettua lo shutdown, la macchina ritorna nello stato in cui era stata avviata;

(c) non è possibile visualizzare la stessa macchina virtuale da più persone contemporaneamente (ovvero nessuna forma di collaborazione è possibile);

(d) la tecnologia utilizzata per mostrare la macchina virtuale risulta essere particolarmente pesante e obsoleta, richiedendo anche all'utente l'installazione di software di terze parti (java applet engine);

(e) le macchine utilizzate non possono in alcun modo essere "agganciate" ad una rete;

tuttavia, per la realizzazione di ShockVM Live, è stato inizialmente preso ad esempio il funzionamento stesso di FLOZ. In un certo senso, l'applicazione web può essere vista come un miglioramento di FLOZ, rendendolo quindi un progetto "staccato" da OSZoo.org e dando maggiore libertà agli utenti. Al momento della stesura del documento né FLOZ né OSZoo.org risultano essere raggiungibili a causa di un guasto hardware della macchina che ospitava entrambi i progetti.

2. Microsoft™ **Hyper-V Server**®[2] è un prodotto proprietario sviluppato da Microsoft il cui scopo è quello di permettere la virtualizzazione nativa (qualora supportata dall'hardware) del sistema operativo Microsoft™ Windows Server®; è possibile visualizzare qualsiasi "istanza" del Sistema virtualizzato all'interno di un sistema di Cloud Computing, rendendo di fatto possibile la cooperazione simultanea[2] da parte di client differenti. La soluzione proposta da Microsoft, tuttavia, è una soluzione proprietaria, fornita solamente a pagamento e limitata al singolo sistema operativo sopracitato; inoltre non si tratta di un'applicazione web "pubblica" in quanto per poter accedere alle istanze delle macchine è necessario disporre di un account reale sulla macchina server che avvia le istanze virtualizzate.

3. VMWare **vCloud Director**™ è probabilmente la soluzione che più si avvicina a quella proposta: permette una creazione facile di macchine "template" da parte dell'utente, ovvero l'amministratore del server mette a disposizione degli utenti un set di macchine "prefabbricate" dal quale gli utenti possono iniziare a creare (ed utilizzare) le proprie; permette a più utenti di collaborare su di una stessa macchina virtuale e l'accesso al sistema avviene via web. Tuttavia:

   (a) si tratta di un sistema proprietario e a pagamento, le licenze vengono fornite solo ad aziende e per ogni licenza è possibile creare non più di 25 macchine virtuali;

   (b) è un sistema fortemente centralizzato e per poterlo utilizzare è necessario disporre di un server su cui risulti attiva un'altra licenza VMWare apposita. Lo spostamento di vCloud Director da un server all'altro dev'essere autorizzata da VMWare stessa e può comportare diverse spese;

(c) il sistema di emulazione utilizza macchine virtuali create con VMWare le quali utilizzano formati proprietari per i dischi virtuali quindi l'importazione locale di una macchina virtuale è possibile solo continuando ad usufruire dei prodotti VMWare.

La necessità era quindi quella di creare un sistema che concedesse agli utenti la più assoluta libertà verso le proprie macchine virtuali create, che fosse liberamente utilizzabile sfruttando unicamente un browser web e che fornisse adeguati sistemi per poter garantire la collaborazione simultanea di più utenti. Più avanti nel documento verranno affrontati con maggiore precisione tutti i "punti chiave" di ShockVM Live.

1.2 Struttura del documento

Al termine dell'introduzione, il documento presenta un'analisi di entrambe le parti del progetto soffermandosi su quali siano i punti focali di ciascuna delle due e fornendone una piccola guida "user-friendly"; il capitolo immediatamente successivo invece cercherà di trattare gli stessi elementi ponendo più attenzione sui dettagli tecnici e giustificando le scelte effettuate. Fondamentalmente, il prossimo capitolo risponde alla domanda "**cosa fa ShockVM?**" mentre quello successivo risponde alla domanda "**come fa a farlo?**".

2 Presentazione del progetto

ShockVM è un progetto che si compone di due parti principali:

- *ShockVM Cluster*: un cluster di macchine virtuali eterogenee cooperative il cui scopo è quello di fornire servizi alle macchine "client";
- *ShockVM Live*: un'applicazione web che permette la gestione di macchine virtuali all'interno della rete del cluster, ovvero permette la creazione e condivisione delle macchine "client".

La prima parte (*il cluster*) fornisce i servizi che saranno poi resi disponibili alle macchine virtuali create dagli utenti all'interno dell'applicazione web.

Mentre è possibile pensare al cluster come ad un progetto *stand-alone*, non è possibile fare il contrario. L'applicazione web, infatti, necessita dei servizi messi a disposizione dalle macchine virtuali del cluster per poter essere sfruttata al meglio. Servizi basilari (come l'accesso ad Internet) sono messi a disposizione dell'applicazione da parte delle macchine del cluster.

2.1 ShockVM Cluster

Le macchine del cluster virtuale compongono il cuore del progetto: senza i servizi forniti da esse, infatti, l'applicazione web non avrebbe quasi alcun senso. I servizi principali offerti dal cluster rendono possibile l'accesso da e verso il mondo esterno alle macchine virtuali che verranno poi utilizzate all'interno dell'applicazione web.

Il cluster si compone principalmente di 10 macchine, ognuna delle quali fornisce un servizio differente (server dhcp, directory ldap, ...) ed opera su un'architettura ed un sistema operativo differente dalle altre.

Tuttavia, per moitivi di semplificazione, è possibile sfruttare anche una sola macchina (vd. "Il server virtuale") ed ottenere comunque lo stesso risultato. Il server virtuale infatti raccoglie tutti gli altri servizi - comprese le configurazioni.

Lo scopo di questa parte del progetto è di dimostrare che è possibile far collaborare fra loro macchine virtuali di natura completamente differente fino alla completa formazione di un cluster copia di un cluster reale. Per la realizzazione dell'intero cluster è stato preso come esempio il cluster di macchine di cs.unibo.it.

Per ragioni di sicurezza, l'accesso a queste macchine è riservato a pochi utenti e la creazione degli account sulla directory LDAP è completamente manuale. Il server DHCP è inoltre configurato per garantire a queste macchine sempre lo stesso indirizzo IP così da poter sfruttare al meglio anche il server DNS interno, responsabile di risolvere i nomi delle macchine nei corrispettivi indirizzi IP.

Tutte le macchine virtuali sono facilmente accessibili da remoto anche in caso di "guasto" della rete virtuale tramite il protocollo *VNC*. L'accesso al server VNC è garantito solo se la connessione proviene da un indirizzo IP della rete 130.136.0.0, in questo modo solo chi possiede un account sulle macchine di virtualsquare (v2.cs.unibo.it) può accedervi (tramite tunnel SSH).

2.1.1 Lista delle macchine

Come già anticipato, le macchine che compongono il cuore di *ShockVM Cluster* sono 10.

Macchina	**IP**	**Breve descrizione**
vincent	10.0.42.5	Questa è probabilmente la macchina più importante del cluster. Usata "stand-alone" fornisce tutti i servizi basilari per il corretto funzionamento dell'intero cluster. Tuttavia, se utilizzata in modalità "cooperativa" fornisce solamente i servizi DHCP, NFS e directory LDAP. La macchina presenta un'architettura x86 e distribuzione Debian Lenny.
mia	10.0.42.10	Questa macchina è una ArchLinux amd64 con pre-installato un ambiente grafico (GNOME). Fornisce il server DNS del cluster.
marsellus	10.0.42.15	Debian Lenny armel, fornisce il servizio DISTCC per permettere la compilazione distribuita di codice sorgente.
butch	10.0.42.20	Ubuntu 10.04 su architettura amd64. Questa macchina presenta un DBMS (PostgreSQL) raggiungibile da tutte le altre macchine all' interno del cluster - ma non direttamente dall'esterno. Il DBMS può essere utilizzato per la creazione e manutenzione di un sito web interno.
jules	10.0.42.25	FreeBSD 8.1-RELEASE su architettura x86. Questa macchina fornisce un server DNS di backup, viene sfruttato solo nel caso in cui il server DNS primario (operativo su vincent o su mia) non sia disponibile. L'installazione di pacchetti aggiuntivi su questa macchina tuttavia si è rivelato particolarmente oneroso, a causa del fatto che il gestore di pacchetti di FreeBSD effettua compilazioni da codice sorgente.
pumpkin	10.0.42.30	Fedora 13 amd64 con l'ambiente grafico GNOME installato. Questa macchina fornisce il server web nginx. Al momento nessun sito è ospitato. I server DNS sono configurati in modo tale da risolvere verso questo indirizzo sia il nome pumpkin.virtuacluster.cs.unibo.it che il nome wiki.virtuacluster.cs.unibo.it.

wolf	10.0.42.35	CentOS 5 su architettura x86. Questa macchina fornisce un DBMS (MySQL) raggiungibile direttamente solo dalle macchine del cluster virtuale. Il suo utilizzo è perfettamente identico a quello della macchina butch.
honeybunny	10.0.42.40	Debian Lenny su architettura mipsel. Questa macchina fornisce un server web (Apache). Il suo utilizzo è perfettamente identico a quello della macchina pumpkin. II server DNS sono configurati in modo tale da risolvere verso questo indirizzo sia il nome honeybunny.virtuacluster.cs.unibo.it che il nome web.virtuacluster.cs.unibo.it.
lance	10.0.42.45	Debian Lenny su architettura PPC (emulazione di un Motorola 68k tramite Qemu). Questa macchina non fornisce alcun servizio a causa del fatto che per questa architettura mancano molte librerie fondamentali all'interno della distribuzione Debian.
vdefloz	dinamico	Debian Lenny su architettura x86. Questa macchina fornisce i servizi di vde pubbliche assieme ad un'interfaccia di rete virtuale configurata in bridging con il virtuacluster per poter permettere l'accesso alle macchine dall'esterno tramite vde. Le modalità di connessione verranno discusse più avanti all'interno del documento (vd sez. "Il router virtuale").

2.1.2 Il server virtuale

La macchina vincent è quindi la macchina fondamentale dell'intero cluster. Senza di essa non si avrebbero né le home degli utenti condivise tramite NFS né la directory LDAP degli utenti stessi (utilizzata per garantire un account unico per tutte le macchine).

2.1.3 Il router virtuale

Per poter sfruttare al meglio le potenzialità delle macchine (sia delle macchine core che delle macchine client) è necessario poterle interfacciare ad Internet, senza che però la connessione in uscita sia effettuata tramite gli indirizzi IP universitari, questo per ovvie ragioni di sicurezza e di tutela dell'Università stessa. La macchina **vdefloz** serve esattamente a questo scopo. Si può pensare a questa macchina come ad un router: è posizionata fra il mondo esterno e il cluster virtuale e accetta connessioni (solo sulla porta 22) dall'esterno. La sua connettività è molto limitata: grazie infatti al firewall netfilter configurato sulla

macchina fisica che la ospita (magari.v2.cs.unibo.it), questa macchina non ha accesso al mondo esterno, pur potendo ricevere connessioni.

Le sue due interfacce di rete sono collegate sia al virtuacluster che alla macchina fisica (entrambe tramite VDE).

La macchina è configurata in modo tale da poter accettare connessioni SSH da account senza password, questi account sono a loro volta configurati per fungere unicamente da tunnel broker per VDE. A seguire un estratto dei file **/etc/shadow** e **/etc/passwd** per mostrare la configurazione della macchina:

/etc/shadow con campo delle password vuoto

```
virtuacluster::15004:0:99999:7:::
vde1::15005:0:99999:7:::
vde2::15005:0:99999:7:::
vde3::15005:0:99999:7:::
vde4::15005:0:99999:7:::
vde5::15005:0:99999:7:::
vde6::15005:0:99999:7:::
info::15247:0:99999:7:::
```

/etc/passwd con campo delle home impostato su vde_plug

```
vde1:x:1002:1002:,,,:/home/vde1:/opt/vde/bin/vde_plug
vde2:x:1003:1003:,,,:/home/vde2:/opt/vde/bin/vde_plug
vde3:x:1004:1004:,,,:/home/vde3:/opt/vde/bin/vde_plug
vde4:x:1005:1005:,,,:/home/vde4:/opt/vde/bin/vde_plug
vde5:x:1006:1006:,,,:/home/vde5:/opt/vde/bin/vde_plug
vde6:x:1007:1007:,,,:/home/vde6:/opt/vde/bin/vde_plug
vde0:x:1000:1000:,,,:/home/vde0:/opt/vde/bin/vde_plug
info:x:1009:1010:,,,:/home/info:/bin/getinfo.sh
```

Per connettersi al cluster di macchine virtuali è quindi sufficiente aprire una nuova connessione vde verso *virtuacluster*@*vde*2.*v*2.*cs.unibo.it*.

2.2 ShockVM Live

Live ShockVM è la seconda parte del progetto, quella che sarà probabilmente la più utilizzata dagli utenti finali. Come già anticipato, si tratta di un'applicazione web che permette la creazione, la gestione e l'utilizzo delle macchine "client" che saranno poi connesse al cluster di macchine virtuali. In questa parte del documento non ne verrà descritta la parte implementativa ma solo le funzionalità principali disponibili. L'URL è http://forse.v2.cs.unibo.it:8080/LiveVC e per qualsiasi riferimento il contatto principale è gengarel@cs.unibo.it .

