ALMA MATER STUDIORUM · UNIVERSITÀ DI BOLOGNA

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

Esperimenti di Sistemi Distribuiti Virtuali riconfigurabili per il risparmio energetico

Tesi di Laurea in Sistemi Operativi

Relatore:
Chiar.mo Prof.
Renzo Davoli

Presentata da:
Angelo Maurizio Alfano

I Sessione
Anno Accademico 2009/2010

ALMA MATER STUDIORUM · UNIVERSITÀ DI BOLOGNA

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

Esperimenti di Sistemi Distribuiti Virtuali riconfigurabili per il risparmio energetico

Tesi di Laurea in Sistemi Operativi

Parole Chiave:

virtualizzazione, reti, virtual networking, sistemi operativi

Relatore:
Chiar.mo Prof.
Renzo Davoli

Presentata da:
Angelo Maurizio Alfano

I Sessione
Anno Accademico 2009/2010

...a mia madre, mio padre e Valentina

Indice

Introduzione

Il significato del termine italiano *virtuale* indica un evento che esiste potenzialmente ma non si è ancora verificato, ad esempio in una gara si ha un *vincitore virtuale*; lo stesso termine nell'ambito dell'ottica indica qualcosa di fittizio, non reale, *immagine virtuale*; nella fisica rappresenta qualcosa che si potrebbe effettuare, *lavoro virtuale*; nell'ambito burocratico invece indica la condizione in cui il pagamento di un tributo o simile avvenga direttamente e non tramite una marca da bollo, *imposta virtuale* e cosi via nei vari contesti. Nell'ambito puramente informatico il termine virtuale dalla definizione, data da Wikipedia[23], sta ad indicare "la creazione di una versione virtuale di una risorsa normalmente fornita fisicamente". In pratica definisce un livello di astrazione che si va a sovrapporre all'hardware fisico di un elaboratore, permettendo in questo modo di poter installare diversi sistemi operativi sulla stessa macchina. Ciò viene reso possibile grazie alla creazione di *virtual machine*, che sono ambienti appositamente creati dal sistema di virtualizzazione e dispongono di un set di risorse hardware virtuali (RAM, CPU, HD, NIC, etc.). Su di esse è possibile installare un sistema operativo ed eseguire delle applicazioni in modo indipendente, rispetto all'hardware fisico sottostante, aumentando in tal modo l'utilizzo e la flessibilità delle risorse hardware disponibili.

La virtualizzazione comunque è un concetto già presente nell'architettura dei sistemi operativi, infatti esiste la *memoria virtuale* che consiste nel simulare uno spazio di memoria centrale maggiore di quello fisicamente presente, grazie all'utilizzo di spazi di memoria secondaria su altri dispositivi, oppure esiste la condizione in cui ogni processo caricato nel sistema, *virtualmente,* utilizza tutta la

potenza di calcolo quando invece, tale potenza di calcolo è ripartita tra tutti i processi secondo gli algoritmi di scheduling del sistema operativo in uso.

In questa tesi ci occuperemo della virtualizzazione intesa come livello di astrazione dell'hardware e analizzeremo i vantaggi che una tale soluzione comporta. Il principale vantaggio che la virtualizzazione comporta è che le virtual machine con essa create, possono essere implementate in modo tale da garantire che siano *isolate* dalla macchina fisica, si andrebbe così a creare un *box* in cui sarebbe possibile fare test di debugging o ad esempio testare la sicurezza di un sistema, riproducendone virtualmente le caratteristiche. Viene poi fornita una sorta di *incapsulamento*, in quanto tutta la virtual machine sarà contenuta in pochi file, facili da spostare e da copiare semplificando in tal modo le procedure di *backup* e *disaster recovery*, in quanto avendo una copia giornaliera delle immagini virtualizzate delle risorse[1], sarà possibile un ripristino rapidissimo in caso di malfunzionamenti dovuti ai più disparati motivi. Inoltre con la virtualizzazione si ottiene il *partizionamento* delle risorse disponibili (un po' come succede per gli hard disk), realizzando una sorta di mascheramento di tali risorse agli utilizzatori, con l'obiettivo di evitare all'utente di dover comprendere e gestire dettagli complessi delle risorse e di aumentare, nel contempo, la condivisione e l'utilizzo delle risorse, garantendo la possibilità di una successiva espansione.

Esistono poi sistemi di virtualizzazione che permettono la migrazione a caldo di una virtual machine detta anche *migrazione live*, con gestione dei *failov*er del tutto trasparente all'utente. In pratica, nell'eventualità di un guasto hardware, la virtual machine sarà automaticamente migrata su un altro server in servizio senza alcuna interruzione operativa.

La virtualizzazione di una risorsa, hardware o software, consente di realizzare una copia identica per funzionalità alla risorsa reale. Unità ottiche, spazio di archiviazione, sistemi operativi, divisione di un hard disk in unità logiche, costituiscono tutti esempi di risorse virtualizzabili.

Il concetto di virtual machine non è comunque una novità, infatti questa esi-

[1] che è rappresentata da un semplice file o comunque realizzabile attraverso la tecnica degli *snapshot*, che verrà illustrata in seguito

genza di poter sfruttare una macchina potente era già utilizzato nel 1967, quando ci fu la prima implementazione del CP-40 su IBM System 360, un sistema operativo basato su memoria virtuale e time sharing, che consentiva di eseguire tante istanze software del Sistema Operativo sullo stesso hardware con lo scopo di partizionare la potenzialità dei grandi mainframe.

Nel corso degli anni l'introduzione di Minicomputer e Personal Computer, fornì una via più accessibile ed efficiente per distribuire la potenza di calcolo, che portò intorno agli anni '80 all'accantonamento del concetto di virtualizzazione. Negli anni '90 invece si assiste alla reviviscenza di questa tecnologia, poiché risultò chiaro che sarebbe potuta essere stata d'aiuto per la risoluzione di tutte quelle problematiche relative alla proliferazione di hardware meno costosi tipo: il sottoutilizzo delle risorse, i costi di gestione e la manutenzione elevati e la vulnerabilità sempre più alta.

Struttura della tesi

Premettiamo che quando possibile, nei capitoli seguenti, si è cercato di utilizzare le voci di Wikipedia come un riferimento primario per la maggior parte dei termini e delle tecnologie trattate, perché in molti casi Wikipedia fornisce una valida descrizione, per lo più imparziale, che non promuove alcuna soluzione tecnica unica per una data tecnologia.

Nel *capitolo* 1 - UN PO' DI TEORIA vengono citati tutti i riferimenti teorici alla base della virtualizzazione, partendo dai teoremi fondamentali di Popek e Goldberg, per poi passare alle definizioni di Virtual Machine, Hypervisor ed illustrare i tipi di virtualizzazione possibili. Inoltre vengono illustrate le soluzione hardware introdotte sia da Intel che da AMD, per garantire una virtualizzazione più efficiente su macchine x86. Infine vengono analizzati i pro e i contro che una soluzione virtualizzata comporta;

Nel *capitolo* 2 - OBIETTIVI DELLA TESI viene presentata la situazione attuale in cui questo progetto dovrebbe trovare l'applicazione, cioè il laboratorio infor-

matico denominato "Ercolani", illustrandone la struttura ed il funzionamento per poi passare alla presentazione dell'idea che dovrebbe permettere, grazie all'utilizzo di appositi script e della virtualizzazione, un notevole risparmio energetico, illustrando nei vari paragrafi tutta una serie di prodotti che hanno fatto parte della realizzazione del progetto finale.

Nel *capitolo* 3 - IMPLEMENTAZIONE viene illustrato lo schema di ragionamento utilizzato per realizzare l'idea proposta nel capitolo 2, con la descrizione dei principali script, che permettono il funzionamento dell'intero sistema e la discussione dei principali problemi che sono stati affrontati nella realizzazione della stessa.

Nel *capitolo* 4 - CONCLUSIONI E SVILUPPI FUTURI si conclude il lavoro riassumendo le fasi del progetto e le linee guida seguite, che hanno permesso di realizzare questo lavoro di tesi, si fa una valutazione sui risultati ottenuti e si citano gli aspetti negativi da tener presente per non vanificare i vantaggi che questa soluzione potrebbe portare. Infine, sono state elencate delle idee nate durante lo svolgimento del progetto, che si propongono di migliorare alcuni aspetti implementativi e l'aggiunta di nuove funzionalità rispetto alla situazione attuale.

Capitolo 1

Un po' di teoria

1.1 Teorie di Popek e Goldberg

In questo capitolo tratteremo un approfondimento di quelle che sono le linee guida per la virtualizzazione di Popek e Goldberg citando, di seguito, le condizioni necessarie e sufficienti che una piattaforma deve soddisfare affinché possa essere correttamente virtualizzata.

Tali parametri sono stati introdotti nel 1974 appunto da Gerald J. Popek e Robert P. Goldberg[1] [1] e ancora oggi rimangono validi rappresentando le direttive che i programmatori devono seguire durante la progettazione di architetture in grado di supportare la virtualizzazione in maniera efficiente.

In definitiva, seguendo i loro presupposti, possiamo parlare di virtualizzazione se e solo se vengono rispettate le seguenti tre caratteristiche fondamentali:

- **Equivalenza**: Il sistema eseguito nell'ambiente virtuale si deve comportare esattamente come se fosse eseguito su una macchina fisica equivalente;

- **Controllo delle risorse**: L'ambiente di virtualizzazione deve avere il completo controllo delle risorse virtualizzate;

[1] Gerald J. Popek ricercatore dell'University of California, Los Angeles e Robert P. Goldberg Honeywell ricercatore dell'Harvard University

- **Efficienza**: Buona parte delle istruzioni della piattaforma virtualizzata devono essere eseguibili senza l'intervento dell'ambiente di virtualizzazione.

La realizzazione a livello software della virtualizzazione delle risorse è affidata ad una categoria di applicazioni che va sotto il nome di Virtual Machine Manager (VMM), che rappresenta uno strato di software che si interpone tra l'hardware fisico e le virtual machine rispettando le tre caratteristiche fondamentali viste prima.

Per definire i teoremi di Popek e Goldberg viene introdotta una classificazione delle *instruction set architecture* (ISA) del processore in tre gruppi:

- **Istruzioni privilegiate**: L'esecuzione di istruzioni privilegiate[2], da parte di una virtual machine, causano una eccezione che viene catturata (*trap*) dalla VMM che la emula, ricorrendo ai servizi del sistema operativo host.
- **Istruzioni sensibili al controllo**: Tali istruzioni tentano di cambiare la configurazione delle risorse nel sistema.
- **Istruzioni sensibili al comportamento**: Sono istruzioni il cui risultato dipende dalla configurazione delle risorse (modalità di funzionamento del processore o contenuti dei registri).

Enunciata tale distinzione tra le categorie di istruzioni del processore, risulta possibile ora enunciare i due teoremi.

Teorema 1: *«Per ogni computer convenzionale, di terza generazione*[3]*, può essere realizzato un VMM in grado di emularlo se i set di istruzioni sensibili al controllo e al comportamento sono sottoinsiemi delle istruzioni privilegiate.»*

In altre parole, tutte le istruzioni privilegiate, sono catturate perché eseguite dalla VMM in un contesto non privilegiato mentre le istruzioni non privilegiate possono essere eseguite direttamente senza l'intervento della VMM, coerentemente al principio dell'efficienza. Questa caratteristica consente di realizzare una

[2] sono privilegiate le istruzioni che possono essere eseguite solo in Supervisor mode, nel ring 0

[3] è una macchina che ha un processore convenzionale con due modalità di funzionamento: supervisor e user. In modalità supervisor è disponibile l'intero set di istruzioni, mentre in modalità user no.

virtualizzazione ricorsiva, utile a definire le condizioni in cui un VMM può essere eseguito in una copia di se stesso, come meglio specificato nel secondo teorema:

Teorema 2: *«Un sistema di terza generazione è virtualizzabile ricorsivamente se:*

1. *è virtualizzabile*
2. *è possibile costruire per essa una macchina virtuale che non abbia dipendenze di temporizzazione.»*

In sostanza, se una virtual machine può eseguire una copia del VMM, allora la macchina originale è *ricorsivamente virtualizzabile* e l'unico limite al numero di VMM nidificate è rappresentata solo dalla quantità di memoria disponibile nel sistema.