2.2.1 Creare una macchina virtuale

La creazione di una macchina virtuale in **ShockVM Live** è probabilmente una delle operazioni più comuni che è possibile effettuare. Innanzitutto è necessario, una volta effettuato il login, cliccare alla voce "Create a new machine from scratch" che compare sotto il menù che riporta in modo completo il nome dell'utente.

Una volta fatto questo, nella parte centrale della pagina compare il Wizard che aiuterà e guiderà l'utente nella creazione della macchina.

È possibile scegliere il disco d'installazione della macchina (tipicamente le distribuzioni GNU/Linux vengono distribuite con un'immagine ISO9660 detta "Live CD"), il nome della macchina virtuale, la dimensione del disco primario e - se si desidera averlo - la dimensione del disco secondario e la possibilità di avere una seconda interfaccia di rete collegata ad uno switch vde privato.

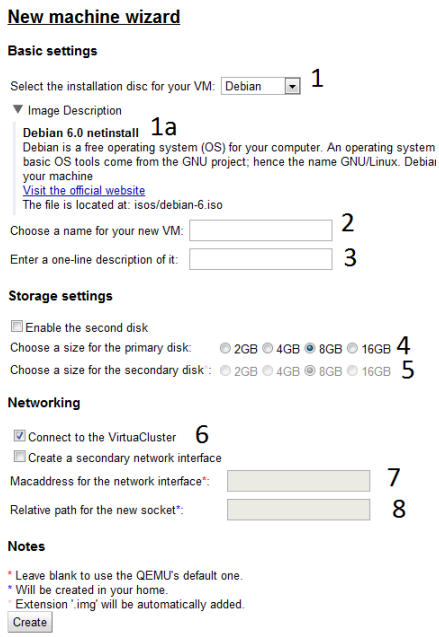


Figura 1: Il Wizard per la creazione delle nuove macchine

Come si può vedere nella figura accanto, il Wizard è già di per sè abbastanza esplicativo. Al punto 1 è possibile scegliere il disco d'installazione per la macchina virtuale; al punto 1a compare una breve descrizione del disco selezionato; al punto 2 bisogna inserire il nome della nuova macchina virtuale; al punto 3 una breve descrizione (una riga) per la macchina virtuale che si sta creando; al punto 4 si sceglie la dimensione del primo disco (obbligatorio) mentre al punto 5 la dimensione del secondo disco (facoltativo); al punto 6 si configurano le interfacce di rete; e ai punti 7 ed 8 s'inseriscono le impostazioni per il socket VDE privato. È chiaramente possibile inserire il nome di un socket già esistente. In tal caso il socket non verrà creato ma la macchina verrà solamente connessa ad esso.
Una volta completato il Wizard, cliccando sul bottone "Create" la macchina compare nel menù principale di sinistra. In caso di errori, l'utente verrà notifica-

to con un paragrafo di testo rosso che comparirà subito dopo le note, all'interno del Wizard stesso.

Figura 2: La nuova macchina compare nel menù principale

Cliccando sul nome della macchina comparirà un breve riassunto della stessa. Al punto 1 si potrà visualizzare la descrizione breve della stessa; al punto 2 viene visualizzato il nome del disco d'installazione scelto; al punto 3 un breve riassunto dei dischi della macchina; al punto 4 vengono mostrate le interfacce di rete della macchina stessa; al punto 5 sono presenti i nomi utente delle persone con le quali si sta condividendo la visione della macchina (verrà spiegato più avanti cosa significa); al punto 7 si seleziona con quale device la macchina deve effettuare il boot mentre al punto 8 sono presenti i comandi per far partire la macchina stessa (o per visualizzarla nel caso la macchina sia già attiva), per cancellarla e per impostare il tablet device.

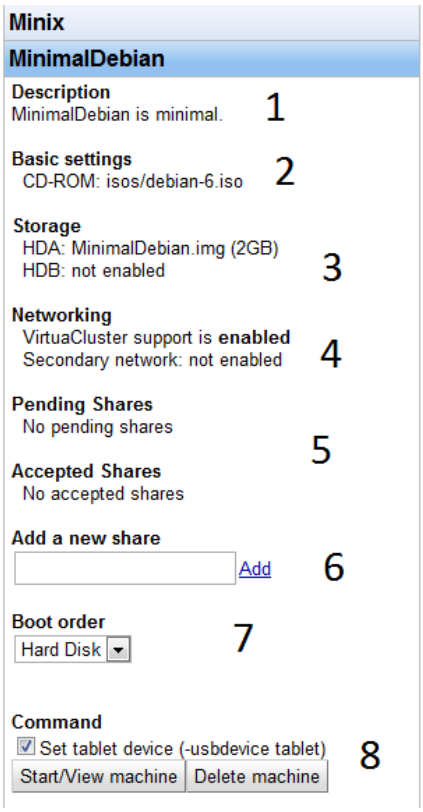


Figura 3: Descrizione completa della macchina

A questo punto la macchina è già pronta per essere avviata, esportata e condivisa con altri utenti del sistema. Nei prossimi punti verranno spiegate altre modalità per ottenere una macchina virtuale già esistente ("Importare una macchina virtuale"), per esportare una macchina virtuale o per condividere la visualizzazione di una macchina con altri utenti registrati a **ShockVM Live** .

2.2.2 Importare una macchina virtuale

Il processo d'importazione di una macchina è - come si può immaginare - molto meno complesso ed oneroso di quello di creazione di una macchina *ex-novo*. Per poter cominciare l'importazione di una macchina è sufficiente cliccare sulla voce "Create from an existing one" che compare nel menù principale nel sotto-menù personale.

Il Wizard che appare è diviso in base agli utenti che hanno già esportato le loro macchine virtuali. Molto semplicemente, è sufficiente navigare all'interno delle liste per cominciare il processo d'importazione di una macchina virtuale.

Una volta cliccato su "Import this machine", l'utente verrà notificato nel pannello di sinistra e potrà tenere traccia del processo d'importazione. Al termine dello stesso, comparirà un'ulteriore notifica.

Il processo d'importazione fallisce mostrando una pagina d'errore all'utente nel caso in cui l'utente abbia già importato quella stessa macchina.

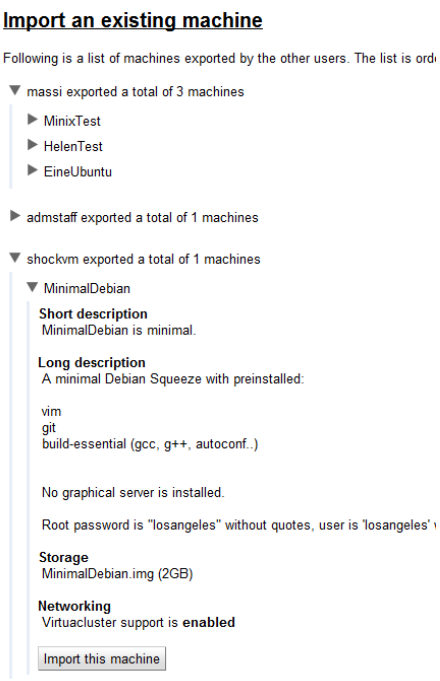
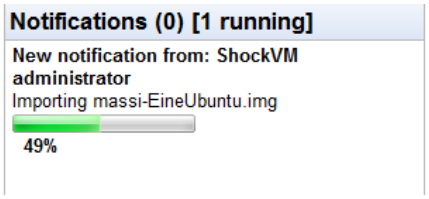
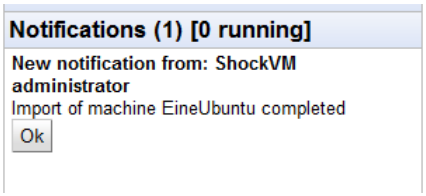


Figura 4: Wizard per importare una macchina virtuale



(a) Esportazione in corso



(b) Esportazione completa

Figura 5: Le notifiche di importazione

2.2.3 Esportare una macchina virtuale

ShockVM Live permette agli utenti di importare ed esportare macchine virtuali. Questo significa che una volta che una macchina ha raggiunto una "certa maturità" e vogliamo renderla disponibile a *tutti* gli altri utenti dell'applicazione possiamo farlo senza alcuna limitazione di sorta. È sufficiente infatti cliccare sul link "Export an existing machine" nel menù principale di sinistra, al sottomenù personale per far apparire il Wizard per l'esportazione della macchina virtuale.

Il Wizard è molto semplice e l'intero processo richiede pochi istanti. Al punto 1 si trova un menù a tendina elencante tutte le macchine virtuali possedute *in modo diretto* (i.e. non condivise da altri utenti) dall'utente, mentre al punto 2 è possibile inserire una descrizione più accurata della macchina che si vuole esportare. La descrizione lunga deve contenere tutte le informazioni fondamentali per l'utilizzo della macchina stessa da parte degli utenti che la importeranno in seguito. Ad esempio non possono mancare informazioni riguardanti la password dell'utente *root* (o equivalenti in caso di sistemi operativi non Unix-like), il parco software preinstallato, eventuali accorgimenti sulle modalità d'uso, eccetera. Una volta cliccato sul pulsante "Export this machine", salvo errori, il server comincerà la procedura d'esportazione della macchina e l'utente potrà tener traccia del progresso nel menù principale di sinistra, nel sotto-menù "Notifications". Terminata la procedura, l'utente verrà ulteriormente notificato con un semplice messaggio da parte del server.

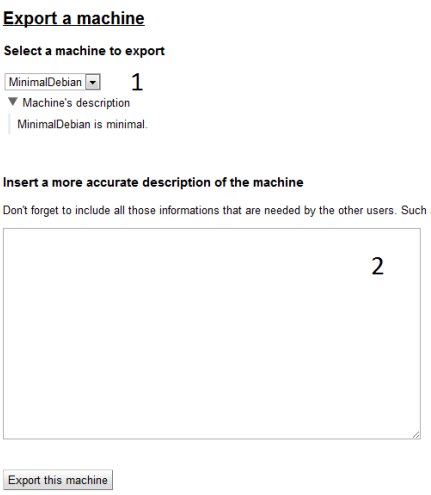
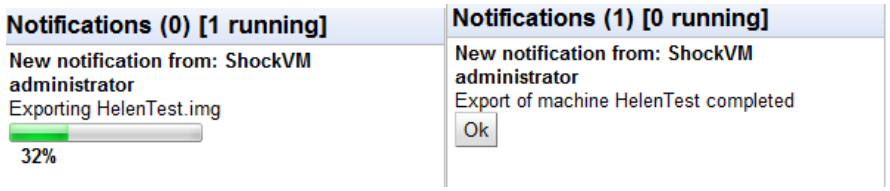


Figura 6: Wizard per esportare una macchina virtuale



(a) Esportazione in corso (b) Esportazione completa

Figura 7: Le notifiche di esportazione

2.2.4 Visualizzare una macchina virtuale

Una volta terminata la creazione (o l'importazione) di una macchina virtuale, è possibile avviarla semplicemente cliccando sul pulsante "Start/View machine" posizionato nel pannello principale alla voce della macchina stessa, dopo aver impostato il parametro relativo all'ordine di boot dei devices. Per poter visualizzare correttamente è necessario disporre di un engine che abbia il pieno supporto di HTML5, come ad esempio Gecko (Firefox), dalla versione 4.0.0 o Webkit (Google Chrome, Chromium, Safari, ..).

Figura 8: Boot di Debian

Come mostrato nella figura soprastante, una volta avviata, la macchina sarà visibile nella parte centrale della pagina, assieme a dei bottoni per controllare la visualizzazione del *canvas* che contiene il client VNC. I comandi che è possibile impartire al client VNC sono:

- "Release Keyboard", una volta avviato il client VNC "cattura" tutti gli eventi registrati dalla tastiera, per poter tornare a scrivere nei campi della pagina è necessario prima far riottenere il controllo degli eventi al browser. Questo bottone fa esattamente questo.
- "Grab Keyboard", l'esatto contrario del bottone precedente. Il controllo della tastiera torna completamente al client VNC.
- "Send C-A-D", invia l'evento "Ctrl - Alt - Del" alla macchina virtuale.

- "Disconnect", disconnette (ma non rilascia la tastiera) il client VNC. Questo pulsante **non** spegne la macchina virtuale, la macchina virtuale continuerà ad esistere nel server e ci si potrà riconnettere seguendo nuovamente la procedura iniziale.
- "30%" "60%" "90%" "100%" scala la visuale del client del fattore indicato.

Se in fase di creazione della macchina è stato scelto di connettere la stessa al cluster virtuale, la macchina potrà usufruire di tutti i servizi messi a disposizione dal cluster stesso, come ad esempio il server DHCP o il server DNS.

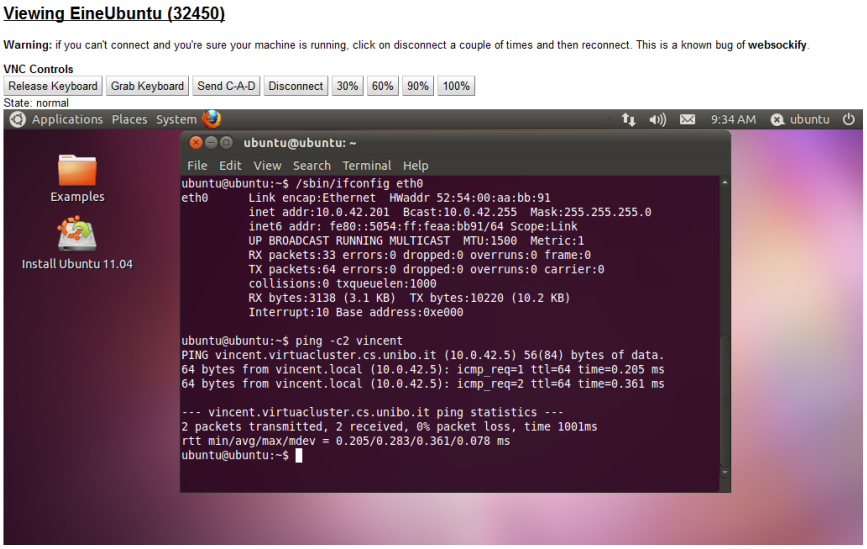


Figura 9: LiveCD di Ubuntu che mostra il client fare uso dei servizi del cluster virtuale. In questo esempio si nota l'indirizzo IP assegnato dinamicamente ed il server DNS

Una volta terminato l'utilizzo della macchina è **necessario** cliccare sul pulsante "Shutdown this machine" presente in fondo alla pagina, per poter liberare il server VNC in uso. È ovviamente consigliato effettuare *anche* uno shutdown "pulito" della macchina, tuttavia è *sempre necessario* cliccare sul pulsante menzionato (o sull'equivalente presente nel pannello di sinistra, alla voce "Active Machines" nel sotto-menù personale).

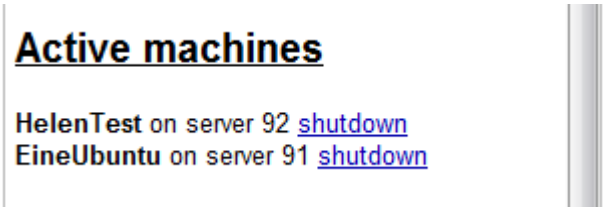


Figura 10: È anche possibile terminare l'esecuzione di una macchina virtuale tramite il pannello principale

2.2.5 Condividere una macchina virtuale

Come già anticipato, su **ShockVM Live** è possibile condividere la visualizzazione di una macchina virtuale con uno o più utenti dell'applicazione. Questo significa che sia il proprietario della macchina che gli utenti condivisi possono accedere *simultaneamente* alla macchina, qualora questa fosse stata attivata. Tuttavia, i permessi per esportare, cancellare, avviare e spegnere la macchina virtuale rimangono disponibili *solamente* al proprietario della macchina stessa.

Per condividere una macchina con utente è sufficiente iniziare a scrivere il nome dell'utente nel campo degli share della macchina che si vuole condividere. Comparirà quindi una lista dei possibili completamenti (aggiornata in tempo reale); una volta selezionato l'utente desiderato, cliccando sul link "Add" il server manderà una notifica al destinatario, chiedendo di confermare o rifiutare la condivisione della visualizzazione della macchina. Fino a quando il destinatario non risponderà la notifica, il proprietario vedrà il nome dello stesso nella lista "Pending Shares" (Condivisioni Pendenti) della macchina virtuale.

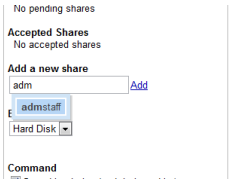


Figura 11: Completamento automatico dei nomi utente

Una volta accettata la richiesta di condivisione, il server manderà un'ulteriore notifica al mittente, contenente la scelta dell'utente (che può essere "Accepted" o "Refused"). Se il destinatario ha accettato la condivisione, il proprietario vedrà il nome dello stesso sotto la voce "Accepted Shares" nel pannello della macchina, assieme ad un link ("Remove") che permette al proprietario di eliminare quel determinato utente fra gli utenti condivisi. Completato il processo, il proprietario può avviare la macchina virtuale secondo la normale procedura e gli utenti visualizzarla nello stesso modo.

Figura 12: Il proprietario della macchina compare fra parentesi

La condivisione di una macchina virtuale permette a più utenti di lavorare in modo concorrente alla stessa macchina, bisogna quindi tenere conto del fatto che tutti gli utenti possono mandare eventi alla macchina nello stesso istante.

Attenzione: il server non effettua alcun processo di "semaforizzazione" per garantire la mutua esclusività degli eventi, è compito degli utenti stessi non interferire con il lavoro degli altri utenti eventualmente connessi. Questo comportamento è voluto ed è stato implementato in questo modo proprio per poter garantire la più assoluta libertà agli utenti stessi.

Per evitare conflitti sul nome della macchina, al nome della macchina degli utenti condivisi è aggiunto fra parentesi il nome del proprietario della macchina stessa, come illustrato nella figura a lato.

Con questo si conclude la sezione del documento dedicata agli utenti. Nelle prossime sezioni verranno illustrate più nel dettaglio le tecnologie utilizzate, le scelte effettuate ed una guida per una "facile" lettura dei sorgenti del progetto.

3 Dettagli implementativi

In questa parte del documento verranno discusse le implementazioni di entrambe le parti principali del progetto, cercando di entrare nel dettaglio il più possibile e di motivare - qualora necessario - le scelte effettuate. **ShockVM Live** è un'applicazione web di circa 10.000 righe di codice, divisa a sua volta in due parti fondamentali: la componente "client-side", ovvero il codice che viene eseguito direttamente sulla macchina dell'utente e la componente "server-side", ovvero il codice che viene eseguito unicamente sul server.

All'interno dei sorgenti, la divisione è netta: le classi client si trovano nella gerarchia it/unibo/cs/v2/client, mentre le classi server si trovano nella gerarchia it/unibo/cs/v2/server; per motivi pratici, tutte le interfacce delle servlet sono ulteriormente dislocate in it/unibo/cs/v2/servlets, mentre le classi condivise (utilizzate cioè sia dal client che dal server) si trovano in it/unibo/cs/v2/shared.

Le singole implementazioni verranno discusse nelle prossime sezioni.

3.1 ShockVM Cluster

Per quanto concerne il cluster virtuale, la sua realizzazione è stata abbastanza monotona. Una volta installato il sistema operativo per la determinata architettura della macchina, il lavoro consisteva nell'installazione e configurazione dei servizi che la macchina stessa avrebbe dovuto offrire alle altre macchine. Le macchine più complesse - senza le quali non esisterebbe il cluster stesso - sono sicuramente il *server virtuale* ("vincent") e il *router virtuale* ("vdefloz"), entrambe già introdotte e commentate nella sezione precedente.

In questa sezione verranno quindi discusse le configurazioni e gli utilizzi dei principali servizi offerti, omettendo i procedimenti d'installazione dei sistemi operativi in quanto si è sempre trattato di installazioni "standard", anche per architetture meno utilizzate nel quotidiano come mipsel e ppc.

Oltre a questo verrà poi spiegato come le macchine siano in grado di comunicare fra loro, anche se sono in esecuzione su macchine differenti, ovvero una descrizione della rete virtuale sottostante interamente costruita tramite VDE.

3.1.1 Virtual Distributed Ethernet

VDE ("Virtual Distributed Ethernet") è uno strumento utilizzato per creare ed utilizzare reti virtuali: permette l'interconnessione di macchine distanti fra loro attraverso Internet, creando di fatto delle LANs[7] completamente virtuali. Gli strumenti messi a disposizione da VDE sono virtualizzazioni di entità fisiche reali: switches, hubs e cavi. [3, see Introduction]
VDE è il "coltellino svizzero per il networking virtuale" [3, at page 27]: permette l'interoperabilità di macchine virtuali con macchine fisiche o altre macchine virtuali così come permette la connessione di macchine fisiche ad altre macchine fisiche, non ha limiti di protocollo ed è compatibile "off-the-shelf" con qualsiasi

[7] Local Area Networks

cosa sia funzionante tramite Ethernet [3]. All'interno del Cluster, VDE viene utilizzato per creare la rete virtuale che permette la cooperazione delle macchine. Ogni macchina *fisica* (i.e.: le macchine su cui vengono avviate le macchine virtuali) fornisce uno switch alle proprie macchine virtuali. Dal punto di vista del cluster, le macchine sono tutte connesse in un grafo "*1-hop*"[8].

3.1.2 Il server LDAP

Il primo dei servizi che qualsiasi macchina incontra ancora in fase di boot è sicuramente il server LDAP.

LDAP ("Lightweight Directory Access Protocol") è un protocollo standard per l'interrogazione e la modifica dei servizi di directory, usato principalmente per mantenere un unico database di utenti e/o gruppi "distribuito" all'interno della rete. Il protocollo permette di aggiungere, cercare, modificare e cancellare le voci all'interno della directory.
Il concetto di "directory" risale agli albori dell'informatica e andando ancora un po' a ritroso nel tempo il primo vero servizio di "directory di utenti" consiste nell'elenco telefonico. Sostanzialmente, un servizio tale in ambito informatico fornisce ai suoi utenti un modo veloce per localizzare facilmente altri utenti, altre risorse, servizi od informazioni. In un ambiente ancora più complesso come Internet, il servizio di directory gioca un ruolo ancora più vitale: permette l'affiliazione di persone, applicazioni e risorse residenti in macchine differenti, reti differenti o limiti geografici differenti. L'ambiente delle directory è costante, non tiene conto delle differenze (siano esse geografiche o di rete) e fornisce un aggregato affidabile. LDAP è in grado di comunicare tramite il protocollo standard OSI "X.500" [5].
Per poter utilizzare LDAP è necessario che vi sia una macchina server che mantiene il database della directory e che sia in grado di comunicare con le altre macchine attraverso il protocollo stesso. La macchina server deve quindi essere in esecuzione perché le altre macchine possano collegarsi.
Le operazioni di modifica, aggiunta e cancellazione sono garantite solo previa autenticazione via password, mentre per l'operazione di ricerca non è necessario inserire alcuna password. Il vantaggio principale nell'utilizzare un protocollo per servizi di directory invece dei classici "plain-text" di Unix (/etc/passwd ed /etc/shadow) risiede nel fatto che un utente registrato nel database ha accesso a *tutte* le macchine che compongono il core del cluster virtuale e quindi la manutenzione degli utenti risulta molto più semplice.

Lato client, l'autenticazione avviene tramite un modulo PAM opportunamente installato e configurato. Il concetto di "Pluggable Authentication Modules" (Moduli di autenticazione connettibili) è stato disegnato dalla Sun Microsystems e definito standard RFC successivamente. PAM fornisce un framework che consente agli amministratori ed ai produttori di personalizzare i servizi utilizzati per autenticare gli utenti in un sistema locale. Ad esempio, esistono moduli per permettere il logging degli accessi effettuati solo tramite il protocollo SSH, così come moduli più "esotici" come un CAPTCHA per il login via seriale. Tutti questi moduli sono formiti "a basso livello" e le API rimangono

[8] Ovvero un solo nodo di distanza fra una macchina e l'altra

costanti "ad alto livello", di modo che il sistema d'accesso sia sempre lo stesso, senza necessità di cambiare le API [1].

PAM è attualmente supportato da tutti i principali sistemi UNIX (AIX, BSD, GNU/Linux, Mac OSX e Solaris) ed è attualmente standardizzato come parte del processo di standardizzazione X/Open UNIX (XSSO).
Il modulo di autenticazione utilizzato per poter accedere e comunicare con il server LDAP è denominato "*pam_ldap.so*", distribuito tramite licenza "GNU LGPL". La configurazione del modulo consiste di tre parti fondamentali:

- la configurazione del file /etc/pam_ldap.conf, contenente le informazioni principali del modulo, ovvero come contattare il server LDAP, quale directory utilizzare e in quale directory ricercare gli utenti. Un estratto del file (uguale per tutti i client) di configurazione:

  /etc/pam_ldap

  ```
  ## LDAP Server. Must be resolvable without using LDAP.
  host         10.0.42.5

  ## Distinguished name of the search base.
  base         dc=virtuacluster,dc=cs,dc=unibo,dc=it

  ## LDAP protocol version to use
  ldap_version 3

  ## Credentials to bind with. Since this is a client
  ## we don't need to bind as the admin user,
  ## we just bind as anonymous.
  #bindpw      none
  ```

  Come si può notare è sufficiente dichiarare l'host, la directory base su cui effettuare le ricerche, la versione di LDAP e le credenziali (in questo caso il campo è lasciato commentato, dato che i client non hanno bisogno di autenticarsi).

- La configurazione del file /etc/nsswitch.conf, questo file è presente in tutti i sistemi basati su UNIX ed è il file di configurazione per il servizio "Name Service Switch", un servizio che viene utilizzato dal sistema operativo per la risoluzione dei nomi. All'interno di questo file vengono fornite al sistema operativo le istruzioni su come risolvere i nomi. Un esempio di configurazione è il seguente:

  /etc/nsswitch.conf

  ```
  passwd: files  [UNAVAIL=return] ldap
  group:  files  [UNAVAIL=return] ldap
  shadow: files  [UNAVAIL=return] ldap
  ```

  In questo caso, il sistema operativo effettua il lookup dei nomi, dei gruppi e delle password prima sui file locali (questo perché gli utenti root sono tutti locali per garantire l'accesso alla macchina anche in caso di fallimento del server LDAP) e poi sul server LDAP. Nel caso in cui la macchina non riesca a contattare il server LDAP, la risoluzione del nome fallisce. Senza l'istruzione "[UNAVAIL=return]" prima di "ldap", il sistema operativo avrebbe

continuato all'infinito il tentativo di risoluzione del nome, bloccando di fatto la macchina fino alla disponibiltà del server LDAP stesso.