Esistono poche architetture di terza generazione che risultano essere pienamente virtualizzabili, quindi per far si che vengano comprese anche quelle architetture che non rispettano pienamente le caratteristiche viste prima, vengono ampliate quelle definizioni in modo da ottenere una forma più generale anche se meno efficiente. Parliamo di Hybrid Virtual Machine (HVM), la sua struttura è molto simile al sistema del VMM visto prima, ma in questo caso ci sono più istruzioni interpretate di quelle eseguite direttamente, questo comporta un'efficienza minore. Da questa nuova definizione deriva un ulteriore teorema, che altro non è che una estensione del teorema 1:

Teorema 3: *«Un monitor HVM può essere costruito per qualsiasi macchina convenzionale di terza generazione, in cui l'insieme di istruzioni sensibili sono un sottoinsieme del set di istruzioni privilegiate.»*

1.2 Virtual Machine e Hypervisor

Dalla definizione di Popek e Goldberg una virtual machine è rappresentata da: "*un efficiente, ed isolato duplicato di una macchina reale*", definizione che tiene fede ancora oggi alla condizione che ogni utente di una virtual machine, può installare un diverso sistema operativo ed avere l'impressione di essere l'unico utilizzatore dell'intero sistema. Tale tecnica utilizzata per permettere la condivisione tra

più utenti di un'unica macchina, viene realizzata grazie ad un software chiamato hypervisor o virtual machine monitor.

Un hypervisor realizza un livello di astrazione tra l'hardware reale dell'elaboratore, su cui funziona, e l'ambiente virtuale, in modo da far credere alle varie virtual machine in esecuzione, di aver a disposizione tutte le risorse hardware, controllando e gestendo questi accessi da parte delle virtual machine in maniera celata alle stesse. L'hypervisor realizza queste operazioni in diversi modi a seconda del sistema di virtualizzazione e si occupa di risolvere le eventuali eccezioni che avvengono quando una virtual machine prova ad eseguire un'istruzione privilegiata, mantenendo una discreta leggerezza di esecuzione, per non gravare sull'efficienza delle virtual machine. Possiamo quindi definire una virtual machine come un ambiente creato dall'hypervisor.

Esistono fondamentalmente due tipi di hypervisor:

- TIPO 1: detti *nativi* (o *bare metal*) (vedi fig.1.1 a), vengono eseguiti direttamente utilizzando le risorse hardware della macchina host e non si appoggiano a software sottostante. Soluzione adottata da Xen;

- TIPO 2: o *hosted* (vedi fig.1.1 b), sono applicazioni eseguite sempre sulla macchina host, ma in modo tale che software di virtualizzazione, hypervisor e sistema operativo host, siano eseguiti su tre livelli diversi. Questo tipo di hypervisor è quello implementato fra l'altro dalle soluzioni adottate da: VMWare Workstation, Microsoft Virtual PC e VirtualBox.

1.2.1 Approfondimento sulle virtual machine

Iniziamo con il dividere le virtual machine in due categorie: le *system virtual machine*, che forniscono una virtualizzazione completa di una piattaforma e supportano l'esecuzione di un sistema operativo e le *process virtual machine*, realizzate per eseguire un solo programma, ovvero che supportano l'esecuzione di un singolo processo.

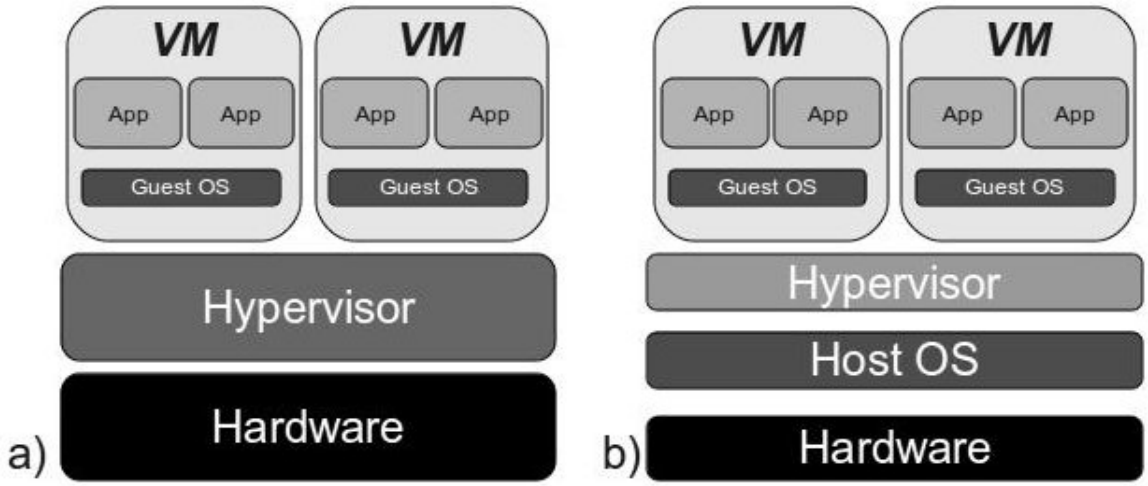

Figura 1.1: a) Hypervisor nativi - b) Hypervisor hosted

Una caratteristica essenziale delle virtual machine è che il software in esecuzione su di esse è limitato a sfruttare le risorse virtuali che gli vengono fornite.

Le *system virtual machine* sono a volte anche chiamate *hardware virtual machine* e consentono la condivisione delle risorse fisiche del sistema tra diverse virtual machine, ciascuna con il proprio sistema operativo.

I principali vantaggi che si ottengono con queste macchine sono: la possibilità di poter eseguire diversi sistemi operativi su un unica macchina, garantendo comunque l'isolamento tra i diversi sistemi, la possibilità di eseguire un *instruction set architecture* diverso da quello che fornisce l'hardware reale ed infine semplificano la distribuzione di applicazioni, la manutenzione, il *disaster recovery* e l'*high availability.*

L'esecuzione di numerose virtual machine sullo stesso server, ciascuna con il proprio sistema operativo, è una pratica spesso usata nel *server consolidation* che ha proprio come obiettivo quello di sfruttare al meglio le risorse dei server che si hanno a disposizione, riducendo al minimo il numero delle macchine fisiche, combattendo il problema del *server sprawl*, che si presenta quando si hanno numerosi server che occupano molto spazio e consumano molte più risorse di quanto

dovrebbero rispetto al carico di lavoro cui sono sottoposti.

La maggior parte di questa tipologia di apparati subisce un carico di lavoro del 15-20% della loro capacità, il che evidentemente costituisce un paradosso dal punto di vista economico. Servizi che solitamente sono eseguiti su server diversi, in quest'ottica sono eseguiti su VM diverse ma sullo stesso server, pratica questa che è solitamente definita come *quality-of-service isolation* (QoS isolation). La pratica di passaggio da numerosi piccoli server ad un solo server che virtualizza l'esecuzione degli altri è detta *Physical to virtual* o *P2V transformation*. Questa tecnica viene anche utilizzata nell'analisi forense e consente la creazione di una copia esatta e funzionante di un calcolatore, comprese le partizioni nascoste o criptate, e il tutto avviene senza alterare in alcun modo i dati.

Le *process virtual machine* sono virtual machine eseguite come se fossero normali applicazioni all'interno di un sistema operativo. Esse vengono create quando il processo è avviato e distrutte quando viene terminato. Il loro scopo è quello di fornire un ambiente di programmazione che crei un livello di astrazione dalle risorse hardware a disposizione del sistema operativo e che consenta per tanto di eseguire il programma su qualsiasi piattaforma allo stesso modo. Una process virtual machine deve fornire un livello di astrazione concettualmente simile a quello dei linguaggi di programmazione ad alto livello, non deve invece preoccuparsi di come sono implementate a livello hardware le sue istruzioni, a differenza delle system virtual machine, i cui set di istruzioni sono a basso livello. Ogni process virtual machine si appoggia su un interprete e sulla compilazione *just in time* per la traduzione delle istruzioni.

Un esempio molto diffuso di questo tipo di macchine è rappresentato dalla *Java Virtual Machine* o JVM (vedi fig.1.2) , la quale esegue i programmi scritti in bytecode[4] e presenta la caratteristica di disporre di implementazioni della virtual machine Java per diversi ambienti operativi, che rappresenta la chiave della portabilità di Java, proclamata nello slogan "*write once, run everywhere*" ("scrivi una volta, esegui dappertutto"). La virtual machine realizza infatti un

[4] risultato della compilazione di sorgenti scritti in linguaggio Java

ambiente di esecuzione omogeneo, che nasconde al software Java (e quindi al programmatore) qualsiasi specificità del sistema operativo sottostante.

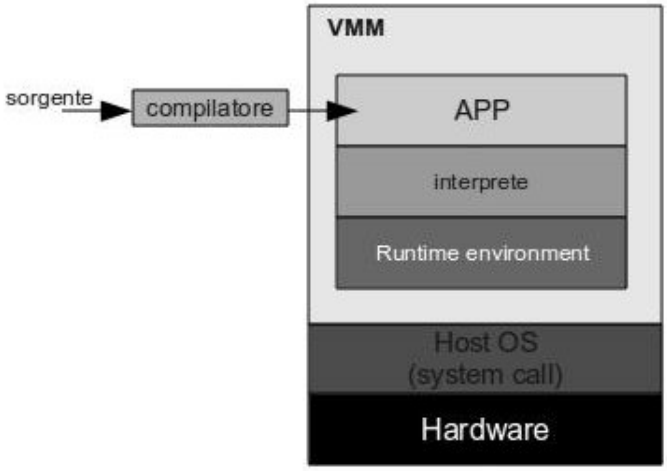

Figura 1.2: Process virtual machine

Un altro esempio è quello del .NET Framework di Microsoft, la cui virtual machine si chiama *Common Language Runtime*, altri tipi di process virtual machine sono quelle che consentono di astrarre dai meccanismi di comunicazione dei computer cluster[5], tali meccanismi di comunicazione che di conseguenza spesso risultano essere eterogenei. In tal caso esiste una virtual machine per ciascun processo consentendo al programmatore di occuparsi degli algoritmi di condivisione delle risorse e distribuzione più che sui meccanismi di comunicazione.

1.3 Tipi di Virtualizzazione

Prima di approfondire il discorso relativo ai tipi di virtualizzazione bisogna fare una piccola premessa, l'architettura IA-32, più comunemente conosciuta come

[5] gruppo di computer collegati che lavorano insieme, condividendo capacità di calcolo ed altre risorse

x86, non è nativamente compatibile con quelli che sono i criteri di virtualizzazione di Popek e Goldberg visti precedentemente e per ovviare a queste mancanze, esistono sia soluzione software (che affronteremo subito), che soluzioni hardware che saranno esaminate successivamente.

Il problema della virtualizzazione è stato affrontato in diversi modi e dal lato software può essere realizzato secondo quattro metodologie differenti: l'*emulation*, la *full virtualization*, la *paravirtualization* e la *operating system level virtualization*. Tutte queste metodologie danno l'impressione all'utilizzatore di disporre di un sistema operativo *stand alone*, non virtualizzato.

1.3.1 Emulation

Con l'emulazione l'hypervisor presenta alle varie virtual machine un ambiente simulato dell'hardware sottostante, non necessariamente uguale alle risorse realmente disponibili che consente al sistema operativo guest[6] di essere utilizzato senza alcuna modifica (vedi fig.1.3).

Figura 1.3: L'emulatore simula un'architettura hardware diversa da quella fisica

[6]sistema operativo in esecuzione sulla virtual machine

I limiti prestazionali e di funzionalità sono facilmente intuibili, infatti l'emulatore presentando un hardware standard ha bisogno di interfacciare tutte le risorse hardware reali (cpu, memoria, rete, etc.), inoltre non ha la possibilità di sfruttare l'hardware e le eventuali istruzioni implementate per favorirne la virtualizzazione, come vedremo nel paragrafo seguente.

Per quanto riguarda la memoria, viene generalmente assegnata alla virtual machine una sequenza di indirizzi di memoria che utilizza in modo esclusivo e l'emulatore somma all'indirizzo della cella richiesta una costante.

Un esempio di emulatore è rappresentato da QEMU.

1.3.2 Full Virtualization

La *Full Virtualization* detta anche *Native Virtualization* è molto simile all'emulazione. Come per l'emulazione, i sistemi operativi guest girano senza alcuna modifica in macchine virtuali ospitate sul sistema host[7], però in questo caso i sistemi operativi guest devono essere compatibili con l'architettura hardware della macchina fisica (vedi fig.1.4).

[7] la struttura fisica composta da componenti hardware

Figura 1.4: La Full Virtualization presenta al sistema operativo guest la stessa architettura hardware presente sull'host fisico

In questo modo, si ottiene una efficienza migliore, in quanto alcune istruzioni possono essere eseguite direttamente sull'hardware senza l'utilizzo di intermediari, con un notevole aumento nella velocità di esecuzione e con l'ulteriore possibilità di sfruttare le specifiche istruzioni implementate nei processori.

Esempi di software che utilizzano la full virtualization sono VMware, Parallel Desktop, Virtual Box e Xen, limitatamente ai sistemi operativi proprietari non modificabili.

1.3.3 Paravirtualization

Un'altra tecnica di virtualizzazione è la *paravirtualization*. L'hypervisor mostra alle macchine virtuali una versione virtuale dell'hardware sottostante, mantenendone tuttavia la medesima architettura di quello reale, quindi si ha qualche similitudine con il sistema precedente, ma si rende necessario modificare il sistema operativo in esecuzione sulle macchine virtuali per evitare l'esecuzione di alcune particolari istruzioni privilegiate. Non viene invece modificata l'*Application Binary Interface* (ABI) e ciò consente di eseguire le applicazioni senza modifiche (vedi fig.1.5).

Questa tecnica permette di ottenere un decadimento delle prestazioni minimo rispetto al sistema operativo non virtualizzato, infatti la maggior parte delle istruzioni sensibili vengono direttamente eseguite sull'hardware reale e solo una parte minore utilizza il sistema delle trap.

Esempi di paravirtualizzazione sono Xen e User-mode Linux.

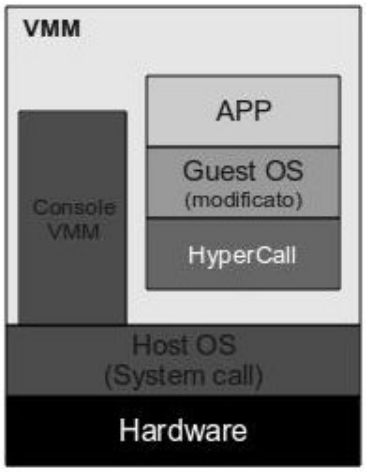

Figura 1.5: La Paravirtualization è simile alla full virtualization, ma è necessario modificare il sistema operativo guest per ottimizzare l'esecuzione sull'ambiente virtualizzato

1.3.4 Operating System level Virtualization

Una quarta tecnica di virtualizzazione è la cosiddetta *Operating System level Virtualization*, che viene realizzata attraverso la creazione di copie del sistema operativo installato sull'host. I sistemi guest creati saranno a tutti gli effetti istanze del sistema operativo host con un proprio file system, configurazione di rete, applicazioni e librerie, ed avrà la possibilità di avere un utente root ed altri utenti, il tutto senza la necessità di un hypervisor che gestisca le risorse, cosa che viene realizzata soltanto dal kernel (vedi fig.1.6).

Il vantaggio principale di questa tecnica è il miglior utilizzo delle risorse grazie alla condivisione di spazi di memoria ed una migliore efficienza nell'esecuzione delle varie istanze. Essendo i sistemi operativi delle macchine guest equi-

valenti a quello della macchina host, le istanze guest non richiederanno un kernel privato, ma utilizzeranno lo stesso con un conseguente minor utilizzo di memoria fisica. Questo è tuttavia il principale svantaggio dei questa tecnica, che non la rende idonea nel caso si volesse avere la possibilità di eseguire sistemi operativi diversi sullo stesso host.

Un altro punto debole di questa tecnica è il limitato isolamento tra le macchine guest, infatti un problema di stabilità o di performance di un solo guest può influire negativamente sugli altri.

Esempi di Operating System level Virtualization sono OpenVZ (Virtuozzo), Linux VServers, Solaris Containers.

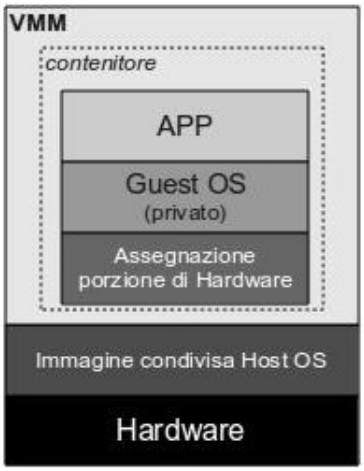

Figura 1.6: La Operating System level Virtualization mette a disposizione ai sistemi guest l'immagine del sistema operativo in esecuzione sull'host.

1.4 Intel VT e AMD-V

Come abbiamo anticipato nei paragrafi precedenti, per realizzare una virtualizzazione il più efficiente possibile, anche i produttori di hardware hanno cercato di contribuire con l'introduzione di un set di istruzioni grazie al quale si ottengono notevoli vantaggi, ad esempio si riesce a ridurre, se non addirittura eliminare, le

modifiche da apportare al sistema operativo nel caso di una paravirtualizzazione, oltre ad un notevole incremento delle performance, in quanto il VMM è meno impegnato.

La soluzione offerta inizialmente da Intel nella tecnologia Intel-VT, consente alla CPU di agire come se il sistema fosse dotato di più CPU che lavorano in parallelo, questo rende possibile l'esecuzione di più sistemi operativi contemporaneamente sulla stessa macchina fisica. Intel-VT aggiunge un set di dieci nuove istruzioni specificatamente realizzate per la virtualizzazione: VMPTRLD, VMREAD, VMPTRST, VMCLEAR, VMRESUME, VMWRITE, VMLAUNCH, VMCALL, VMXOFF e VMXON. Inoltre prevede l'introduzione di una nuova modalità operativa chiamata VMX root[8] nella quale il VMM è eseguito con privilegi massimi, eliminando la necessità di de-privilegiare il kernel del sistema operativo guest che opera nella modalità VMX non–root. Entrambe le modalità supportano i quattro livelli di privilegio dal ring 0 al 3. In questo modo il kernel del sistema operativo guest potrà lavorare nel ring 0, eliminando la necessità che il VMM intervenga mascherando il livello di privilegio in alcune chiamate di sistema (vedi fig.1.7).

[8] detto anche ring -1

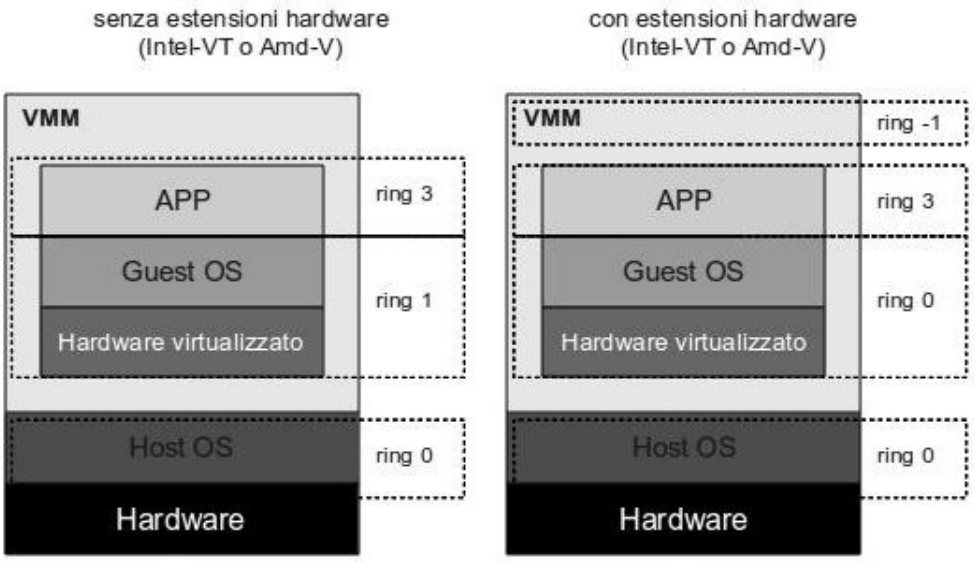

Figura 1.7: Differenze tra un sistema con estensione hardware e senza

Esistono poi due nuove transizioni di privilegio: *VM entry* che permette di passare dalla modalità VMX root a quella VMX non-root e *VM exit* che fa il lavoro inverso. In pratica, per avviare un guest OS, si abilita la modalità VMX root con l'istruzione VMXON, dopodiché è necessario allocare 4 KB di memoria e passarne l'indirizzo al comando VMPTRLD, questo array di 4 KB è dedicato al salvataggio di tutti i bit che descrivono lo stato del sistema operativo guest mentre questo non è attivo e rimane allocato finché la sessione rimane attiva, ovvero finché VMCLEAR non è lanciato. Questo è sufficiente per avviare una sessione di una virtual machine. Successivamente, usando l'istruzione VMLAUNCH che permette l'interazione iniziale con VM entry, oppure VMRESUME per le interazioni successive, viene eseguita una virtual machine in non-root mode, mentre ovviamente il VMM viene eseguito in root-mode. Infine VMXOFF spegne il guest transitando, tramite VM exit, nella modalità non-root.

Anche AMD, con l'AMD Virtualization (o in breve AMD-V o Pacifica), si è occupata di tale problema producendo una tecnologia che consente di eseguire le applicazioni progettate per Intel-VT anche se i nomi dei comandi mnemonici, visti prima, sono diversi. Nella pratica la prima implementazione da parte di In-

tel e AMD di questa tecnologia non ha offerto sostanziali vantaggi prestazionali rispetto alle soluzioni preesistenti, in quanto le istruzioni VM Exit e VM Entry incorrono in un notevole overhead, principalmente dovuto alla ricostruzione nei registri dello stato della VM o dell'host. Questo overhead, in particolare, risulta oneroso nella gestione del processo di traduzione degli indirizzi fisici delle macchine virtuali in indirizzi fisici dell'host.

Per comprendere meglio quest'ultimo passaggio bisogna fare una precisazione sulla gestione della *memoria virtuale*[9] nei sistemi operativi che viene implementata attraverso un processo chiamato di *paginazione*, che consiste nel realizzare un'area di memoria virtuale sul disco e una mappa di memoria che crea una corrispondenza tra gli indirizzi della memoria fisica e gli indirizzi della memoria virtuale su disco. Questa mappa della memoria è gestita da un componente hardware chiamato Memory Management Unit (MMU).

In un ambiente di virtualizzazione, può accadere che più sistemi operativi debbano accedere alla stessa memoria fisica. Per risolvere tale conflitto, si aggiunge un nuovo livello di indirizzamento chiamato *shadow paging*: il VMM ha il compito di occuparsi di gestire una mappatura tra gli indirizzi della memoria fisica dell'host e gli indirizzi della memoria "fisica" della virtual machine, memorizzandola in una tabella detta shadow page table(SPT). Il VMM intercetta tutte le istruzioni che modificano la struttura della memoria affinché lo stato della MMU non possa essere alterato dalla virtual machine ed inoltre garantisce la sincronizzazione tra lo SPT e le relative pagine fisiche di memoria.

Ritornando di nuovo ai processori di AMD e Intel, una seconda generazione di estensioni hardware ha parzialmente risolto i problemi di efficienza attraverso l'utilizzo di tecniche come le *Nested Page Tables* di AMD (*Extended Page Tables* per Intel), che in pratica riducono il lavoro svolto dal VMM dato che la mappatura tra le pagine di memoria della virtual machine e le pagine di memoria della macchina fisica sono gestite in hardware.

[9] è una architettura di sistema capace di simulare uno spazio di memoria centrale maggiore di quello fisicamente presente

1.5 Virtualizzazione: Svantaggi

In questo paragrafo, dopo aver visto i vantaggi che un sistema virtualizzato comporta, verranno analizzati anche i fattori negativi che la accompagnano. La virtualizzazione infatti, sta rivoluzionando i data center in larga parte positivamente, ma nessuna tecnologia è senza lati negativi e in questo caso, ci sono molti problemi potenziali di gestione, sicurezza, consumi energetici e ritorni d'investimento che possono pregiudicare un progetto di virtualizzazione se questo non è stato accuratamente pianificato.

Partiamo dall'idea che con un equivalente sistema virtualizzato si riduca il lavoro di gestione per l'IT. Non è del tutto vero, infatti la semplicità di poter creare una nuova virtual machine per una nuova funzionalità, può portare all'incontrollata prolificazione di virtual machine (sprawl), con la conseguenza di dover gestire un numero considerevole di macchine sia fisiche che virtuali. Insomma, non è detto che il tempo da dedicare alla gestione si riduca, anche l'infrastruttura virtuale va gestita.

Altra cosa da valutare è che, per le applicazioni che girano sulle virtual machine molti fornitori software non offrono lo stesso supporto disponibile per quelle che girano nei server 'veri'. Il calcolo degli esborsi per le licenze può essere anche più complicato del solito in un data center virtuale: occorre studiarsi bene i termini e condizioni nei contratti di licenza dei vari fornitori.

Una volta consolidati i server può venire da pensare di aver finalmente ridotto le bollette dell'energia. Ma anche qui non si può dare niente per scontato. E' vero che i server sono in numero minore, ma ciascuno ha un tasso più alto di utilizzo della CPU, e richiede più energia. Bisogna quindi tenere in considerazione anche questo fattore quando si calcolano i benefici in un eventuale passaggio ad una soluzione virtualizzata. Un altro problema, direttamente collegato al risparmio energetico, è il dissipamento del calore, quando si eliminano molti server, il data center deve essere riconfigurato per evitare di sprecare aria condizionata su spazi vuoti.