- La configurazione di PAM stesso. PAM deve infatti essere istruito per utilizzare il modulo di autenticazione ldap. Per fare questo è sufficiente modificare i file di configurazione di PAM (che ogni distribuzione fornisce in modo differente) riguardanti il processo di login. L'esempio che segue è tratto dalla configurazione di un client Debian:

  /etc/pam.d/common-account

  ```
  account sufficient  pam_ldap.so
  account required    pam_unix.so
  ```

  In questo caso, la keyword "sufficient" posta dopo la keyword "account" termina il lookup del nome utente qualora il modulo termini correttamente (i.e. "è sufficiente che l'account venga trovato nella directory del server LDAP senza procedere con ulteriori metodi di autenticazione").

In questo modo dunque, avendo i client "predisposti" all'utilizzo di un server LDAP per la risoluzione dei nomi, è necessario procedere con la configurazione del server. In realtà si tratta di un procedimento molto simile a quello effettuato sui client (anche il server comunicherà poi con sé stesso per la risoluzione dei nomi), con in aggiunta l'installazione del server LDAP, che è specifica a seconda della distribuzione utilizzata. Su Debian, ad esempio, è sufficiente installare i pacchetto "slapd" ("OpenLDAP server") e "ldap-utils" e procedere normalmente con la creazione delle directory attraverso le utility messe a disposizione dall'ultimo pacchetto per poter "parlare" con *slapd*. L'esempio che segue è un processo di creazione di un utente all'interno della directory degli utenti:

```
dn: uid=utente,ou=people,dc=virtuacluster,dc=cs,dc=unibo,dc=it
objectClass: inetOrgPerson
objectClass: posixAccount
objectClass: shadowAccount
uid: utente
shadowFlag: 0
shadowWarning: 7
shadowMin: 8
shadowMax: 99999
sn: Utente Creato
givenName: Utente
uidNumber: 1000
gidNumber: 1000
loginshell: /bin/bash
homedirectory: /home/utente
userpassword: passwordbanale
cn: Utente Creato
```

Questo esempio, salvato in un file e passato come parametro all'utility "ldapadd" crea l'utente "utente" nella directory "virtuacluster.cs.unibo.it", all'interno dell'unità organizzativa "people" con password (non cifrata) "passwordbanale". Per cambiare successivamente la password e salvarla sul database in modo cifrato si fa utilizzo dell'utilty "ldappasswd".

3.1.3 Le "home" e il server NFS

Uno dei primi problemi che ci si pone durante la realizzazione di un cluster completamente cooperativo, è quello di garantire agli utenti che - qualora una determinata macchina non sia disponibile - si possa comunque lavorare utilizzando un'altra macchina.

Grazie ad LDAP è stato risolto il problema della gestione degli utenti: una sola entry per utente nella directory permette all'utente stesso di potersi collegare a qualsiasi macchina, mentre grazie ad NFS ("Network File System") viene risolto il problema delle directory home degli utenti.

NFS è un protocollo, originalmente sviluppato dalla Sun Microsystems nel 1984 che permette ad un utente di accedere a files tramite la rete in un metodo simile a quello con cui accede ai files locali. NFS è un open standard: permette quindi a chiunque di implementare il protocollo [4]. NFS fornisce una trasparenza sia a livello di rete che a livello di filesystem, i files di un utente possono risiedere sia su un hard disk della stessa macchina che su una macchina distante di una rete remota: dal punto di vista dell'utente — e della macchina — è perfettamente uguale. Vengono mantenuti i permessi originali: questo significa che le informazioni sui gruppi e sugli utenti devono essere "consistenti" sulle macchine che utilizzeranno il server NFS. [4]

Sulla macchina "vincent" è quindi installato un server in grado di comunicare tramite il protocollo NFS, mentre sulle restanti macchine è installato un client in grado di comunicare con il server. La configurazione del server è banale, è sufficiente aggiungere la riga

```
/home    10.0.42.0/24(rw)
```

al file /etc/exports per condividere tutta la gerarchia delle home locali con tutte le macchine appartenenti alla rete 10.0.42.0-255. Grazie ad LDAP viene anche risolto il problema degli uid degli utenti. È sufficiente assegnare la proprietà della cartella home all'uid che compare nella directory LDAP.

Se non si fosse utilizzato LDAP e si fossero utilizzati i file locali su di ogni macchina, si sarebbe potuta creare un'importante falla di sicurezza sulle home stesse, basta immaginare infatti la situazione seguente: l'utente "alice" è registrata con uid 1000 sulla macchina "vincent" mentre sulla macchina "butch" compare con uid 1001; l'utente "bob" è registrato con uid 1000 sulla macchina "butch" mentre compare con uid 1001 sulla macchina "vincent". Quando "alice" effettua il login su "vincent" è l'effettiva proprietaria della sua home, mentre se effettua il login su "butch" si trova a non essere proprietaria della sua home ma di quella di "bob"; viceversa per quest'ultimo.

Nelle macchine client è sufficiente aggiungere una riga di configurazione nel file /etc/fstabs per permettere l'automount delle home in fase di boot:

```
10.0.42.5:/home    /home    nfs    defaults,auto    0    0
```

Il primo parametro è il filesystem, in questo caso viene anteposto l'indirizzo IP della macchina server ("vincent") al path NFS da utilizzare (/home); il secondo parametro è il punto di mount da utilizzare; il quarto parametro il tipo di filesystem utilizzato; il quinto parametro sono le opzioni mentre il sesto ed il

sesto parametro istruiscono il sistema operativo di non effettuare alcun backup del punto di mount (in realtà, questo parametro è obsoleto[9]) e di non effettuare alcun controllo d'integrità sul filesystem.

Per facilitare un'eventuale operazione di "trasferimento dati" dal server alle altre macchine, le home risiedono all'interno di un file immagine a parte, e viene montato come secondo disco dalla macchina virtuale "vincent". Così facendo, qualsiasi altra macchina potrebbe servire da server NFS e non ci sarebbero perdite di dati.

3.1.4 Il server DHCP

Il server DHCP è il primo servizio che un qualsiasi client proveniente dal cluster, dall'applicazione web o dall'esterno, incontra. Il suo scopo è quello di assegnare in modo dinamico gli indirizzi IP dei client, delegando al server la completa responsabilità e gestione del pool di indirizzi. DHCP ("Dynamic Host Configuration Protocol") è un protocollo divenuto standard nel 1993 ed esistono diversi server che lo implementano. All'interno del cluster viene utilizzato il server "dhcp3" e la sua configurazione risiede in un unico file: /etc/dhcp3/dhcpd.conf, di seguito ne vengono riportati i punti principali.

/etc/dhcp3/dhcpd.conf

```
option domain-name "virtuacluster.cs.unibo.it";
option domain-name-servers 10.0.42.5;

subnet 10.0.42.0 netmask 255.255.255.0 {
    range 10.0.42.200 10.0.42.250;
}

host butch {
    hardware ethernet 52:54:00:DD:EE:33;
    fixed-address 10.0.42.20;
}

host mia {
    hardware ethernet 52:54:00:DD:EE:00;
    fixed-address 10.0.42.10;
}
```

Le prime due righe istruiscono i client su quali siano i DNS da utilizzare, assieme al nome del dominio, in questo caso il server DNS principale che viene utilizzato è "vincent", e il nome del dominio è "virtuacluster.cs.unibo.it".

Il blocco seguente è relativo alla subnet all'interno della quale il server DHCP deve gestire gli indirizzi IP, la rete è quella del cluster virtuale (10.0.42.0/24) e gli indirizzi IP assegnabili dinamicamente partono da 10.0.42.200 e terminano al 10.0.42.250.

I due blocchi che seguono sono configurazioni statiche per due delle dieci macchine che appartengono al core (gli altri otto blocchi sono stati omessi nel

[9] man mount(8)

documento perché uguali a questi). Si istruisce il server DHCP ad assegnare indirizzi IP statici, nella fattispecie 10.0.42.20 alla macchina "butch" e 10.0.42.10 alla macchina "mia" in base al MacAddress[10] dell'interfaccia di rete che richiede la configurazione.

In questo modo, nelle macchine client è sufficiente impostare l'interfaccia richiesta in modalità DHCP e sarà il server l'unico responsabile dell'assegnazione e configurazione degli indirizzi IP. Di seguito viene riportato il file /etc/network/interfaces della macchina "butch", questo file viene letto in fase di boot dal servizio "networking" e serve per istruire il sistema operativo su come configurare le interfacce di rete.

/etc/network/interfaces

```
auto lo
auto eth0

iface lo inet loopback

iface eth0 inet dhcp
```

Come si può notare, l'interfaccia principale ("eth0") è configurata in DHCP[11].

3.1.5 Il server DNS

Con DNS ci si riferisce al "Domain Name System", ovvero il sistema utilizzato in Internet e nelle reti locali per poter tradurre un nome facilmente memorizzabile da un essere umano ("google.com", ad esempio) in un indirizzo IP. Si può pensare al servizio di DNS come ad una sorta di elenco telefonico[12], dove ad ogni nome corrisponde un dato indirizzo IP. [6] Internamente al cluster virtuale, il server DNS serve *solo ed unicamente* per poter risolvere i nomi delle macchine con i corrispondenti indirizzi IP. Ogni server DNS è responsabile di una determinata zona ("virtuacluster.cs.unibo.it", ad esempio), qualora venga fatta richiesta per una zona non di competenza del server, quest'ultimo può essere configurato in modo da avanzare la richiesta ad ulteriori server DNS fino alla risoluzione del nome. Grazie a questo comportamento, spesso ci si riferisce al servizio di DNS come ad un "servizio gerarchicamente distribuito".

Il suo funzionamento è quello di ricevere delle richieste dalle macchine client sotto forma di nome della macchina (e.g. il client chiede al server a quale indirizzo IP corrisponda un determinato nome) e di rispondere con l'indirizzo IP equivalente, qualora questo fosse presente nella lista.

Il server utilizzato è "bind9" a causa del fatto che lo stesso server esiste sia per GNU/Linux che per FreeBSD. La configurazione lato client consiste di due

[10] Il MacAddress è un identificativo univoco dell'hardware di rete

[11] Il file di configurazione /etc/network/interfaces è specifico di Debian e Ubuntu. Ogni distribuzione fornisce il proprio file, l'importante è che la scheda sia configurata in modo tale da non usare un indirizzo IP statico

[12] Storicamente, agli albori delle prime reti, non esistevano i DNS ed ogni macchina manteneva un file (HOSTS.TXT) che associava un nome ad un indirizzo IP. Questo file esiste tutt'ora in molti sistemi operativi anche se ormai contiene solo la voce "localhost 127.0.0.1"

file principali: il primo serve per istruire il server su quali siano le "zone" da esso servite, mentre il secondo è l'"elenco telefonico" della zona servita.

/etc/bind/named.conf

```
zone "virtuacluster.cs.unibo.it" {
    type master;
    file "/etc/bind/master/virtuacluster"
};
```

Questo è l'unico blocco di "zona" servita dal server DNS. Come già spiegato in precedenza, istruisce il server ad effettuare il lookup dei nomi nel file /etc/bind/master/virtuacluster.

/etc/bind/master/virtuacluster

```
@               IN    A    10.0.42.5
@               IN    NS   vincent.virtuacluster.cs.unibo.it.
@               IN    NS   dns.virtuacluster.cs.unibo.it.

; nameservers
vincent         IN    A    10.0.42.5
dns             IN    A    10.0.42.5

; usefull names
ldap            IN    A    10.0.42.5
nfs             IN    A    10.0.42.5

; clients
mia             IN    A    10.0.42.10
marsellus       IN    A    10.0.42.15
butch           IN    A    10.0.42.20
jules           IN    A    10.0.42.25
```

La porzione del file di configurazione mostrata sopra è l'"elenco telefonico", nella fattispecie viene dichiarato che:

- I Name Servers utilizzati sono "vincent.virtuacluster.cs.unibo.it" e "dns.virtuacluster.cs.unibo.it", entrambi corrispondenti all'indirizzo IP 10.0.42.5. Come si può notare, è perfettamente legittimo collegare più nomi allo stesso indirizzo IP. Non è invece possibile fare il contrario, per ovvie ragioni di conflitto.
- Allo stesso indirizzo IP di "vincent" e "dns" corrispondono anche i nomi "ldap" ed "nfs", questo per facilitare la gestione dei servizi stessi. Nel caso in cui si volesse spostare un determinato servizio da una macchina all'altra sarebbe sufficiente infatti cambiare l'indirizzo IP nella tabella dei nomi senza dover riconfigurare ogni client (che continuerebbero a cercare "nfs.virtuacluster.cs.unibo.it" senza essere a conoscenza dello spostamento del servizio da una macchina all'altra)
- Ad ogni client è assegnato un nome ed un indirizzo IP. La configurazione del server deve essere ovviamente coerente con la configurazione del servizio di DHCP, altrimenti ad un determinato nome potrebbe rispondere un'altra macchina.