Anche se sotto alcuni punti di vista la virtualizzazione aumenta la sicurezza,

infatti la possibilità di "clonare" le virtual machine e trasferirle da una macchina all'altra, facilita di molto le strategie di *disaster recovery*, e quindi la riduzione dei rischi di perdite di dati e indisponibilità, d'altra parte la virtualizzazione, se non gestita in modo appropriato, comporta nuove minacce per la sicurezza dei dati e la continuità dei sistemi. Infatti, concentrare più servizi su un unica macchina fisica, vuol dire ridurre in un unico punto tutte le vulnerabilità. Inoltre gli hypervisor non si occupano di crittatura, per cui aprono in un certo senso la strada ad attacchi cosiddetti *"man-in-the-middle"*, in cui eventuali malintenzionati possono intercettare dati non crittati mentre le virtual machine sono trasferite da un server fisico all'altro.

Come si è visto, un progetto di virtualizzazione, deve essere bene pianificato per non incorrere in una molteplicità di problematiche implementative, le quali rischiano di oscurare i benefici che se ne otterrebbero. In buona sostanza, ogni innovazione ha i suoi pro e i suoi contro, ma un buono studio iniziale ed una progettazione accurata, fanno si che da ogni tecnologia si possano massimizzare solo i vantaggi, limitando al minimo l'incidenza degli aspetti negativi.

Capitolo 2

Obiettivi della Tesi

L'idea di questa tesi parte dall'obiettivo principale di garantire un risparmio energetico al laboratorio in dotazione alla facoltà di informatica, senza pregiudicarne o limitarne le funzionalità ed i servizi attualmente disponibili.

Questo laboratorio, denominato "Ercolani", è composto da 62 personal computer (host) con la stessa configurazione: dual core con 2 Gb di ram. Tali macchine sono accese 24h/24, in quanto devono sopperire all'esigenza di essere sempre raggiungibili dall'esterno tramite *ssh*[1]. Il laboratorio osserva il seguente orario di apertura al pubblico: dal lunedì al venerdì dalle 09:00 alle 19:45 e sabato dalle 09:00 alle 13:45, ma nonostante tali orari, le macchine rimangono sempre accese.

Principalmente, l'obiettivo di questa tesi, consiste nel trovare un sistema che permetta di ridurre il numero di macchine accese durante il periodo di chiusura del laboratorio, lasciando inalterate le funzionalità attuali. Per far ciò, sfruttiamo la virtualizzazione. In pratica verrà creato, su ogni macchina, un suo clone virtuale (VM) che coesiste ad essa prendendo però l'indirizzo IP della macchina reale e assegnando a quest'ultima un altro IP non raggiungibile dall'esterno (vedi fig.2.1).

[1] è un protocollo che permette di stabilire una sessione remota cifrata ad interfaccia a linea di comando con un altro host.

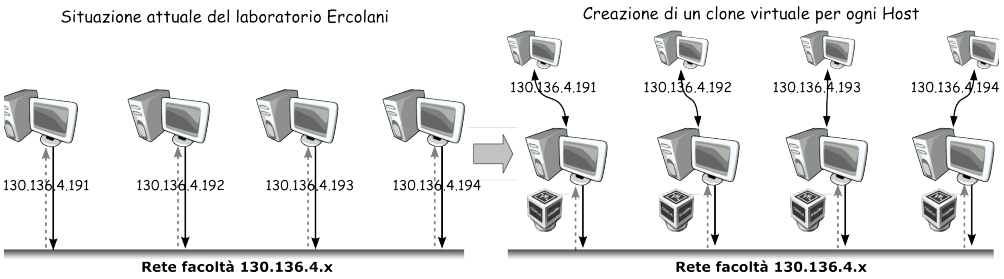

Figura 2.1: Passaggio dalla situazione attuale ad una virtualizzata

Durante l'orario di apertura del laboratorio, tale situazione resta statica, ma alla sua chiusura, tramite appositi script, si avranno una serie di fasi di migrazione di VM da un host all'altro con il fine di compattare le VM in un numero di host fisici il più piccolo possibile. A questo punto gli host che non avranno in esecuzione nessuna VM verranno spenti (vedi fig.2.2).

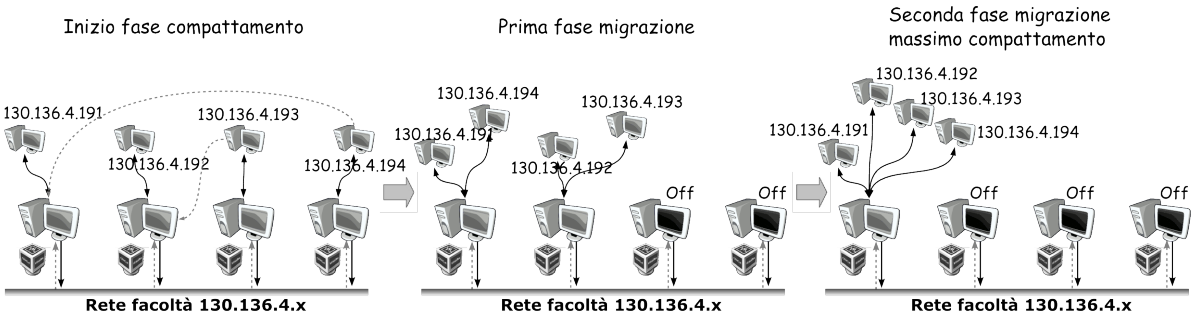

Figura 2.2: Compattamento delle VM in caso di underload, con varie fasi di migrazioni

In questo modo otterremmo un discreto risparmio energetico sia sotto il profilo del consumo elettrico, sia per quanto riguarda l'usura delle macchine fisiche. Nonostante ciò, le funzionalità saranno garantite dal fatto che tramite ssh, le macchine saranno sempre raggiungibili, con la sola eccezione che gli utenti esterni si connetteranno con una VM invece che con un host fisico, cosa che comunque non comporterà nessuna differenza funzionale né tanto meno una reale consapevolezza da parte dell'utilizzatore. Inoltre, tale connessione con le VM rimane garantita anche nella fase di migrazione tra un host e l'altro, in quanto, come meglio spe-

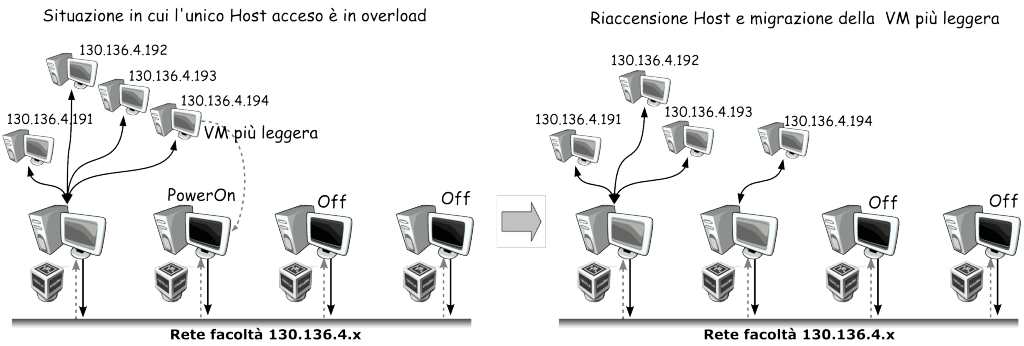

Figura 2.3: Situazione in cui si ha un host in overload e bisogna migrare la VM più leggera

cificato in seguito, si utilizzerà la migrazione live che consente per l'appunto di spostare una VM, senza pregiudicarne l'esecuzione e quindi la sua connessione.

Per effettuare la virtualizzazione la scelta è ricaduta sul prodotto VirtualBox, sistema di virtualizzazione molto interessante e che negli ultimi tempi sta avendo un'impressionante crescita. Nei paragrafi successivi verranno comunque menzionate le caratteristiche interne di questo sistema di virtualizzazione, confrontandolo con altri prodotti disponibili sul mercato.

Ritornando alla virtualizzazione del laboratorio, alla fine della fase di compattamento delle VM, si avrà un demone in esecuzione sull'host manager che monitorerà il carico di lavoro ed in caso di overload, riaccenderà gli host necessari sfruttando la funzionalità wake-up-on-lan[2]. Una volta avviati gli host necessari, si provvederà a migrare su quest'ultimi una o più VM, in modo da equilibrarne il carico (vedi fig.2.3).

Entrando più nel dettaglio della tesi, per creare l'ambiente su descritto, useremo il termine "cluster" per definire nell'insieme tutte le macchine fisiche. Le stesse saranno caratterizzate dal fatto di rappresentare singolarmente un "nodo" di detto cluster ed una sola sarà il "nodo manager" che avrà la caratteristica di gestire le migrazione tra i vari nodi e tenere aggiornato un file di statistica chiamato "ClusterInfo.xml", in cui è descritto l'intero cluster. Questo file tiene conto delle caratteristiche dei singoli nodi e delle VM in esecuzione su di essi, in pratica

[2]funzionalità che permette di accendere una macchina invocando il suo mac-address

analizzando tale file si potrà verificare su quale nodo è in esecuzione una determinata VM, o quanti nodi sono spenti o qualsiasi altra informazione riguardante il funzionamento dell'intero cluster. L'intera struttura del cluster viene creata attraverso una shell opportunamente realizzata, che permette di inserirne i vari nodi.

Per rendere possibile la migrazione delle varie VM da un host all'altro, bisogna che le immagini dei loro dischi risiedano su una parte condivisa, in modo tale che ogni host possa avere accesso a quelle informazioni. Sfruttando la situazione già presente nel laboratorio in esame, si utilizza il sistema NFS[3] in cui vengono salvati oltre alle immagini dei dischi delle varie VM anche tutti i file necessari al funzionamento del sistema: gli script di gestione del cluster, in modo tale che tutti gli host abbiano la possibilità di eseguirli; i file xml che forniscono le proprietà dei singoli nodi e dell'intero cluster ed infine salviamo i file xml creati da VirtualBox, relativi alla configurazione delle varie virtual machine. Un altro prerequisito alla migrazione prevede che la VM da migrare sia registrata su entrambi gli host, quello di partenza e quello di destinazione, e per far ciò si sfrutta il file xml che abbiamo salvato nella parte condivisa.

Durante l'orario di chiusura del laboratorio, l'intero sistema si adatta alla situazione del momento, confrontando lo stato di carico delle macchine ad un valore di soglia minima e massima. Tali valori che sono presenti in un apposito file di configurazione, rappresentano rispettivamente il limite sotto il quale una macchina è considerata in *underload* ed il limite al di sopra del quale, si considera in *overload*. Tra i due limiti la macchina viene considerata invece *regularload*. A seconda di tale stato, il gestore decide se è necessario migrare una VM, spegnere la macchina o non fare nulla.

C'è da specificare che, come anticipato all'inizio di questo paragrafo, tale gestione deve essere attuata solo durante il periodo di chiusura del laboratorio, mentre negli orari di apertura dello stesso bisogna che tutte le macchine siano accese e rimangano tali fino all'orario di chiusura. Per realizzare tale condizione

[3]NFS sta per Network File System ed è un file system che consente ai computer di utilizzare la rete per accedere ai dischi remoti, come fossero dischi locali.

verranno utilizzati vari *crontab*[4] che avviano lo script di gestione a determinati orari e giorni della settimana, mentre quando il laboratorio riapre, viene eseguito un apposito script che si occupa di normalizzare la situazione accendendo gli eventuali host spenti e distribuendo le VM tra tutti gli host.

Teoricamente l'adozione di questo sistema nei periodi di chiusura del laboratorio e quindi di attività del nostro progetto, potrebbe ridurre il numero di macchine fisiche accese a 8 dalle 62 attuali, considerando una situazione di basso carico da parte di eventuali utenti remoti che si collegano alle VM tramite ssh. Tale risultato si ottiene assegnando alle VM una quantità di RAM pari a 256MB e che su ogni host venga imposto un limite massimo di 8 VM (valore ottenuto dalla quantità di memoria disponibile nell'host diviso la memoria assegnata ai guest). In tali ideali condizioni, si avrebbe un risparmio dell'87% di elaboratori in funzione, con i relativi vantaggi di usura delle stesse, risparmio energetico e minore produzione di calore.

Nei paragrafi seguenti, vengono illustrati tutti gli strumenti che sono stati utilizzati per realizzare il progetto di questa tesi, facendo riferimento anche ai sistemi che sono stati valutati e alla fine non hanno fatto parte dello stesso, spiegandone i motivi delle scelte fatte.

2.1 Xen

Xen[10, 14] è stato il primo sistema di virtualizzazione valutato, esso è composto da un hypervisor che permette di eseguire diversi sistemi operativi sulla stessa macchina, in maniera concorrenziale. Nato come progetto della University of Cambridge Computer Laboratory, è ora sviluppato e mantenuto dalla XEN Community che lo rilascia sotto licenza GPL2. E' oramai utilizzato da anni in ambienti sia enterprise che accademici e può essere installato ed eseguito su numerose architetture quali x86-32, x86-64, IA-64, ARM e PowerPC 970.

[4] *il comando crontab consente la pianificazione di comandi, ovvero consente di registrarli presso il sistema per essere poi mandati in esecuzione periodicamente*

Questo sistema di virtualizzazione non è installabile su un sistema operativo già configurato, ma lo va a sostituire prendendo direttamente il controllo dell'hardware della macchina, questo consente un consumo minimo di risorse ed elevate performance.

Il componente Xen Hypervisor si occupa di creare un livello di astrazione che mette in comunicazione l'hardware dell'host fisico con il sistema operativo delle virtual machine, al di sopra di esso sono eseguiti tutti i domini (virtual machine guest), in particolare all'avvio del sistema Xen modificato, si andrà a creare un dominio base chiamato *Domain0*, all'interno del quale è eseguito un demone chiamato *xend*, fondamentale per la gestione del sistema di virtualizzazione.

Come scritto sopra, l'hypervisor si colloca direttamente al di sopra dell'hardware della macchina, gestendo tutte le problematiche di più basso livello, come lo scheduling del processore o il partizionamento della memoria; è inoltre compito dell'hypervisor controllare e gestire l'esecuzione delle virtual machine. Questo componente è molto complesso, quasi come un vero e proprio sistema operativo ed il suo compito è fornire un'astrazione delle risorse hardware della macchina, ai vari sistemi operativi ospitati. Non si occupa invece, delle problematiche di più alto livello, come funzionalità di rete e di I/O, le quali vengono delegate a Domain0. Il Domain0 (o Dom0) è una virtual machine speciale, che ha permessi privilegiati di accesso alle risorse di I/O e che interagisce con le altre virtual machine. Non è possibile eseguire altre virtual machine se non è in esecuzione Dom0, esso è infatti l'unico dominio a cui è permesso l'accesso diretto all'hypervisor ed è quindi il dominio più importante.

Con il nome di *DomainU* (o guest) si indicano, invece, tutte le altre virtual machine diverse da Dom0 presenti in un sistema XEN. Esse possono essere paravirtualizzate (PV guests) o completamente virtualizzate (HVM[5] guests). Il demone xend si occupa di servizi di supporto per la gestione e il controllo dell'ambiente di virtualizzazione. Scritto nel linguaggio di programmazione python, è in

[5] c'è da precisare che questo acronimo sta per Hardware Virtual Machine e non è da confondere con lo stesso acronimo visto nel capitolo precedente che però si riferiva alle Hybrid Virtual Machine

esecuzione su Domain0 e fornisce gli strumenti per creare, modificare, spegnere e avviare virtual machine. La paravirtualizzazione è il punto di forza della tecnologie che stanno alla base di XEN, questa modalità di virtualizzazione è in assoluto quella che consente migliori prestazioni, come abbiamo già discusso nel capitolo precedente.

In questo caso è possibile virtualizzare però, solo sistemi operativi modificati opportunamente per essere eseguiti sopra uno specifico hypervisor. Una di queste modifiche può essere, ad esempio, la creazione di una zona di memoria condivisa tra macchina paravirtualizzata e Dom0, utile a velocizzare operazioni di scambio di dati. Dover modificare il sistema operativo guest significa spesso avere la possibilità di paravirtualizzare solo sistemi operativi open source, in quanto si ha la necessità di andare ad interagire direttamente con i codici sorgenti.

Inizialmente XEN supportava solo la paravirtualizzazione, in un secondo momento grazie alle estensioni hardware da parte dei produttori di processori, è stato aggiunto il supporto alla virtualizzazione completa. Il termine Hardware Virtual Machine (HVM) viene utilizzato per indicare virtual machine completamente virtualizzate, che utilizzano queste estensioni.

La virtualizzazione completa permette di eseguire sulle virtual machine sistemi operativi identici a quelli eseguiti sulle macchine fisiche. Una macchina completamente virtualizzata, non possiede alcuna modifica o driver specifico. Questo tipo di virtualizzazione ci consente di utilizzare sistemi operativi proprietari come guest.

All'avvio di ogni HVM guest, viene caricato in Dom0 un demone utilizzato per intercettare le richieste di accesso alle risorse di I/O e girarle all'hypervisor. Questo tipo di macchine necessita inoltre del caricamento di un BIOS virtuale, che consenta di inizializzare il sistema all'accensione, compito svolto dallo XEN Virtual Firmware. In un sistema XEN tutto ciò che è sopra l'hypervisor è una virtual machine.

La gestione delle virtual machine da parte di un utente è in genere eseguita principalmente via console testuale, ma ci sono tuttavia soluzioni anche grafiche che facilitano i compiti di amministrazione, ad esempio XEN può utilizzare a

questo scopo VirtualManager.

Dopo aver effettuato parecchi test con Xen, alla fine questo sistema è stato escluso poiché sulle macchine del laboratorio è stato installato ubuntu che dalla versione 8.04 ha scelto di utilizzare come virtualizzatore implementato nel kernel KVM al posto di Xen, rendendo quindi la scelta di KVM un passaggio obbligato.

2.2 KVM

Il secondo sistema preso in considerazione è Kernel-based Virtual Machine[8], che è un sistema full-virtualization esclusivamente per GNU Linux, su piattaforma x86 ed è già presente nella maggior parte delle distribuzioni installate nei PC. Esso è stato infatti inserito nel kernel Linux fin dalla versione 2.6.20 uscita nel 2007, ma può essere facilmente installato su qualsiasi distribuzione, l'unico prerequisito è possedere un processore con istruzioni Intel-VT o AMD-V, di cui abbiamo già discusso in precedenza. Esso consiste in un modulo kernel (kvm.ko) che fornisce la virtualizzazione dell'infrastruttura di base ed un modulo specifico per il processore in uso (kvm-intel.ko o kvm-amd.ko) a seconda dei casi.

Usando KVM possono essere eseguite più virtual machine basate su GNU Linux o Microsoft Windows senza modificarne le immagini e ogni virtual machine ha il suo hardware virtualizzato (schede di rete, adattatore grafico, mouse, etc.).

KVM, come abbiamo detto, è esclusivamente destinato a sistemi operativi basati su GNU Linux ed è già presente nella maggior parte delle distribuzioni installate nei PC. Esso è stato infatti inserito nel kernel Linux fin dalla versione 2.6.20 uscita nel 2007, ma può essere facilmente installato su qualsiasi distribuzione, l'unico prerequisito è possedere un processore con istruzioni Intel-VT o AMD-V.

KVM presenta un approccio diverso rispetto ad altri sistemi di virtualizzazione professionale, come XEN, VirtualBox e VMWare ESX. Esso non prevede infatti l'utilizzo di un hypervisor a se stante, ma utilizza il kernel Linux a questo scopo, questa è una scelta strategica decisiva che porta numerosi vantaggi. Uno tra tutti è il fatto che grazie a questo, il team di sviluppo di KVM, non deve preoccuparsi di scrivere da sé driver per le periferiche o tool di gestione, tutto ciò è offerto dal

kernel Linux. Questo permette rilasci frequenti e un codice molto più snello, ma sopratutto rende KVM compatibile con tutte le periferiche, che possono essere utilizzate su kernel Linux.

L'integrazione di KVM con il kernel è così profonda, che le virtual machine sono nel sistema dei semplici processi. Questo ci permette di gestire/monitorare una virtual machine, come si fa con qualunque altro processo del sistema operativo GNU Linux. Ad esempio si possono utilizzare strumenti come, *kill* e *top*, per eliminare o monitorare una virtual machine. In pratica, una volta ricavato il *pid*[6], relativo al processo che rappresenta una determinata virtual machine, diventa semplice verificarne l'andamento o gestirne il suo ciclo di vita, proprio come si fa con gli altri processi del sistema.

Ricapitolando, il kernel Linux, è posto direttamente "sopra" l'hardware della macchina fisica, contiene al suo interno il modulo KVM e le virtual machine sono dei processi che emulano le periferiche di I/O e che comunicano con il kernel attraverso il device */dev/kvm*, un canale di comunicazione tra l'hypervisor e le VM, creato dai due moduli kvm.ko ed a seconda del processore, kvm-intel.ko oppure kvm-amd.ko.

La funzione principale di */dev/kvm* è quella di mettere in contatto la parte user-space (le virtual machine) con quella kernelspace del sistema operativo, che ospita le virtual machine. Attraverso di esso vengono svolte attività fondamentali come l'allocazione di memoria o la lettura/scrittura dei registri dei processori delle VM. Inoltre realizza la modalità "guest mode" che si aggiunge alle due modalità standard dell'ambiente Linux (kernel e user). È proprio questo che permette di trasformare il kernel in un hypervisor. La modalità "guest mode" è utilizzata infatti, per l'esecuzione di operazioni privilegiate da parte delle virtual machine.

Il fatto che le singole virtual machine vengano viste dal sistema come semplici processi, consente al kernel di trattarli come tali e questo ne aumenta notevolmente la facilità di gestione. Infatti per lo scheduling e il mapping della memoria delle VM vengono utilizzati gli strumenti tipici del sistema operativo GNU Linux,

[6] ogni processo viene identificato dal sistema da un numero identificativo univoco, il process ID o pid

cosa che permette di implementare automaticamente un rudimentale sistema di distribuzione di carico tra le diverse virtual machine. Grazie a ciò infatti la macchina che ospita la VM, se dotata di hardware sufficiente, risulta sempre fluida e dimostra buona velocità di risposta, anche quando il sistema ospitato è sotto sforzo.

KVM fa uso di una versione opportunamente modificata di QEMU[7] per realizzare la parte di I/O (dischi, schede di rete, audio, video, ecc.) di una virtual machine. Ogni richiesta di I/O da parte di una VM viene intercettata per essere emulata da QEMU.

La particolarità di KVM di essere praticamente integrato nel kernel, fa in modo di non stravolge il sistema su cui è installato, come farebbero XEN o VMWare ESX. I due prodotti appena citati, infatti, modificherebbero la macchina su cui sono installati, al punto tale da sconsigliare o rendere impossibile l'utilizzo del sistema ospitante per funzioni desktop.

2.3 VirtualBox

Inizialmente non è stato preso in considerazione questo sistema, per via della poca "maturità" e per la mancanza di alcune funzionalità vitali per questo progetto, infatti fino alla versione 3.0 non era possibile effettuare la migrazione live di una virtual machine da un host all'altro. L'ultima versione rilasciata al tempo della stesura di questa tesi, la 3.2.4, rappresenta un valido prodotto che è arrivato a colmare il gap che aveva con gli altri sistemi, ragion per cui è stato scelto per questo progetto, oltre alla sua perfetta compatibilità con ubuntu, sistema utilizzato nelle macchine del laboratorio.

VirtualBox[15] fornisce soluzioni per la virtualizzazione sia professionale che desktop. Esistono infatti due versioni in cui viene rilasciato il programma, la prima, chiamata semplicemente VirtualBox è rilasciata solamente attraverso i binari

[7] QEMU è un emulatore open source. Una emulazione è una simulazione del comportamento dell'hardware necessario per avviare un determinato sistema operativo (hard disk, schede grafiche e di rete, BIOS)

ed è gratuita per un uso personale, ma non per un uso professionale, con una licenza PUEL (Personal Use and Evaluation License). La seconda versione di nome VirtualBox Open Source Edition (OSE) è rilasciata sotto licenza GPL, quindi fornita assieme ai sorgenti. Le due versioni sono equivalenti a meno del fatto che VirtualBox possiede alcune feature specifiche per i clienti enterprise; un esempio di queste funzioni sono il supporto USB e l'RDP (Remote Display Protocol) Server.

Per il sistema operativo ospitante, una VM è semplicemente un programma come qualunque altro. Questa soluzione un po' richiama quella di KVM, solo che in questo caso non è il kernel con appositi moduli a espletare la funzione di hypervisor, ma una componente a parte di VirtualBox, fatto che consente di utilizzarlo anche su macchine che non hanno le specifiche estensioni del processore per la virtualizzazione, ma nel caso in cui quest'ultime fossero presenti permette di sfruttarne le funzionalità. Questa scelta consente a questo prodotto un più ambio raggio di applicazione, anche se bisogna pagare un piccolo prezzo sulla differenza di prestazioni.

Quando avviamo una virtual machine su VirtualBox, nel sistema si aprono contemporaneamente tre processi, uno per l'interfaccia grafica principale o GUI, uno per collegare la VM con la finestra su cui è stata aperta e un ultimo che lavora in background come demone per tenere traccia delle virtual machine avviate e del loro stato. I nomi di questi processi sono rispettivamente VirtualBox, VirtualBox(chiamato con il parametro -startvm), e VBoxSVC.