Grazie al servizio DHCP, sui client non è necessaria alcuna configurazione, vengono già istruiti dal server sia ad utilizzare "vincent" per risolvere i nomi che ad appartenere alla zona "virtuacluster.cs.unibo.it". Il nome completo di una macchina configurata tramite DHCP difatti è "<nomemacchina>.virtuacluster.cs.unibo.it".

Con questa sezione si conclude l'introduzione dei servizi principali offerti da **ShockVM Cluster**, i servizi restanti (server web, dbms, ...) non hanno avuto bisogno di configurazioni particolari per poter funzionare correttamente all'interno del cluster e sono tutti perfettamente funzionanti "out of the box".

3.1.6 Crediti

La realizzazione di **ShockVM Cluster** non sarebbe mai stata possibile senza i seguenti software:

- **QEMU**, un emulatore di CPU scritto da Fabrice Bellard e attualmente rilasciato con licenza "GPL v2". Qemu permette di poter emulare le seguenti piattaforme: x86, x86_64, PowerPC, ARM e SPARC. All'interno del cluster virtuale sono state utilizzate tutte le emulazioni possibili senza mai incorrere in problemi seri. Qemu è inoltre in grado di emulare una quantità enorme di periferiche, fra le quali: schede di rete, cd-rom/dvd, floppy disk, schede video, supporti PCI e ISA per dischi virtuali, usb, .. Qemu è disponibile liberamente al sito web http://qemu.org
- **KVM**, "Kernel-based Virtual Machine" è un progetto atto a creare un'infrastruttura di virtualizzazione per il kernel di Linux. Di fatto non è una vera e propria macchina virtuale, ma crea un device (/dev/kvm) che viene poi utilizzato dagli emulatori stessi per poter effettuare la "virtualizzazione nativa"[13]. KVM viene utilizzato per le macchine con architettura x86 e x86_64 del cluster virtuale e per tutte le macchine create dall'applicazione web. Il sito di riferimento per KVM è http://linux-kvm.org
- **VDE**, "Virtual Distributed Ethernet" è l'"emulatore" (creato dal team VirtualSquare[14]) di rete che funge da collante per tutte le macchine virtuali. Come spiegato in precedenza, le macchine virtuali sono in esecuzione su macchine fisiche differenti per non gravare troppo su una singola macchina. Grazie a VDE le macchine virtuali sono in grado di comunicare fra di loro, emulando di fatto l'intera rete. VDE è distribuito con licenza GPL v3 al sito http://vde.sourceforge.net

[13] Ovvero avere una virtualizzazione non più basata solo sul software ma anche sull'hardware, aumentando notevolmente le prestazioni del sistema virtualizzato

[14] Del quale fan parte molti ex-studenti di questo stesso corso di laurea

3.2 ShockVM Live

ShockVM Live è un'applicazione web il cui scopo è quello di permettere agli utenti la creazione, gestione e condivisione di macchine virtuali. Collabora in modo diretto con *ShockVM Cluster*: ne sfrutta i servizi principali al fine di ottenere un cluster dinamico di macchine virtuali. Ci si può riferire a Live ShockVM come ad un "social network per macchine virtuali", nel senso che gli utenti hanno la possibilità di collaborare al fine di creare nuove macchine.

In questa sezione verranno discussi gli aspetti tecnici e le scelte stilistiche effettuate in fase di progettazione che hanno portato al risultato ottenuto. Prima di procedere con il progetto vero è proprio, è necessario introdurre il framework utilizzato al fine di rendere la comprensione più semplice.

3.2.1 Google Web Toolkit

Google Web Toolkit (GWT) è un toolkit — sviluppato da Google Inc. — di sviluppo per creare ed ottimizzare complesse applicazioni "browser-based"[15]. Molto spesso, nella progettazione di un sito web, si incorre in problemi di "compatibilità" fra browser, per colpa dei quali bisogna sempre adottare due o più soluzioni: una valida per un browser ma non per un altro ed una valida per il secondo browser ma non per il primo. Con GWT, lo sviluppatore delega tutti questi problemi al framework che - lato client - traduce il codice Java con un codice Javascript equivalente risolvendo tutti i cosiddetti "browser quirks"[16].

GWT è diviso principalmente in due parti: il lato server ed il lato client. Lato client, lo sviluppatore ha a disposizione circa una cinquantina di classi, suddivise in elementi visibili ("Widgets"), elementi d'impaginazione ("Layout"), strumenti di temporizzazione ("Timers"), strumenti di parsing del testo ("RegEx"), strumenti per la manipolazione di nodi XML ("XMLReader/Writer"), un intero set di classi per poter eseguire semplici animazioni ("AnimationsFramework") ed altri strumenti utili per il debug dell'applicazione (logging, principalmente). Per poter gestire i diversi stati dell'applicazione, GWT fa uso di un elemento "IFrame" contenente diversi "tokens", il cambio di un token all'interno dell'IFrame genera un evento che rende possibile il cambiare determinati elementi della pagina visualizzata senza la necessità di ricaricare tutta la pagina (ciò che è possibile fare normalmente utilizzando AJAX[17]). Il codice Java viene "tradotto" in codice Javascript equivalente e diviso in circa una cinquantina di file differenti, ogni file viene rinominato sfruttando un meccanismo di hashing basato sui nomi delle funzioni che contiene. Agendo in questo modo, il lookup delle funzioni risulta molto velocizzato (bisogna considerare che una funzione scritta in Java può essere divisa in circa una trentina di funzioni Javascipt equivalenti) e si ottengono prestazioni ottimali, molto simili a quelle che si otterrebbero scrivendo il codice Javascript direttamente.

Per quanto concerne il lato server, lo sviluppatore ha a disposizione tutte le classi disponibili dal server su cui vengono eseguite le servlet (normalmente si tratta di più del 90% dell'SDK di Java) più una classe aggiuntiva ("gwt-servlet") che estende le servlet classiche presenti in Java, rendendole molto più "naturali"

[15] http://code.google.com/webtoolkit
[16] Con questo termine ci si riferisce ai problemi di compatibilità fra browser
[17] Asynchronous Javascript and XML

da usare all'interno dell'applicazione Web. Utilizzando le servlet classiche, infatti, lo sviluppatore può solamente inviare segmenti di testo dal server al client e viceversa, rendendo obbligatori meccanismi di parsing per poter ricostruire un eventuale oggetto; utilizzando gwt-servlet è possibile scambiare interi oggetti: il tutto avviene in modo trasparente tramite un uso molto accurato del formato JSON[18] da parte del framework.

Il risultato finale che si ottiene nell'utilizzare GWT in vece di un mix di altri linguaggi (solitamente Javascript, HTML e CSS lato client e uno a scelta fra Python, PHP, Ruby, ... lato server) è che scrivere un'applicazione web complessa diventa molto più simile a scrivere un'applicazione desktop utilizzando un qualsiasi framework grafico. Fondamentalmente c'è poca differenza: se è vero che per produrre un'applicazione grafica per Desktop si deve completamente separare il concetto di "visualizzazione" dal concetto di "logica" del programma costruendo quindi un'interfaccia grafica completamente indipendente dalla logica (e viceversa), produrre un'applicazione web utilizzando GWT significa esattamente la stessa cosa: la parte "logica" sarà quella che verrà eseguita sul server, mentre la parte "grafica" quella che si costruirà lato client.

Oltre a questo, GWT fornisce anche una potente applicazione di debugging in grado di eseguire un server web minimale ("Jetty"), rendendo persino possibile il debug dell'applicazione lato client direttamente dal codice Java.

Nei paragrafi che seguono verranno presentati gli oggetti principali utilizzati lato client e quelli lato server.

3.2.2 XML

All'interno dell'applicazione si fa un uso costante di file in formato XML.
XML ("eXtensible Markup Language") è un linguaggio formato da un insieme di regole per codificare dati in una forma che sia "friendly" sia per l'essere umano che per la macchina. Il suo design ne enfatizza la semplicità, l'utilizzo generico e l'usabilità attraverso Internet. XML ha la capacità intrinseca di mantenere intatte le informazioni, le strutture e le "forme" dei dati, questo avviene tramite simboli inclusi nel testo, chiamati appunto simboli di "markup". In un certo senso, il markup permette di migliorare il senso stesso del dato, identificandone le parti fondamentali e permettendo a queste di essere "collegate" in modo logico fra loro. Il vantaggio principale di XML è che è facilmente comprensibile da una macchina, tramite appositi processi chiamati "parsers" [8].

Si tratta di un formato dati testuale formato da una gerarchia ad albero, ovvero è necessario che ci sia un nodo, detto "radice" che al suo interno può contenere uno o più nodi, detti "figli" i quali a loro volta possono contenere ulteriori nodi, fino a giungere ai nodi terminali detti "foglie". Un figlio può avere un solo padre, mentre un padre può avere un insieme infinito di figli o foglie.

Nel corso degli anni sono stati sviluppati dei motori di parsing XML per praticamente qualsiasi linguaggio, e ad oggi risulta essere il formato più utilizzato per il passaggio d'informazioni fra macchine, basti pensare che "HTML" è un

[18] JavaScript Object Notation, un formato d'interscambio dei dati basato su dizionario

"dialetto" di XML per capire quanto sia importante l'utilizzo di questo formato.

In **ShockVM Live** XML viene utilizzato per qualsiasi cosa abbia bisogno di essere memorizzata: dagli utenti alle macchine virtuali passando per le immagini CD-ROM disponibili; la scelta di XML al posto di un qualsiasi DBMS[19] è dovuta al fatto che si vuole facilitare la gestione dei dati salvati da parte degli amministratori: per modificare un file XML è sufficiente un editor di testo, mentre per modificare dei dati presenti in un database sono necessari degli strumenti più o meno sofisticati ed una conoscenza del linguaggio del database stesso. In termini prestazionali, la scelta non comporta alcuna perdita (o alcun guadagno) di prestazioni rispetto ad un DBMS, si tratta infatti di file poco complessi e il "parsing" non risulta mai essere particolarmente oneroso.

3.2.3 Configurazione del framework

Come ogni applicazione, anche un'applicazione web necessita di un "entry-point", ovvero una funzione dalla quale l'intero programma deve cominciare la sua esecuzione. Affinché la traduzione del codice avvenga in modo corretto, è necessario specificare un file di configurazione, che verrà letto da GWT ad ogni "traduzione". All'interno di questo file si definiscono le classi utilizzate dall'applicazione, il tema scelto e il nome del modulo compilato. Quello che segue è il file di configurazione utilizzato da Live ShockVM[20].

FlozConfigurator.gwt.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<module rename-to='flozconfigurator'>
  <inherits name='com.google.gwt.user.User'/>

  <inherits name='com.google.gwt.user.theme.standard.Standard'/>
  <entry-point class='it.unibo.cs.v2.client.FlozConfigurator'/>

  <source path='servlets'/>
  <source path='client'/>
  <source path='shared'/>

</module>
```

Come si può notare, l'intero modulo viene "tradotto" all'interno della cartella "flozconfigurator"; si fa uso dell'intera gerarchia delle classi com.google.gwt.user.User; si utilizza il tema standard di GWT; l'entry-point è situato nella classe FlozConfigurator all'interno del pacchetto it.unibo.cs.v2.client e le cartelle servlets, client e shared contengono codice che dev'essere tradotto in codice Javascript equivalente.

3.2.4 Gestione degli utenti

La prima operazione che dev'essere eseguita da chi vuole utilizzare l'applicazione, consiste nella creazione dell'utente: è necessario scegliere un nome utente

[19] DataBase Management System

[20] All'interno del codice viene spesso utilizzato il nome "FlozConfigurator", questo perché il progetto originariamente si sarebbe dovuto chiamare così

univoco, inserire il proprio nome e cognome e la password che si vuole utilizzare per effettuare il login. Lato server, gli utenti sono memorizzati all'interno di un file XML situato in una cartella "protetta" tramite adeguate regole di configurazione del server stesso; la struttura del file consiste in un elemento radice ("FlozConfigurator"), contenente a sua volta elementi "user" che a loro volta contengono elementi riguardanti le informazioni dell'utente stesso.

users.xml

```
<?xml version="1.0" encoding="UTF-8"?>

<FlozConfigurator version="1.0">
  <user displayname="Alfred Hitchcock">
    <login role="user">alfred_h</login>
    <password encrypt="BCrypt">
        $2a$10$obMN .... [cut]
    </password>
    <home>/srv/projects/LiveVC/war/users/alfred_h</home>
  </user>
</FlozConfigurator>
```

Nell'esempio sopra, l'utente "Alfred Hitchcock" identificato tramite il nome utente "alfred_h" ha accesso all'applicazione con il ruolo di "user", la sua password è salvata in modo cifrato utilizzando l'algoritmo "BCrypt" e il path assoluto della sua home è "/srv/projects/LiveVC/war/users/alfred_h".

Durante la creazione di un nuovo utente, questo file viene "navigato" per verificare che l'utente abbia effettivamente scelto un nome utente univoco. Il codice responsabile di questo controllo è il seguente:

```
try {
    DocumentBuilder db = DocumentBuilderFactory.newInstance().
        newDocumentBuilder();
    Document users = db.parse(usersFile);
    NodeList usersNodeLst = users.getElementsByTagName("user");

    for (int i = 0; i < usersNodeLst.getLength(); i++) {
        Node n = usersNodeLst.item(i);

        if (!(n.getNodeType() == Node.ELEMENT_NODE))
            continue;