VirtualBox è organizzato, architetturalmente, secondo il paradigma Client/ Server, tutte le sue funzionalità e le relative virtual machine possono essere controllate da librerie, scritte per creare un livello di astrazione tra il cuore (core) del programma e le interfacce che si trovano a volerlo utilizzare. Queste librerie, chiamate VboxVMM.dll in Microsoft Windows e VboxVMM.so in GNU Linux, nascondono tutta la parte complicata del lavoro del programma e la espongono, sotto forma di funzioni e variabili più facilmente utilizzabili, a chi è interessato ad interagire a basso livello con VirtualBox.

Un esempio di utilizzo di queste librerie è *VBoxManage*, una utility a riga di

comando che permette di controllare le VM alla stessa maniera della GUI. Ed è proprio attraverso questa libreria, che verrà realizzato il progetto di questa tesi, infatti la possibilità di interagire con le VM attraverso la riga di comando, consente di poter realizzare una serie di script per automatizzare alcune sue funzionalità. Inoltre l'utilizzo di questa libreria, permette di effettuare alcune importanti operazioni, non ancora implementate in maniera grafica, che sono già disponibili per VboxManage, ad esempio la migrazione di una VM da un host fisico ad un altro, funzionalità chiamata *teleporting* che verrà analizzata nel prossimo paragrafo.

La classica GUI e VBoxManage non sono gli unici modi per gestire VirtualBox, esistono anche altre soluzioni, come VBoxWeb e l'RDP Server. Entrambi utili a gestire VirtualBox da remoto, il primo è un progetto opensource di una interfaccia grafica scritta in Ajax e quindi accessibile via browser, il secondo è incluso nella versione non OSE ed è parte integrante dell'offerta business di VirtuaBox. Abbiamo anche un ulteriore modo per interfacciarsi con VirtualBox ed è attraverso la libreria *libvirt*, a cui è dedicato un intero paragrafo in seguito.

Ritornando all'architettura di VirtualBox, possiamo dire che sfrutta le caratteristiche della full-virtualization, viste in dettaglio nel capitolo precedente, infatti tutte le operazioni privilegiate, come l'allocazione di memoria fisica e il salvataggio dello stato della CPU, sono effettuate tramite un driver (*vboxdrv*) installato sul sistema operativo ospitante al momento dell'installazione del programma. Questo driver permette di far credere, al sistema operativo della virtual machine, di essere eseguito a livello più alto possibile di privilegio del processore.

Ma, in queste circostanze, il sistema operativo della virtual machine, non essendo eseguito al giusto livello di privilegio, genera una serie di eccezioni, che devono essere opportunamente gestite, altrimenti si instaurerebbe un decremento nella velocità di esecuzione e nella fluidità di esecuzione del programma stesso. Per limitare *questo* tipo di inconveniente, il team di sviluppo di VirtualBox ha sviluppato due sistemi, chiamati: "Patch Manager" (PATM) e "Code Scanning and Analysis Manager" (CSAM). Queste sono tecnologie molto avanzate e complesse, che come abbiamo spiegato nel capitolo precedente, fanno parte delle soluzioni

di patching, in pratica permettono a VirtualBox di analizzare il codice prima di eseguirlo, trovando e sostituendo le istruzioni problematiche con altre gestibili dal programma stesso.

Una novità importante delle ultime relase è stata l'aggiunta del supporto a quelli che, sul sito ufficiale sono definiti come, *branched snapshot*s. Questa funzionalità permette di creare snapshot innestati. Gli *snapshot* sono letteralmente delle istantanee della virtual machine. In pratica creare una istantanea, vuol dire fare una fotografia del sistema in un dato momento storico, che può essere successivamente ripristinata. La possibilità di creare snapshot innestati, significa poter fare una istantanea di una istantanea, questi particolari snapshot sono effettuati dalle differenze degli uni rispetto agli altri. Inoltre dalla relase 3.2.2 è possibile effettuare i suddetti snapshot, anche mentre la virtual machine è in esecuzione.

Creare uno snapshot di una virtual machine può essere molto utile in ambito di produzione, subito prima di installare aggiornamenti di sistema, oppure nel caso in cui un utente stia per effettuare operazioni rischiose, che mettono in pericolo il corretto funzionamento del sistema operativo. Tra uno snapshot e l'altro eseguiremo qualunque operazione sulla virtual machine (installazione programmi o driver, aggiunta dispositivi) e come una specie di macchina del tempo, rievocando uno snapshot, la situazione precedente a qualunque operazione effettuata potrà essere ripristinata in ogni momento.

Oltre a quelle descritte sopra ci sono ancora numerose novità importanti che riguardano ad esempio le prestazioni delle VM. Sono state migliorate le prestazioni delle virtual machine a 64bit ed è stata aggiunta l'accelerazione video 2D per le macchine Windows. Inoltre, nelle ultime relase, tra le possibilità di scegliere il modo in cui la scheda di rete della virtual machine si interfacci con la macchina reale, in modalità: "Bridge", "NAT", "Rete interna", "Scheda solo Host" e "Non connessa" è stata aggiunta la possibilità di utilizzare una scheda VDE (Virtual Distributed Ethernet)[8][19], a patto che sulla macchina fisica sia installata la libreria "libvdeplug.so.2". VDE consente di creare reti virtuali più complesse,

[8] software realizzato dal prof.Renzo Davoli relatore di questa tesi

reti ibride (reali+virtuali), reti condivise tra diversi software di virtualizzazione (kvm, qemu, umlinux, bochs, etc.).

Ed infine, sempre nelle ultime relase disponibili, è reso addirittura possibile cambiare il tipo di connessione alla rete, mentre la virtual machine e in esecuzione.

Rispetto a KVM, VirtualBox è eseguibile su ogni sistema operativo (GNU Linux, Microsoft Windows e Mac OS) e non necessita di processori dotati di istruzioni VT-x o AMD-V. A livello prestazionale VirtualBox però soffre un po', rispetto ad altri sistemi di virtualizzazione come XEN o lo stesso KVM, a causa della sua architettura. Questi due prodotti infatti, rendono il sistema operativo ospitante consapevole del fatto che stanno eseguendo una virtual machine, e da ciò ne traggono vantaggio. C'è da dire però che tutti e tre questi sistemi di virtualizzazione stanno avendo una crescita sempre più veloce, dovuta probabilmente dal fatto che la *virtualizzazione* in se, comincia ad essere una soluzione appetibile ed adattabile a diverse situazioni. Basta pensare che il Team di VirtualBox nell'ultimo anno ha rilasciato ben 18 versioni, tra cui 3 importanti upgrade e l'acquisizione prima da parte di Sun e poi di Oracle.

2.3.1 Teleporting

In assoluto la novità più acclamata delle ultime release di VirtualBox è la già citata *Teleportation*. Questa funzione permette di effettuare una migrazione a caldo di virtual machine tra host diversi, senza perdita di continuità di servizio, a patto che su entrambi gli host sia in esecuzione una versione di VirtualBox 3.1 o superiore. La migrazione real-time, delle sessioni attive, si applica indipendentemente dal sistema operativo host (che deve comunque essere tra quelli supportati), ma è necessario che le due macchine possano comunicare via TCP/IP, abbiano accesso allo stesso network storage array iSCSI (o via NFS o Samba/CIFS)[9] e utilizzino una CPU "sufficientemente simile".

[9] per condividere le immagini dei dischi delle varie virtual machine da migrare

2.4 Citazioni su altri sistemi di virtualizzazione

Per dovere di cronaca faccio un rapido accenno ai maggiori sistemi di virtualizzazione esistenti, che però non sono stati presi in considerazione per una scelta implementativa, tra questi sistemi ci sono:

- **VMWare[?]** - Software proprietario, di cui l'approccio alla virtualizzazione consiste nell'inserimento di un sottile strato di software direttamente nell'hardware del computer o nel sistema operativo host. Tale strato di software crea le virtual machine e contiene un sistema di monitoraggio o "hypervisor" che alloca le risorse in maniera dinamica e trasparente per eseguire contemporaneamente più sistemi operativi indipendenti in un singolo computer fisico. La virtualizzazione di un singolo computer fisico è solo l'inizio: VMware offre una solida piattaforma di virtualizzazione in grado di supportare centinaia di computer fisici e dispositivi di storage interconnessi per la creazione di un'intera infrastruttura virtuale.

  Questo sistema rappresenta probabilmente la punta di diamante in questo settore, peccato che non sia prevista una versione open source, che l'avrebbero resa adatto al questo progetto.

- **OpenVZ[17]** - è la versione GPL su cui si basa Virtuozzo, il paravirtualizzatore di SWSoft, tra le sue caratteristiche ha il vincolo che sia il kernel del guest che quello dell'host devono essere opportunamente modificati e accetta solo kernel linux. Basa il suo funzionamento su Virtual Environment (VE) che rappresentano entità separate le une dalle altre. Ognuna di esse ha i propri: file, librerie, applicazioni, dati, utenti e gruppi (compreso root), spazio dei processi (compreso init), rete (compreso indirizzo IP e regole di firewalling e routing) e dispositivi. Il punto di forza di OpenVZ è la gestione delle risorse, ogni VE ha una quota di spazio disco ed è eseguito da uno scheduler fair-share (a livello di VE), inoltre ha associati dei beancounter, che permettono di controllare altri tipi di risorse in modo granulare (/proc/user_beancounter). Prevede la migrazione live trasferendo

un VE in funzione da un host ad un altro. Il trasferimento dei dati avviene via ssh.

- **Microsoft Virtual PC[12]** – Software di virtualizzazione proprietario gratuito di Microsoft. Emula vari sistemi operativi su Windows e Mac OS X. Esso consente l'esecuzione di sistemi operativi diversi, come varie versioni di Windows o GNU Linux, anche in contemporanea. L'emulatore ricrea in forma virtuale un ambiente di lavoro che riproduce quasi integralmente quello di un PC basato su Intel. Originariamente sviluppato dalla Connectix, attualmente il prodotto Virtual PC è distribuito da Microsoft, ed è destinato - nelle intenzioni della casa in questione - soprattutto a consentire l'uso di vecchie applicazioni non più supportate dai moderni sistemi operativi. La successiva introduzione dei processori Intel nei computer Apple ha tolto parecchio interesse pratico all'utilizzo di Virtual PC da parte degli utenti Macintosh, poiché è divenuto possibile riavviare (dual boot) tali elaboratori in Windows XP "nativo" (ossia senza ricorrere all'ausilio di un emulatore, con le limitazioni e le lentezze inevitabilmente legate a soluzioni di quel tipo) o utilizzare software di virtualizzazione più performanti come Parallels Desktop for Mac. Microsoft ha pertanto deciso nel 2006 di non sviluppare una versione di Virtual PC per i Mac con processore Intel.

Esistono molti altri sistemi di virtualizzazione, più o meno complessi, ma li tralascio per evitare un inutile lista di prodotti e funzionalità.

2.5 libvirt

Libvirt[4] è una libreria che fornisce un API per le funzionalità di virtualizzazione su GNU Linux e supporta molti hypervisor tra cui Xen, KVM, QEMU e VirtualBox.

Al suo interno sono integrate funzioni per gestire varie strategie di gestione remota, varie tecnologie di storage e vari aspetti delle virtual machine, fra cui la rete e i dischi.

In libvirt, col termine *nodo*, si indica la singola macchina reale (sistema host) su cui si vogliono eseguire le virtual machine (sistemi guest), col termine *hypervisor* si indica il virtualizzatore (o meglio: la tecnologia di virtualizzazione) impiegato, col termine *domain* si indica il sistema virtualizzato (il guest) che si vuole eseguire sull'hypervisor scelto.

Nelle premesse di questo sistema, appare chiaro l'intenzione dei programmatori di creare una astrazione, tra il virtualizzatore utilizzato e la gestione dei domini sui vari nodi. Il nodo da gestire può essere sia locale che remoto e ciò si realizza grazie ad appositi *uri*, i quali rappresentano le stringhe che permettono una connessione tra la libreria ed il virtualizzatore scelto. Ad esempio per collegarsi a VirtualBox localmente la stringa è "*vbox:///session*", mentre per collegarsi ad un nodo remoto tramite ssh, si dovrà inserire "*vbox+ssh://user@example.com/session*". Nel sito del progetto[10] risultano ben *documentate* tutte le opzioni possibili.

Sia l'organizzazione dei domini che l'organizzazione della rete, vengono gestiti tramite appositi file xml opportunamente strutturati. Stessa cosa per la gestione dello storage, organizzata attorno a due concetti chiave: quello di volume, una singola unità di storage basata su device o file opportunamente strutturati, e quello di pool, un apposito sistema in grado di gestire ed organizzare unità di storage in volumi.