        Element user = (Element) n;
        Element login = (Element) user.getElementsByTagName("login").
            item(0);

        if (login.getTextContent().equals(username))
            return false;
    }

    return true;

} catch (Exception e) {
```

```
    return false;
}
```

Viene utilizzato un parser XML per aprire il file "usersFile", in seguito ne vengono estratti tutti gli elementi il cui tag è "user" e inseriti all'interno di un oggetto di tipo NodeList; da quest'oggetto, in modo iterativo, si analizzano i singoli nodi e di ogni nodo viene controllato il tag "login"; se il contenuto di questo tag è effettivamente identico al nome scelto dall'utente, l'intera esecuzione termina ritornando il valore "false". Se l'iterazione invece termina correttamente, allora viene ritornato il valore "true". A seconda di questo valore, il server procede o meno con l'inserimento del nuovo utente all'interno del file XML e la creazione della sua directory home.

L'algoritmo scelto per salvare in modo sicuro le password è "BCrypt", un algoritmo che sfrutta la cifratura "Eksblowfish" ("expensive key schedule Blowfish"): il setup della chiave comincia con una forma modificata del classico Blowfish, durante il quale sia la chiave che il "sale"[21] vengono utilizzati per impostare tutte le sotto-chiavi; da qui viene utilizzato per un numero costante c[22] di volte l'algoritmo standard Blowfish alternando in egual modo sia la chiave che il sale, ovvero utilizzando la chiave per cifrare il sale e il sale per cifrare la chiave in modo da ottenere in tutto c^2 fasi. La scelta della costante è fondamentale: se troppo alta si rischia di saturare la CPU, mentre se troppo bassa si è più esposti ad attacchi di tipo "bruteforce"[23]: *in medium stat virtus* [9].

All'interno del progetto ne viene utilizzata l'implementazione in Java, disponibile con licenza "ISC/BSD" e scritta da Damien Miller. Al momento della stesura del documento (Ottobre 2011), "jBCrypt" risulta essere l'algoritmo di cifratura migliore disponibile per questo linguaggio. Dopo accurate ricerche, le altre soluzioni proposte prevedevano cifrature "unsalted" o algoritmi basati sulla rotazione dell'alfabeto ("cifrari di Cesare" o "Rot13"), confermando quindi quanto scritto anche in [7] già nel 2008.

Una volta registrato all'interno del file, l'utente può effettuare il login utilizzando il nome utente scelto e la password.

3.2.5 Creazione di una macchina virtuale

Lato server, la configurazione di una singola macchina virtuale viene salvata in un file XML all'interno della home dell'utente. Ogni macchina può essere configurata in diversi modi, per questo si rende necessario un meccanismo per poter facilmente recuperare le opzioni richieste in fase di creazione. Il file XML è caratterizzato dalla seguente struttura:

SomeVirtualMachine.xml

```
<?xml version="1.0" encoding="UTF-8"?>
```

[21] Il "sale" è un'ulteriore chiave che viene aggiunta alla chiave originale per poter prevenire gli attacchi a dizionario. Due chiavi uguali quindi non saranno mai cifrate nello stesso modo poiché avranno un "sale" differente

[22] La costante viene configurata dal programmatore

[23] Tipologia di attacco "ignorante", senza alcun metodo scientifico vengono provate tutte le combinazioni possibili

```
<VirtualMachine>
  <name>SomeVirtualMachine</name>
  <description>A short one-line description</description>
  <iso path="isos/archlinux.iso"/>
  <realowner>massi</realowner>
  <hda enabled="true" path="SomeVirtualMachine.img" size="2GB"/>
  <hdb enabled="false" path="false" size=""/>
  <virtuacluster enabled="true"/>
  <secondaryNetwork enabled="false" macaddress="" socket=""/>
</VirtualMachine>
```

Il nodo radice "VirtualMachine" contiene tutti gli altri nodi, che mostrano come la macchina sia stata configurata. In questo caso l'utente "massi" è il legittimo proprietario della macchina virtuale "SomeVirtualMachine", che utilizza il CD-ROM "archlinux.iso" ed un singolo disco virtuale di 2GB chiamato "SomeVirtualMachine.img"; la macchina è collegata al socket del cluster virtuale e non presenta né una seconda immagine disco né un socket VDE privato.
La creazione del file XML avviene sfruttando le API per il parsing e la modifica di file XML messe a disposizione del linguaggio stesso (all'interno del progetto qualsiasi modifica ad un file XML avviene utilizzando le adeguate API, per garantire che il file sia sempre valido), mentre la creazione del disco virtuale avviene eseguendo in background il processo "qemu-img", messo a disposizione da parte dei software "Qemu" e "KVM", come mostrato di seguito:

```
try {
    qemuImg = new ProcessBuilder("qemu-img", "create",
        hdaFileName, machineInfo.getHdaSize());
    qemuImg.directory(new File(userHome));
    Process p = qemuImg.start();
    p.waitFor();

    if (machineInfo.isHdbEnabled()) {
        qemuImg = new ProcessBuilder("qemu-img", "create",
            hdbFileName, machineInfo.getHdbSize());
        qemuImg.directory(new File(userHome));
        Process p2 = qemuImg.start();
        p2.waitFor();
    }
} catch (IOException e) {
    logger.log(e.getMessage());
}
```

Il file immagine per il disco primario ("hdaFileName") porta lo stesso nome della macchina virtuale, mentre il disco secondario presenta il suffisso "-secondary". L'esecuzione del processo avviene sfruttando le API del linguaggio Java, messe a disposizione dal package "java.io", il processo viene eseguito all'interno della home dell'utente e in caso di eccezione, anziché terminare l'esecuzione dell'intero processo di creazione, viene notificato l'amministratore dell'avvenuto errore tramite l'oggetto logger, una semplice classe che scrive in un file apposito tutti gli errori avvenuti. Questo perché l'esecuzione del processo può fallire solamente in due casi:

1. Qemu (o KVM) non sono stati installati, in questo caso l'amministratore procede installando i pacchetti necessari;
2. lo spazio su disco è insufficiente, in questo caso l'amministratore comunica agli utenti che non è più possibile creare nuove macchine virtuali.

Un altro parametro interessante è costituito dall'immagine CD-ROM che l'utente vuole utilizzare. La lista delle immagini disponibili presentata all'utente è infatti ottenuta grazie ad una servlet, responsabile di effettuare il parsing di un file XML contenente le informazioni fondamentali su ogni immagine presente, il file è situato all'interno della cartella "isos" e chiamato "imagesList.xml". La struttura del file è la seguente:

imagesList.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<ImagesList>

  <Image>
    <name>ArchLinux</name>
    <version>2010.05 netinstall</version>
    <description>
      Arch Linux is an independently-developed i686/x86-64 community
      distribution, based on a rolling-release model and targeted at
      competent GNU/Linux users which offers large binary repositories
      and full-featured package management as well as a ports-like
      packaging system. Development focuses on a balance of minimalism,
      elegance, code correctness and modernity. Version 0.1 (Homer)
      was released March 11, 2002.
    </description>
    <web>http://www.archlinux.org</web>
    <iso path="isos/archlinux.iso"/>
  </Image>

</ImagesList>
```

Ogni immagine è salvata all'interno del nodo "Image", il quale a sua volta contiene informazioni sul nome della distribuzione, la sua versione, una breve descrizione, il sito web di riferimento e la locazione relativa dell'immagine stessa. Aggiungere un'immagine a disposizione dell'utente è quindi un processo che consta di due fasi: ottenere l'immagine ed aggiungerne le informazioni all'interno del file XML.

In fase di creazione del file, il server controlla anche che non si stia per generare un conflitto fra nomi, ovvero che non ci siano già altre macchine virtuali (all'interno della home dell'utente) con lo stesso nome. I nomi delle macchine virtuali non possono contenere spazi o caratteri non alfanumerici.

3.2.6 Visualizzazione di una macchina

Il processo di esecuzione e visualizzazione di una macchina virtuale è probabilmente il processo più oneroso per il server. Al momento della richiesta, il client procede con l'invio al server di un oggetto di tipo "MachineInfo" — contenuto nel package di oggetti condivisi fra il server ed il client — il quale a sua volta contiene informazioni circa la macchina virtuale che si desidera visualizzare.

```
public class MachineInfo implements IsSerializable {
    private String configurationFile;
    private String iso;
    private String name;
    private String description;
    private String hda;
    private String hdaSize;
    private boolean hdbEnabled;
    private String hdb;
    private String hdbSize;
    private boolean virtuacluster;
    private boolean secondNetwork;
    private String macAddress;
    private String socketPath;
    private boolean bootCdrom;
    private String realOwner;
    private boolean userOwner;
    private boolean tabletDevice;
    private String longDescription;
    private LinkedList<String> sharedWith = new LinkedList<String>();
    private LinkedList<String> pendingShares = new LinkedList<String>();

    /* Getters and setters */
    [cut]
```

Il primo passo è quello di controllare che la macchina richiesta non sia già in esecuzione, per fare ciò esiste un file speciale nella home di ogni utente chiamato "ACTIVEMACHINES.txt", questo file tiene traccia delle macchine virtuali in esecuzione, del server VNC utilizzato e del PID del processo; la struttura del file è molto semplice ed è uno dei pochi file a non essere salvato in XML (proprio a causa della sua estrema semplicità): ogni riga corrisponde ad una macchina virtuale ed è divisa a sua volta in tre campi, separati dai caratteri "::", segue un esempio:

ACTIVEMACHINES.txt

```
SimpleVirtualMachine::95::3198
Debian::96::4116
```

In questo caso sono in esecuzione due macchine virtuali: "SimpleVirtualMachine" sul server VNC 95 con PID[24] 3198 e "Debian" sul server VNC 96 con PID 4116. Se la macchina richiesta dall'utente è presente in questo file, il server costruisce un oggetto di tipo "MachineProcessInfo" — uno degli oggetti presenti sia a lato server che a lato client — e lo mette a disposizione del client, in caso contrario procede come segue:

1. effettua la lettura di un file speciale, chiamato ".freeServers" e residente nella root del progetto, per ottenere una lista di server VNC non attualmente utilizzati, qualora non fosse disponibile un server VNC, viene ritornato un errore e l'esecuzione della servlet termina; l'accesso al file di cui sopra è inserito all'interno di un blocco "synchronized ()" di Java, in modo da garantire la mutua esclusività — ovvero due utenti non possono leggere il file in modo contemporaneo;

[24] Process ID

2. viene costruita una stringa costituita dal comando della macchina virtuale assieme a tutti i parametri in base alla configurazione della macchina:

```
String kvmCommand = "kvm -cdrom " + getServletContext().
    getRealPath(machineInfo.getIso());
kvmCommand += " -hda " + machineInfo.getHda();
[cut]
```

   Questa stringa viene poi salvata, assieme al comando "pgrep" all'interno di uno shell script. Il comando "pgrep" è costruito in modo da ottenere il PID della macchina virtuale dell'utente e scriverlo su un file temporaneo presente nella home dell'utente stesso, quindi:

```
String pgrepCommand = "pgrep -nu tomcat6 -fx \"" +
    kvmCommand + "\" > .temp\n\n";
```

3. Viene eseguito il processo "/bin/sh" con parametro il file contenente lo shell script per eseguire la macchina virtuale e — se l'esecuzione dello script termina senza errori — viene scritto il file delle macchine attive;

4. Viene costruito un oggetto di tipo "MachineProcessInfo", contenente tutte le informazioni necessarie al client per visualizzare la macchina virtuale e inviato al client stesso.

Leggermente diverso è il processo di visualizzazione di una macchina virtuale della quale l'utente non ne sia il proprietario reale (ovvero il caso di una macchina condivisa), difatti viene letto il file "ACTIVEMACHINES.txt" del proprietario reale della macchina e — se la macchina è presente — viene costruito un oggetto "MachineProcessInfo" contenente le informazioni riguardanti il server VNC e il PID del processo, qualora invece la macchina non sia attiva, l'utente riceve un messaggio d'errore.

La classe "MachineProcessInfo" menzionata in precedenza è definita come segue:

```
public class MachineProcessInfo implements IsSerializable {
    private int pid;
    private int vncServer;
    private String machineName;
    private boolean owned;

    /* Getters and setters */
    [cut]
```

Come si può immaginare, lo spegnimento di una macchina virtuale consta nel processo contrario: il server è incaricato di terminare il processo corrispondente (se ancora attivo), leggere il file delle macchine attive dell'utente e "liberare" il server VNC ponendo il numero del server in coda al file dei server liberi. Pertanto, non è sufficiente che l'utente spenga la macchina dall'interno della macchina virtuale, è *sempre* necessario liberare anche il server VNC.

3.2.7 Esportazione di una macchina

Dal punto di vista implementativo, l'esportazione di una macchina consta nell'effettuare una copia dell'immagine del disco (o dei dischi) nella cartella destinata a contenere le macchine "pubbliche". La copia del file avviene utilizzando le API Java, di modo che sia più facile poter tener traccia del progresso effettuato. Esiste una funzione statica all'interno della classe "Utils" contenuta nel package del server il cui scopo è quello di effettuare una copia binaria di un file generico:

```
public static void customCopy(File src, File dst, int chunkSize)
    throws Exception {
    dst.createNewFile();

    FileInputStream srcReader = new FileInputStream(src);
    FileOutputStream dstWriter = new FileOutputStream(dst);

    byte[] chunk = new byte[chunkSize];

    int len = 0;
    while ((len = srcReader.read(chunk, 0, chunkSize)) != -1)
        dstWriter.write(chunk, 0, len);

    srcReader.close();
    dstWriter.flush();
    dstWriter.close();
}
```

Il primo parametro è il file sorgente, il secondo il file di destinazione mentre il terzo è un parametro configurabile che indica quanti byte copiare alla volta. Un numero troppo elevato potrebbe portare ad una rapida saturazione dello heap di Java, mentre un numero troppo basso ad una copia troppo lenta.