Con questa libreria riusciamo a creare reti virtuali, concettualmente svincolate dal punto di vista fisico. Essa è basata essenzialmente su due concetti chiave: quello di *shared physical device*, ovvero un bridge contenete una delle interfacce reali sull'host, in modo che il dominio appaia direttamente sulla nostra LAN, e quello di *virtual network*, un bridge contente solo interfacce virtuali, per fornire qualsiasi connettività alla rete è necessario impostare opportune regole di forward, un caso particolare di questa tipologia di rete è l'*isolated networking*, tipologia in cui non è fornita alcun tipo di forward e le virtual machine possono dialogare solo fra di loro. Il tutto è facilmente gestibile tramite file xml opportunamente strutturati.

[10] http://libvirt.org/

Libvirt supporta vari linguaggi di programmazione tra cui si può citare C/C++, java, perl, python ed è disponibile su parecchi sistemi operativi. Ci sono svariate tipologie di software che utilizzano libvirt: ci sono software web based, ci sono applicazioni desktop e ci sono una quantità sempre crescente di applicazioni a riga di comando oltre a *virsh*, software fornito assieme alla libreria, che costituisce la management console base verso tutte le funzionalità offerte da libvirt.

Questa libreria anche se molto promettente non è stata utilizzata negli script di gestione per questo progetto, in quanto sebbene ad una prima analisi avrebbe consentito di creare un buon livello di astrazione, tra gli script di gestione e il sistema di virtualizzazione usato[11]. In realtà dopo vari esperimenti ciò non risultava possibile, in quanto anche se fosse stato realizzabile l'interfacciamento con diversi sistemi di virtualizzazione, la maggior parte delle funzionalità non risultava essere ugualmente implementata. Ad esempio proprio la migrazione, risultava essere utilizzabile per Xen e KVM, ma non per VirtualBox.

2.6 python

Infine, facciamo una breve introduzione a questo linguaggio, utilizzato per realizzare i vari script di gestione, sottolineandone la sua semplicità, ma nonostante ciò la sua potenzialità e flessibilità. Inoltre illustreremo le principali librerie utilizzate, al fine di risolvere le varie problematiche incontrate nello sviluppo di questo progetto.

Il linguaggio Python[18] nasce ad Amsterdam nel 1989, dove il suo creatore Guido Van Rossum lavorava come ricercatore. Nei suoi dieci anni di vita, si è diffuso in tutto il mondo.

Python è innanzitutto un linguaggio di script pseudo compilato. Questo significa che, similmente al Perl ed al Tcl/Tk, ogni programma sorgente deve essere pseudo compilato da un interprete. L'interprete è un normale programma che va

[11] infatti solo cambiando la stringa di connessione di questa libreria al sistema di virtualizzazione, si avrebbe avuto la possibilità di poter far funzionare questo progetto su diversi sistemi di virtualizzazione tra cui: Xen, KVM e per l'appunto VirtualBox.

installato sulla propria macchina, e si occuperà di interpretare il codice sorgente e di eseguirlo. Quindi, diversamente dal C++, non abbiamo un fase di compilazione - linking che trasforma il sorgente in eseguibile, ma avremo a disposizione solo il sorgente che viene eseguito dall'interprete.

Il principale vantaggio di questo sistema è la portabilità: lo stesso programma potrà girare su una piattaforma GNU Linux, Mac o Windows, purché vi sia installato l'interprete, si ha la possibilità di avere un interprete per una vastissima lista di piattaforme, ne esistono per symbian, per android e per parecchi dispositivi embledded.

Python è un linguaggio orientato agli oggetti. Supporta le classi, l'ereditarietà e si caratterizza per il binding dinamico. Ragionando in termini di C++ potremo dire che tutte le funzioni sono virtuali. La memoria viene gestita automaticamente e non esistono specifici costruttori o distruttori; inoltre esistono diversi costrutti per la gestione delle eccezioni.

Un altro importante elemento per inquadrare Python è la facilità di apprendimento. Chiunque nell'arco di un paio di giornate può imparare ad usarlo e a scrivere le sue prime applicazioni. In questo ambito gioca un ruolo fondamentale la struttura aperta del linguaggio, priva di dichiarazioni ridondanti e estremamente simile ad un linguaggio parlato. L'indentazione perde il suo ruolo inteso come stile di buona programmazione, per facilitare la lettura del codice, per diventare parte integrante della programmazione che consente di suddividere il codice in blocchi logici.

Per quando riguarda il progetto di questa tesi, sono state sfruttate le seguenti librerie: la "vboxapi" che rappresenta una api per interfacciare uno script python con *VirtualBox* e le virtual machine con esso create; "paramiko", che consente di instaurare una connessione ssh tra due host, gestendo l'utilizzo di una coppia di chiavi privata/pubblica, in modo da realizzare una connessione sicura, senza scambio di password; "xml.dom.minidom" necessaria per la gestione dei file xml. Comunque il tutto sarà opportunamente analizzato nel capitolo successivo.

Capitolo 3

Implementazione

Dopo aver illustrato gli strumenti utilizzati per la realizzazione di questa tesi, iniziamo a trattare l'implementazione vera e propria.

Nella realizzazione di alcune funzionalità, si è cercato di adottare il metodo del "*riutilizzo*", cercando in rete un pezzo di codice già fatto che almeno per grandi linee risolvesse un particolare problema o comunque una parte di esso, anche se per determinate funzionalità, sono state necessarie delle scelte implementative "nuove". Nonostante ciò, il sistema di scomporre un problema in sotto-problemi e risolvere quest'ultimi con appositi metodi, ha permesso la risoluzione di problemi simili adottando i medesimi metodi, senza andare ogni volta a creare una soluzione diversa.

Entrando nel dettaglio, iniziamo fornendo la nomenclatura usata per invocare i vari elementi per questo sistema. Prima di tutto abbiamo il "*cluster*" che rappresenta una sorta di aggregatore di tutte le macchine fisiche che lo compongono, che chiameremo "*host*" o "*nodi*", su quest'ultime saranno in esecuzione delle virtual machine, dette "*guest*" o "*VM*". Già basandosi su questa prima classificazione sono state realizzate tre classi, che vengono definite di costruzione e chiamate appunto: "*Cluster*", "*Host*" e "*VM*", le stesse rappresentano gli oggetti principali da manovrare durante l'esecuzione degli script di gestione.

In pratica il funzionamento del sistema, prevede che inizialmente vengano aggiunti i nodi al cluster, in modo da conoscerne il numero e le caratteristiche. Tale

aggiunta comporta la creazione di un file xml (il cui nome è l' indirizzo IP .xml) in cui sono elencate le proprietà del nodo e l'aggiornamento di un file generale (ClusterInfo.xml) che raggruppa tutte le caratteristiche del cluster. Successivamente si avvia, per ogni nodo, una VM che rappresenta il clone del nodo su cui viene eseguita.

Finita questa procedura per tutti i nodi del cluster, il sistema si mette in attesa del vero e proprio funzionamento, che viene attivato quando il laboratorio giunge all'orario di chiusura. Quando si verifica tale evento entra in funzione una quarta classe di gestione, chiamata "*Manager*", che rappresenta il cuore del sistema e si occupa di verificare lo stato dei vari nodi e quando si verificano determinati eventi, gestisce la migrazione delle virtual machine da un host all'altro, cercando di tenere il maggior numero di VM in esecuzione sul minor numero di nodi. L'effetto principale di questa procedura è rappresentato dal fatto che i nodi ai quali non rimane in esecuzione nessuna VM, a causa delle migrazioni viste prima, vengono spenti e ciò consente il risparmio energetico che ci siamo prefissi. Nonostante tutto però, i nodi spenti rimangono nella configurazione del cluster, in modo tale che se vengano mutate le caratteristiche, per un eventuale overload di qualche nodo, si può disporre di quei nodi spenti, riavviandoli e assegnandogli, tramite migrazione, una o più VM.

Si è scelto di assegnare il ruolo di nodo manager al primo nodo che viene inserito nel cluster, in modo da non vincolarne la definizione. In pratica su tale nodo ogni 15 minuti viene eseguito un aggiornamento delle caratteristiche dell'intero cluster e nel caso ci fossero host in overload provvede alle opportune migrazioni. Per gestire il fatto che il nodo manager potrebbe guastarsi pregiudicando il funzionamento dell'intero cluster, è stato creato uno script, che viene eseguito da tutti i nodi ogni 30 minuti e che verifica da quanto tempo il file "ClusterInfo.xml" è stato modificato, se tale intervallo è superiore a 15 minuti, vuol dire che il nodo manager non funziona e bisogna avviare lo script di gestione del cluster su un altro nodo, scegliendone il primo della lista dei nodi. In pratica ogni nodo, una volta appurato il fault, verifica se è il primo della lista, in tal caso si elegge manager ed avvia il relativo script.

Alla riapertura del laboratorio, viene sospesa la procedura di compattamento e si ha una normalizzazione del sistema, che riporta l'intero cluster alle caratteristiche di partenza, cioè tutti i nodi accesi con ognuno di essi che esegue almeno una VM.

Esiste un file di configurazione in cui è possibile modificare le impostazioni che riguardano il funzionamento del progetto, ad esempio cambiare la path dove sono inseriti tutti gli script[1], oppure modificare la quantità minima di memoria da assegnare alle VM, o altre impostazioni che verranno illustrate nei prossimi paragrafi.

Per la realizzazione di questo progetto è stato utilizzato un ambiente di test, implementato attraverso due livelli di virtualizzazione. In pratica su un unico elaboratore è stato installato VirtualBox e tramite esso sono state create delle virtual machine, che rappresentavano gli host fisici del laboratorio. Utilizzando queste VM come se fossero delle macchine fisiche, si è potuto fare un accettabile test del funzionamento del cluster su descritto (vedi fig.3.1). La differenza più evidente tra questo ambiente di test e la situazione reale è dovuta al fatto che, il metodo che si occupa del risveglio di un nodo spento, non utilizza la modalità wake up on lan, che verrà meglio descritta in seguito, ma semplicemente riavvia la VM che simula tale nodo.

3.1 Le classi di costruzione

Le tre classi seguenti formano il cosiddetto effetto matrioska, infatti ognuna di esse, come vengono elencate di seguito, rappresenta una gerarchia man mano crescente, che culmina nella classe Cluster, la quale rappresenta l'oggetto che prende in considerazione tutti gli aspetti del sistema.

[1] che comunque deve essere in una cartella condivisa con tutti i nodi del cluster, affinché il sistema possa funzionare

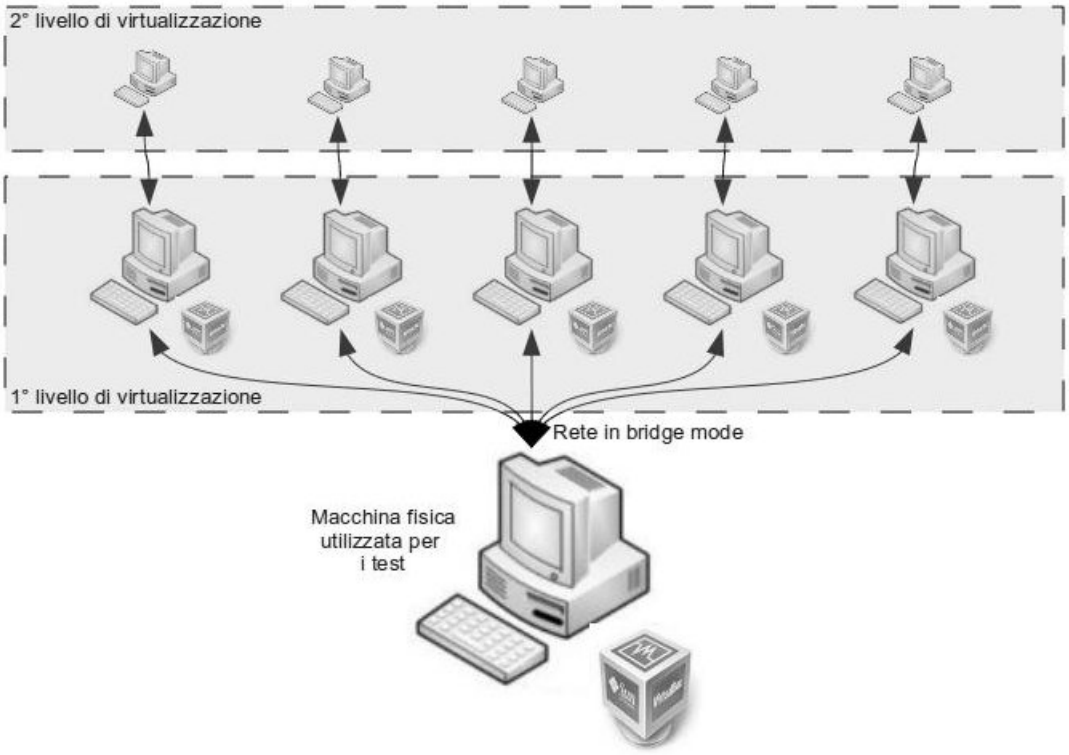

Figura 3.1: Ambiente di test, per la verifica del funzionamento degli script di gestione

3.1.1 La classe VM

Questa classe fornisce una serie di metodi, che permettono di realizzare varie funzionalità, per quanto riguarda la gestione delle virtual machine, i principali sono:

insert permette di creare una nuova virtual machine, chiedendo all'utente di inserire le proprietà della nuova virtual machine, ad esempio il nome, la quantità di memoria ram etc. Dopo ogni inserimento vengono effettuati alcuni controlli per verificare la validità dei valori immessi, alla fine viene stampato a video un riepilogo delle informazioni inserite e se si decide di continuare, viene creata una nuova virtual machine. Inoltre viene salvata una copia del file xml, che VirtualBox crea nella definizione di una nuova virtual machine, nella parte condivisa in un

apposita cartella chiamata "*VM*". Questo file, come ho accennato nel capitolo precedente, verrà utilizzato per registrare una virtual machine su un altro host, attraverso un apposito metodo che illustrerò in seguito.