L'esportazione di una macchina virtuale — lato server — comporta la creazione di due nuovi thread: il primo effettua la copia dei file, mentre il secondo è responsabile di monitorare il progresso effettuato e di notificare l'utente riguardo la percentuale completata. Agendo in questo modo, la copia è completamente asincrona e l'utente può continuare ad utilizzare il sistema mentre il server esegue i lavori in background. Se non si fosse adottata questa soluzione, l'applicazione web si sarebbe completamente bloccata durante tutto il processo.

Per meglio comprendere il funzionamento, viene introdotta la classe "FileMonitor", l'oggetto cioè che è costante del monitoraggio del progresso della copia:

```
public class FileMonitor extends Timer {
    private File srcFile, dstFile;
    private File notificationFile;
    private String message;
    private BufferedWriter bw;

    private TimerTask innerTask = new TimerTask() {

        @Override
```

```
10          public void run() {
                try {
12                  bw = new BufferedWriter(new FileWriter(notificationFile));
                    bw.write(NotificationType.TIMEDJOB.toString());
14                  bw.newLine();

16                  bw.write(message + " " + srcFile.getName());
                    bw.newLine();
18
                    bw.write(new Double(
20                      ((double) dstFile.length() /
                        (double) srcFile.length()) * 100.0)
22                      .toString());
                    bw.newLine();
24
                    bw.flush();
26                  bw.close();
                } catch (Exception e) {
28                  System.out.println(
                        "Exception in FileMonitor: " + e.getMessage());
30              }
            }
32
            @Override
34          public boolean cancel() {
                try {
36                  bw.flush();
                    bw.close();
38              } catch (Exception e) {}

40              notificationFile.delete();
                return super.cancel();
42          }
        };
44

46      public FileMonitor(File srcFile, File dstFile,
            String home, String msg) {
48          this.srcFile = srcFile;
            this.dstFile = dstFile;
50          notificationFile = new File(home + "/notification-" + new Date()
                .getTime());
52          message = msg;
        }
54
        public void start() {
56          scheduleAtFixedRate(innerTask, 1500, 10000);
        }
58
        @Override
60      public void cancel() {
            innerTask.cancel();
62          super.cancel();
        }
```

```
64 }
```

Come si può notare alla linea 7, viene inizialmente definita un'inner class[25] di tipo "TimerTask", questa classe contiene il metodo "run ()" che viene chiamato da Java ad intervalli regolari una volta che l'intero Timer è stato inizializzato, ciò avviene alla riga 56. Alla riga 51 viene creato un file di notifica all'interno della home dell'utente che viene aggiornato ogni 10 secondi dal metodo "run ()" del TimerTask. La creazione dell'oggetto, specificata alla riga 46 prevede il passaggio dei parametri necessari per il corretto funzionamento.

A questo punto, dovrebbe risultare di facile lettura il codice della servlet responsabile di effettuare l'esportazione della macchina virtuale:

```
// Do all the work in another thread
2 new Thread() {
    public void run() {
4       try {
            // Copy the stuff
6           FileMonitor fm = new FileMonitor(hda, hdaDestination,
                home, "Exporting");
8
            fm.start();
10          Utils.customCopy(hda, hdaDestination, ((1024 * 1024) * 10));
            fm.cancel();
12
            cut...
14      }

16      catch (Exception e) {
            // Drop a notification at the user's home.
18          File notification = new File(home + "/notification-"
                + new Date().getTime());
20          cut...
        }
22  };
}.start();
```

(Le sezioni tagliate sono ininfluenti ai fini della comprensione).

Alla riga 7 viene inizializzato un FileMonitor, il cui file sorgente è definito dall'oggetto hda, il file destinazione è hdaDestination e i restanti due parametri sono rispettivamente la home dell'utente e il messaggio utilizzato nella notifica. Alla riga 10 comincia effettivamente la copia dei file. Il thread viene bloccato qui fintanto che la copia non sarà terminata, mentre l'oggetto FileMonitor verrà eseguito da Java ogni 10 secondi. Al termine della copia, l'oggetto FileMonitor viene fermato con la chiamata al metodo "cancel ()". In caso di errore viene mandata una notifica all'utente.

Per evitare possibili collisioni di nome, a tutti i file copiati viene anteposto il prefisso "<nomeutente>-", così facendo si ottengono due vantaggi principali: il primo è la possibilità da parte degli utenti di poter esportare la propria versione

[25] Classe anonima il cui scope è solamente interno alla classe residente

di una macchina virtuale già esportata da qualcun altro; il secondo è la facilità — da parte del sistema — di poter mostrare agli utenti la lista delle macchine esportate ordinate per nome utente.

Il processo d'importazione di una macchina virtuale avviene in modo completamente analogo a quello di esportazione, pertanto una descrizione tecnica accurata risulterebbe ridondante. È importante notare che durante il processo d'importazione, vengono rimossi tutti i prefissi aggiunti durante la fase precedente.

Attenzione: durante l'esportazione vengono rimossi tutti gli elementi di condivisione dalla configurazione della macchina. Questo per garantire che una macchina appena importata sia di fatto equivalente ad una macchina creata *ex-novo*.

3.2.8 Condivisione di una macchina

La condivisione di una macchina virtuale è un processo suddiviso in tre fasi, questo perché l'utente "obiettivo" della condivisione deve prima accettare la condivisione di una macchina virtuale:

1. durante la prima fase, l'utente comunica al server la sua intenzione di condividere una macchina virtuale con un altro utente; il server modifica il file della macchina da condividere aggiungendo il tag "share" con parametro "status" uguale a "pending", in questo modo l'utente richiedente ha sempre a disposizione la lista degli utenti ai quali ha richiesto la condivisione. Il server invia una notifica al destinatario, comunicandogli l'intenzione del mittente di condividere una macchina virtuale, e fornendone una breve descrizione;

2. l'utente che ha ricevuto la notifica può quindi scegliere di accettare o rifiutare la macchina virtuale. Nel caso in cui accetti, il server cambia lo stato dello "share" della macchina virtuale in "accepted" ed effettua una copia del solo file XML della macchina nella home dell'utente destinatario e notifica il mittente; in caso contrario, il mittente viene notificato del rifiuto e lo "share" viene rimosso dal file XML;

3. una volta condivisa la macchina virtuale, vengono aggiornate entrambe le liste delle macchine degli utenti al fine di mostrare la nuova situazione.

L'intero processo è quindi molto semplice. Una volta effettuata la condivisione, l'utente "proprietario" della macchina condivisa continua a mantenerne tutti i privilegi: è l'unico infatti che può avviarla, spegnerla, condividerla, esportarla od eliminarla. Questa gestione dei privilegi avviene grazie al file XML della macchina stessa, contenente un tag ("realOwner") che viene letto dal server ogni qualvolta si richieda una delle operazioni sopra elencate.

A seguire un estratto del codice appartenente alla servlet responsabile di terminare il processo di condivisione di una macchina virtuale — ovvero il destinatario ha accettato la condivisione.

```
synchronized (this) {
    DocumentBuilder db = DocumentBuilderFactory.newInstance().newDocumentBuilder();
    Document xmlDoc = db.parse(remoteMachineFile);

    NodeList shares = xmlDoc.getElementsByTagName("share");
    for (int i = 0; i < shares.getLength(); i++) {
        Element share = (Element) shares.item(i);
        if (share.getAttribute("user").equals(user)) {
            share.setAttribute("status", "accepted");
            break;
        }
    }

    Transformer t = TransformerFactory.newInstance().newTransformer();
    t.setOutputProperty(OutputKeys.INDENT, "yes");
    t.transform(new DOMSource(xmlDoc), new StreamResult(remoteMachineFile));
}
```

Si nota che — utilizzando le API XML di Java — il server scorre iterativamente tutti i nodi "share" dell'XML della macchina virtuale fintanto che non viene trovato quello contenente il nome utente del destinatario che ha accettato la condivisione; questo viene poi modificato, cambiando l'attributo "status" da "pending" ad "accepted" e il file viene successivamente riscritto nella home del mittente.

L'intero processo di modifica del file XML è posto all'interno di un blocco "synchronized ()", questo perché un utente può richiedere la condivisione della macchina virtuale con più utenti e, nell'ipotesi che rispondano tutti contemporaneamente, si creerebbe un conflitto concorrente nell'accesso al file, l'utilizzo del blocco sincronizzato previene l'accesso all'intera servlet fintanto che un processo si trova ancora all'interno di quel blocco.

Quando un utente non proprietario richiede di visualizzare una macchina virtuale condivisa, il server controlla all'interno del file "ACTIVEMACHINES.txt" della home del proprietario, che la macchina sia in esecuzione. In caso contrario, l'utente che desiderava visualizzare la macchina, riceve un messaggio d'errore.

Quello che segue è un estratto del codice che mostra il comportamento del server quando un utente richiede la visualizzazione di una macchina della quale non ne è l'effettivo proprietario.

```
String realOwner = machineInfo.getRealOwner();
if (!realOwner.equals(user)) {
    File otherUserDir = new File(getServletContext().getRealPath("/users/" +
        realOwner));

    if (!otherUserDir.exists())
        throw new Exception(
            "You don't own this machine, and appearently, " +
            "the user that owns it doesn't exist.");

    File otherUserActiveFile = new File(
        getServletContext().getRealPath(
            "users/" + realOwner + "/ACTIVEMACHINES.txt"));

    if (!otherUserActiveFile.exists())
```

```
16          throw new Exception("You don't own this machine and " +
                realOwner + " hasn't started any machine yet.");
18
        else {
20          BufferedReader br = new BufferedReader(
                new FileReader(otherUserActiveFile));
22          String line;
            while ((line = br.readLine()) != null) {
24              if (line.length() < 7)
                    continue;
26
                String[] values = line.split("::");
28              if (values == null)
                    continue;
30
                if (values[0].equals(machineInfo.getName())) {
32                  mp.setPid(new Integer(values[1]));
                    mp.setVncServer(new Integer(values[2]));
34
                    br.close();
36
                    mp.setOwned(false);
38
                    return mp;
40              }
            }
42      }

44      throw new Exception("You don't own this machine and "
            + realOwner + " hasn't started it yet.");
46  }
```

Il processo di lettura dei valori da restituire al richiedente avviene dalla linea 29 in poi, dentro al ramo else. Sia prima che dopo vengono soltanto eseguiti controlli. Essendo l'accesso al file delle macchine attive dell'utente proprietario effettuato in sola lettura, non v'è alcun bisogno d'implementare meccanismi per la gestione di un eventuale accesso concorrente.

3.2.9 Il sistema di notifiche

Il sistema di notifiche si è rivelato essere necessario per il corretto procedimento delle operazioni asincrone che vengono eseguite regolarmente all'interno dell'applicazione. Il suo funzionamento è interamente basato su file su disco. Una notifica è infatti uno speciale file che viene posizionato all'interno della cartella home dell'utente che deve visualizzarla. La ricezione delle notifiche è richiesta dal client al server ad intervalli regolari, tramite un oggetto di tipo "Timer" presente fra le utilità messe a disposizione dal Framework. L'oggetto viene inizializzato una sola volta durante la creazione del pannello principale di sinistra, quella che segue è la sua implementazione:

```
private final Timer notificationsTimer = new Timer() {
2       public void run() {
```

```
        buildNotificationsList();
    }
};

[ cut ... ]

private final void buildNotificationsList() {
    getNotificationsProxy.getNotifications(
        new AsyncCallback<LinkedList<Notification>>() {
        @Override
        public void onFailure(Throwable caught) {
            notificationsPanel.clear();
            notificationsPanel.add(
                new HTML("<b>Failure while getting the notifications :-(</b>"));
            notificationsTimer.cancel();
        }

        @Override
        public void onSuccess(LinkedList<Notification> result) {

            if (result == null)
                notificationsTimer.cancel();

            // if the notifications' list has changed in some way,
            // then refresh the panel
            if (!notifications.equals(result)) {
                int jobs = 0;

                notifications = result;
                notificationsPanel.clear();

                for (final Notification notification : notifications) {
                    notificationsPanel.add(new HTML("<b>New notification from: "
                        + notification.getFrom() + "</b><br />"));
                    switch (notification.getType()) {
                    case TIMEDJOB:
                        final TimedNotification tn = (TimedNotification) notification;
                        notificationsPanel.add(new HTML(tn.getTimedMessage() + "<br />"));
                        Double d = new Double(tn.getPercentage());
                        notificationsPanel.add(
                            new HTML("<progress max=\"100\" value=\"" +
                                d.shortValue() + "\" /><br />"));
                        notificationsPanel.add(
                            new HTML("  <b>" + d.shortValue() +
                                "%</b><br /><br />"));
                        jobs++;
                        break;
                    case ACCEPTEDSHARE:
                    case REFUSEDSHARE:

                        [cut ...]
                        }
                    }

```

```
                setHeaderHTML(
58                  getWidgetCount() - 1,
                    "Notifications (" + (notifications.size() - jobs) + ") " +
60                  "[" + jobs + " running]");
            }
62
            else if (result.size() < 1) {
64              setHeaderHTML(getWidgetCount() - 1, "Notifications (0)");
                notificationsPanel.clear();
66              notificationsPanel.add(new HTML(ZERO_NOTIFICATIONS));
            }
68      }
    });
70  }
```

Parte del codice è stata omessa poiché ridondante. Si può notare che ad ogni chiamata del metodo, viene effettuata la richiesta al server di ricevere una lista di notifiche presenti. In caso di errore viene mostrato un errore, in caso contrario invece si procede con l'analisi dell'oggetto stesso. Se questo oggetto è identico a quello che il client già possiede, allora non è necessario ricostruire l'intera area delle notifiche; qualora non sia così, invece, si procede analizzando ogni singola notifica presente e mostrandola al client. Nel codice visualizzato sopra, è riportato solo il codice appartenente alla visualizzazione di una notifica di tipo "TIMEDJOB".

Il formato di un file di notifica è variabile da notifica a notifica, le uniche parti in comune sono la prima riga — che presenta sempre il tipo di notifica da visualizzare — e la seconda — il mittente della notifica stessa. Esiste una classe astratta che si deve reimplementare qualora si voglia implementare una nuova tipologia di notifiche, la classe è presente nel package degli oggetti condivisi:

```
public abstract class Notification implements IsSerializable {
        protected NotificationType type;
        protected String from;
        protected String message;
        protected String fileName;

        public abstract void setType(NotificationType type);
        public abstract void setMessage(String message);
        public abstract void setFrom(String from);

    [cut ...]

}
```

Nella parte omessa sono presenti i getter per le tre funzioni astratte e sia il getter che il setter per il campo "fileName".

Ogni notifica è un file che deve presentare obbligatoriamente il prefisso "notification-", la parte dopo è completamente irrivelante. Nel progetto viene utilizzato il timestamp per mantenere un ordine cronologico delle notifiche.

Il codice responsabile di costruire — lato server — la lista delle notifiche è il seguente:

```
public LinkedList<Notification> getNotifications() {
    File[] notifications = userDir.listFiles(new FileFilter() {

        @Override
        public boolean accept(File pathname) {
            if (pathname.getName().startsWith("notification-"))
                return true;

            return false;
        }
    });

    LinkedList<Notification> ret = new LinkedList<Notification>();

    for (File notification : notifications) {
        boolean insertFirst = false;
        boolean error = false;

        Notification add = null;
        try {
            BufferedReader notificationReader =
                new BufferedReader(new FileReader(notification));

            // First line is always the notification type
            NotificationType type = Notification.parseNotificationType(
                notificationReader.readLine());
            switch (type) {
            case ACCEPTEDSHARE:
            case REFUSEDSHARE:
                add = new RefuseMachineNotification();
                add.setType(type);
                add.setFrom(notificationReader.readLine());
                ((RefuseMachineNotification) add).setMachineName(
                    notificationReader.readLine());
                break;
            [omissis]

            }

            [omissis]

            ret.addFirst(add);

            [omissis]
        }
    }
}
```

Sono state omesse le parti relative alle costruzioni dei diversi tipi di notifica poiché — come si può immaginare — assomigliano molto a quelli presentati. Osservando il codice si nota che vengono analizzati tutti i file che presentano il suffisso "notification-", ognuno di questi viene letto, parsato ed aggiunto alla lista.

Attenzione: i file che presentano le notifiche vengono cancellati solo quando espressamente richiesto dall'utente (o quando la notifica diventa obsoleta), non al momento della costruzione della lista!

3.2.10 Crediti

Per la realizzazione dell'intero progetto sono state utilizzate differenti tecnologie, senza le quali l'intera realizzazione non sarebbe stata possibile (o per lo meno sarebbe stata molto più difficoltosa). I ringraziamenti per aver messo a disposizione gli strumenti vanno quindi a:

- **Oracle/Sun Microsystems** per il linguaggio di programmazione Java[26], utilizzato ampiamente all'interno di tutto il progetto;
- **Apache Foundation** per il subset di classi appartenenti al package Catalina e per il webserver Tomcat[27], senza i quali non si potrebbero scrivere applicazioni web sfruttando Java;
- **Google Inc.** per l'eccellente framework GWT ("Google Web Toolkit"[28]), che da vita al cuore di tutta l'applicazione web;
- **Joel "Kanaka" Martin**, autore del client VNC ("noVNC"[29]) scritto interamente in HTML5, che permette la visualizzazione delle macchine virtuali sfruttando solamente un browser web;
- **Daniel "Mindrot" Miller**, autore della libreria "jBCrypt"[30], che permette l'hashing di password in Java sfruttando l'algoritmo "BCrypt" basato su "EksBlowfish";

[26] http://oracle.com/java
[27] http://tomcat.apache.com
[28] http://code.google.com/webtoolkit
[29] http://kanaka.github.com/noVNC
[30] http://www.mindrot.org/projects/jBCrypt

4 Conclusioni ed osservazioni

Come già accennato in fase introduttiva, lo scopo del progetto consisteva nella dimostrazione di due tesi differenti:

i → la possibilità di poter costruire un cluster completamente operativo in un ambiente interamente virtuale e formato da macchine appartenenti ad un insieme eterogeneo sia di architetture che di sistemi operativi;

ii → la realizzazione di un'applicazione web che permettesse ai suoi utenti la creazione di macchine virtuali interconnesse al cluster del punto precedente e che fornisse agli utenti stessi un sistema di cooperazione semplice da gestire.

Entrambe le tesi sono state dimostrate con successo, infatti: per quanto concerne il punto primo, il cluster è perfettamente funzionante, i servizi forniti sono "distribuiti" fra macchine di architetture differenti e la collaborazione avviene in modo trasparente in un insieme eterogeneo di sistemi operativi. La rete è anch'essa interamente virtuale e costruita tramite gli strumenti messi a disposizione da VDE. Durante la realizzazione del cluster non si sono presentati particolari problemi né riguardanti l'installazione dei sistemi operativi su architetture non x86 né riguardanti le configurazioni dei vari servizi sia lato client che lato server. Tuttavia è doveroso osservare che fra alcune architetture è presente un sostanziale "gap" di software utilizzabile: PowerPC, ARM e MIPS dispongono di un "parco software" quantitativamente inferiore rispetto alle architetture x86 e x86_64; nonostante ciò, in tutte le architetture sono presenti tutti i pacchetti necessari per procedere alla realizzazione di un cluster. La stessa osservazione è facilmente riportabile anche in materia di sistemi operativi: i sistemi basati su kernel BSD ("Berkley Software Distribution") offrono tendenzialmente meno scelta dei sistemi basati su kernel Linux, probabilmente come conseguenza del fatto che quest'ultimo dispone di un bacino di utenti più vasto; nonostante ciò, nemmeno queste lievi differenze hanno reso difficoltosa la realizzazione del cluster stesso.

Per quanto concerne invece la realizzazione del secondo punto, risulta evidente che le moderne tecnologie in materia di sviluppo web hanno reso possibile la creazione di complesse applicazioni in modo relativamente semplice. L'intero processo di progettazione del sistema, stesura del codice e debugging ha impiegato circa 6 mesi (Maggio – Ottobre) e il risultato è un "social network per macchine virtuali" perfettamente integrato nell'ecosistema del cluster presentato al punto precedente; gli utenti hanno inoltre la massima libertà di gestione delle proprie macchine e la possibilità di collaborare simultaneamente su di una stessa macchina condivisa.

4.1 Possibili sviluppi

Sebbene l'intero sistema allo stato attuale sia perfettamente utilizzabile da chiunque abbia conoscenze minime di reti e sistemi operativi, non risulta particolarmente difficile poter pensare ad evoluzioni ulteriori che possano renderlo più consono a situazioni differenti da quelle sin qui proposte; quelli che seguono sono solo alcuni esempi di possibili evoluzioni e non è escluso che in futuro l'applicazione possa servire a scopi completamente diversi:

→ in ambito didattico si potrebbe utilizzare il sistema per svolgere lezioni interattive: sfruttando il già presente sistema di ruoli si potrebbero definire particolari permessi per i docenti e per gli studenti, in modo che questi ultimi possano solo visualizzare la macchina virtuale su cui opera il docente che esegue dimostrazioni "dal vivo". Un approccio simile potrebbe garantire agli studenti la possibilità di seguire le lezioni anche quando impossibilitati a presenziare in aula;

→ ancora in ambito didattico, durante lo svolgimento dei progetti previsti dai vari corsi, i docenti potrebbero seguire i progressi dei gruppi di studenti in modo più diretto, mettendo magari a disposizione una macchina "prefabbricata" su cui gli studenti possano effettuare delle prove;

→ l'intero sistema può essere adoperato per poter effettuare prove di sistemi distribuiti, simulando anche i "crash" delle macchine per poter verificare l'effettiva stabilità del sistema che si sta progettando;

In generale, l'applicazione può servire ad un numero indefinito di scopi e — sebbene lo stato attuale del progetto sia esauriente ai fini della tesi — le sue potenzialità sono ancora ben lontane dall'essere state espresse totalmente. In futuro, dunque, l'evoluzione dell'intero sistema proseguirà fino a poter coprire più aree possibili (didattica, ricerca, sperimentazioni, ...).

4.2 Ringraziamenti personali

Ringrazio tutti coloro che hanno reso possibile tutto questo. Ringrazio di cuore **i professori** del mio corso di Laurea: in particolar modo il prof. Davoli, i proff. Gabbrielli e Martini e il prof. Gorrieri, grazie a loro e ai loro insegnamenti ho capito che tutti possono imparare a programmare, ma in pochi possono diventare "artisti del codice". Spero un giorno di poter render vera la frase "l'allievo supera il maestro" e renderli fieri del loro lavoro. Ringrazio **i miei amici**, che in questi anni mi hanno (quasi!) sempre sopportato. grazie a Flavio, Alessandro e Marco per avermi regalato una quantità indefinita di risate negli ultimi cinque anni; grazie a Francesca e Monica, perché dietro ad un grande informatico (o medico che vorrebbe essere informatico) c'è sempre una grande donna; grazie ad Andrea, Filippo e Ronny, amici da tempo immemore e per i quali dovrei occupare molto più che un trafiletto; grazie ad Andrea *"il Bolo"*, praticamente mio mentore e fonte d'ispirazione informatica perenne; grazie a tutti i ragazzi dell'*Admstaff* e grazie anche a tutti quelli che non sono presenti in questa lista (colleghi, compagni di studio, ..., sarebbero troppi!). Grazie a mia **nonna**, Ardea, perché la sua forza d'animo è sempre stata una grande lezione. Grazie a **Miria**, inconsapevole e modesta insegnante di vita e divertimento, spero che Mirco e Carlotta siano consapevoli della fortuna che hanno nell'avere una mamma giovane e vivace come te. Grazie a **mio padre**, Paolo: un'affettuosa enciclopedia vivente. È grazie a lui se sono a conoscenza del vero significato di parole come "responsabilità", "rispetto" e "lealtà": valori che è raro trovare tutti insieme in un'unica persona e che grazie a lui riesco a ritrovare sia in me stesso che nei miei fratelli Claudio e Fabio. Grazie a Pietro e Marinella, che hanno messo al mondo e cresciuto una creatura meravigliosa, **Alice**, che mi affianca, mi protegge, mi cresce, mi stima, mi consola e mi sgrida. Ma soprattutto mi sprona a fare sempre del mio meglio e a farlo sempre col sorriso. Grazie.

Infine, grazie a **mia mamma**, Cinzia, che anche se non è più fra noi ne conservo gelosamente il ricordo nel cuore, ed è lì che mi rifugio quando ho bisogno di un calore confortante e un abbraccio. Egoisticamente, avrei voluto averti accanto fino ad oggi, ma ho scoperto che la Felicità e l'Amore di una mamma oltrepassano il contatto fisico. Tutto questo è dedicato a te.

Riferimenti bibliografici

[1] Gerald Carter. *LDAP System Administration.* O'Reilly, 2003.

[2] Microsoft Corporation. *Hyper-V.* http://www.microsoft.com/en-us/server-cloud/hyper-v-server, 2008.

[3] Renzo Davoli and Michael Goldweber. *Virtual Square: Users, Programmers & Developers Guide.* Renzo Davoli, Michael Goldweber Editors, 2011.

[4] Mike Eisler Hal Stern and Ricardo Labiaga. *Managing NFS and NIS.* O'Reilly, 2001.

[5] Timothy Howes and Mark Smith. *LDAP: Programming Directory-enabled Applications With Lightweight Directories.* New Riders, 1997.

[6] Cricket Liu and Paul Albitz. *DNS and BIND.* O'Reilly, 2006.

[7] Damien Miller. *jBCrypt — Strong Password Hashing for Java.* http://www.mindrot.org/projects/jBCrypt, 2010.

[8] Erik T. Ray. *Learning XML.* O'Reilly, 2003.

[9] Wikipedia. *BCrypt — Wikipedia, The Free Encyclopedia.* http://en.wikipedia.org/wiki/BCrypt, 2011.