Questa soluzione è stata adottata in quanto se creiamo una virtual machine su un host e vogliamo che la stessa virtual machine sia registrata anche su un altro host, non basta condividere l'immagine del disco, ma è necessario che entrambi le virtual machine abbiano lo stesso mac-address, altrimenti nella migrazione perdono il segnale di rete.

add_machine consente di creare la virtual machine servendosi dei valori ricavati dal metodo precedente

del_machine allo stesso modo viene data la possibilità di eliminare una virtual machine dall'elenco di quelle registrate.

start_VM permette come dice il nome di avviare una virtual machine. Per far ciò utilizziamo la libreria "*vboxapi*", che ci consente di creare una connessione con VirtualBox e dopo aver verificato la presenza tra le virtual machine registrare, di avviarla.

add_machine_cluster proprio sfruttando i file xml salvati subito dopo la creazione di una nuova VM, di cui si parlava prima, permette di registrare in un altro host, anche remoto, la virtual machine relativa a tale file xml.

3.1.2 La classe Host

Questa classe risulta molto utile, perché consente di semplificare alcune operazioni. Ricordando che ogni volta che aggiungiamo un host al cluster e successivamente ad ogni aggiornamento del cluster[2], generiamo un file xml[3] in cui salviamo

[2] il refresh dell'intero Cluster avviene mediamente ogni 15 minuti, nei periodi di chiusura del laboratorio e quindi di attività del gestore del cluster

[3] che ha come nome l'indirizzo IP di quell'host

tutte le caratteristiche dell'host, quali ad esempio: l'hostname, l'indirizzo IP ed il mac address[4], lo stato della macchina ("*Running*" o "*Off*"), il suo ruolo ("*Manager*" o "*Node*"), il valore del loadavg15 (che rappresenta il carico medio del processore negli ultimi 15 minuti[5]), la memoria ram, il numero di cpu, la percentuale istantanea della cpu, il numero massimo di virtual machine avviabili (tale valore, viene ricavato dalla quantità di memoria dell'host, diviso una quantità minima di ram da assegnare alle varie virtual machine, valore che viene fissato a 256 MB, ma che può essere facilmente modificabile dal file di configurazione "*config.py*"), infine si hanno la lista di tutte le virtual machine in esecuzione su tale host, con ulteriori caratteristiche di queste ultime. Questo file xml viene in sostanza parsato dalla classe Host, che ne crea, appunto un oggetto, in modo da renderlo più gestibile per quanto riguarda l'acquisizione delle varie informazioni.

Oltre questa caratteristica, tramite il metodo "**pretty_print**" si produce un output graficamente più sintetico, che riepiloga le caratteristiche del host in esame, permettendo di listarlo a video.

3.1.3 La classe Cluster

Questa rappresenta la più importante delle tre classi fin'ora enunciate, infatti permette di creare un oggetto Cluster che raggruppa i precedenti due. Di seguito vengono elencati i principali metodi di questa classe, dandone una breve descrizione:

add_node come si può facilmente intuire dal nome, permette di inserire nella struttura del cluster un nuovo host, questo significa in sostanza generare il file xml, di cui si parlava prima, con le caratteristiche dell'host, nell'apposita cartella condivisa chiamata *"Nodes/"* ed infine aggiungere tale file all'interno del file generale "*ClusterInfo.xml*", che come ho anticipato nel capitolo precedente, rappresenta la fusione di tutti i

[4]che verrà utilizzato per il risveglio degli host spenti attraverso la funzione wake up on lan

[5]proprio leggendo questo valore, verrà deciso se migrare o meno le virtual machine in esecuzione

vari file xml dei vari host che compongono il cluster, più qualche altra informazione generale.

Prima di eseguire le suddette operazioni, questo metodo si deve occupare di una serie di controlli, quali ad esempio verificare che l'host da aggiungere sia raggiungibile, infatti non è permesso aggiungere un host spento oppure che non sia pingabile; ancora verifica se tale host è già presente nel cluster, infine verifica se è il primo host ad essere aggiunto ed in tal caso lo nomina "*Manager*". Questa funzionalità è dettata dal fatto che cosi facendo si ha una nomina del Manager abbastanza casuale, in modo da non obbligare una particolare macchina ad eseguire tale compito. In realtà il nodo Manager non fa altro che avviare un script di gestione periodicamente tramite un "*crontab*", ma ha delle limitazioni e dei vantaggi, infatti non può essere spento, anche se non sta eseguendo nessuna virtual machine[6].

del_node allo stesso modo, con questo metodo, viene eliminato un nodo dal cluster, cancellato il suo file xml ed inoltre viene aggiornato anche il file xml del cluster.

reset consente, come suggerisce il nome, di resettare l'intero cluster, cioè elimina tutti i nodi dal cluster ed elimina tutti i file xml, lasciando solo quello del cluster, ma svuotato dai riferimenti ai vari nodi

refresh rappresenta il metodo che aggiorna lo stato del cluster, anche se intuitivamente può sembrare una cosa scontata, non è stato per niente facile realizzare questo metodo, in quanto bisogna tenere in considerazione una serie di circostanze. Per tale motivo è stato deciso di suddividere questo metodo in altri due sotto metodi che vengono invocati quando necessari. In pratica questo metodo si occupa di verificare se le informazioni contenute nel file ClusterInfo.xml sono vere ed in caso

[6] a differenza degli altri nodi del cluster, che invece vengono spenti se non stanno eseguendo nessuna VM

contrario cerca di risolverne i conflitti, inoltre chiede a tutti i nodi di generare un nuovo file xml con le loro caratteristiche. Questa richiesta inizialmente impiegava qualche secondo per ogni nodo. Considerando che a regime avrebbe dovuto aggiornare 62 nodi, il tempo di attesa sarebbe stato insostenibile per l'andamento degli altri script di gestione. Quindi è stata fatta la scelta di usare il multithreading che consente di creare più thread ed ognuno di esso viene istruito affinché si occupi di un solo host, in questo modo si è riuscito a ridurre notevolmente il tempo di aggiornamento.

3.2 La classe di gestione: Manager

Questa ulteriore classe rappresenta il cuore dell'intero progetto, infatti grazie ad essa vengono gestite le migrazioni delle virtual machine da un host all'altro. Tali migrazioni vengono decise principalmente a seconda dello stato dell'host e tale caratteristica si ricava analizzando il valore di loadavg15.

3.2.1 Loadavg15

Loadavg15, rappresenta un elemento importante per il funzionamento di tutto il sistema, infatti grazie al suo valore vengono prese delle decisioni sull'evoluzione del cluster. Il suo valore viene ricavato semplicemente prelevando l'ultimo valore che fornisce il comando linux "*uptime*"[7] e rappresenta il carico medio della macchina negli ultimi 15 minuti. Questo valore deve però essere interpretato, infatti in sistemi single-CPU, esso può essere considerato come il carico medio in percentuale di utilizzo del sistema, durante il periodo di tempo corrispondente, ma in sistemi multi-CPU[8], si deve dividere il suo valore per il numero di processori, al fine di ottenere una percentuale analoga.

[7] che a sua volta recupera tale valore dal file */proc/loadavg*
[8] come nel nostro caso, si tratta di cpu dual core

Ad esempio, provando ad interpretare i valori di loadavg: 1.50 0.25 5.68 che rappresenta il carico medio del sistema rispettivamente negli ultimi: 1, 5 e 15 minuti, in un sistema single-CPU:

- durante l'ultimo minuto la CPU è overloaded del 50%, in quanto c'è 1 CPU con 1.50 processi in esecuzione, quindi 0.50 processi hanno dovuto attendere per un turno;
- durante gli ultimi 5 minuti la CPU è underloaded del 50%, in quanto tutti i processi sono stati eseguiti in un turno;
- durante gli ultimi 15 minuti la CPU è overloaded del 468%, in quanto 4.68 processi hanno dovuto attendere per più di un turno.

In un sistema multi-CPU lo stesso valore sarebbe stato interpretato diversamente, ad esempio in un sistema a 2 CPU, quindi bisogna dividere per 2 i valori, prima di analizzarli, quindi diventano 0.75 0.12 2.84:

- durante l'ultimo minuto la CPU è underloaded del 75%, in quanto ci sono 2 CPU con 1.50 processi in esecuzione, quindi tutti i processi sono stati eseguiti in un turno;
- durante gli ultimi 5 minuti la CPU è underloaded del 12%, in quanto tutti i processi sono stati eseguiti in un turno;
- durante gli ultimi 15 minuti la CPU è overloaded del 284%, in quanto per ogni processore ci sono 2.84 processi.

3.2.2 Classe Manager

La prima attività che viene svolta da questa classe è quella di creare quattro liste di IP degli host presenti nel cluster, che ci permetteranno di effettuare le successive migrazioni e compattamenti, tali liste risultano suddivise in questo modo:

list_under a questa lista vengono aggiunti tutti gli host che hanno un valore di loadavg15 inferiore ad una data soglia[9], che rappresenta la soglia minima di carico per il processore. Da questa lista sono esclusi però, gli host che seppure presentano un carico inferiore alla detta soglia, hanno in esecuzione un numero di virtual machine uguale al numero massimo di VM avviabili. In tale caso, l'host in esame viene aggiunto alla lista "*list-regular*". Questo perché su un host con tali caratteristiche non si possono migrare altre VM;

list_over a questa lista appartengono tutti gli host che superano la soglia max[10], ad esclusione di quelli che hanno in esecuzione una sola VM. Infatti se un host ha una sola VM ed è in overload, è inutile migrare quell'unica VM, sposterebbe soltanto il problema ad un altro host, che magari potrebbe avere in esecuzione già altre VM, creando quindi un loop di migrazione. In tal caso quindi, l'host che presenta queste caratteristiche viene aggiunto alla lista "*list_regular*";

list_regular oltre a comprendere gli host che si trovano nelle condizioni dei due casi visti prima, contiene tutti gli host che presentano un valore di loadavg15 compreso tra la soglia minima e quella massima. Gli host appartenenti a questa lista, fanno parte di una specie di limbo, ragion per cui vengono lasciati cosi come sono, in attesa che passino nella prima o nella seconda lista ed in tal caso vengano prese le opportune misure, che vederemo tra poco;

list_off a quest'ultima lista fanno parte gli host che sono già spenti, ma comunque fanno parte del cluster ed in più tutti gli host che farebbero parte della lista "*list_under*", ma che non hanno in esecuzione nessu-

[9] modificabile nel file di configurazione config.py
[10] come per la minima, anche questo valore risiede nel suddetto file di configurazione

na VM[11]. Infatti è inutile tenere acceso un host se non sta eseguendo nessuna VM.

Risulta chiaro, che quest'ultima lista rappresenta la parte che comporta il risparmio energetico, infatti permette di spegnere quello che "non serve" al momento. Le prime due liste invece sono i sensori del cluster e a seconda della quantità di host che contengono l'una o l'altra, si decide sulle migrazioni da adottare. list_regular, come abbiamo già detto è una lista neutra, qui vengono aggiunti gli host per cui non conviene effettuare nessuna migrazione.

Una volta generate le suddette liste, vengono elaborate da un apposito metodo "*mind*"[12], che controllandone le dimensioni, prende le opportune decisioni. Prima di tutto verifica se list_over è vuota, in tal caso sa che non ci sono macchine in overload e può procedere alla compattazione, che consiste in sintesi, nel migrare il maggior numero di VM su il minor numero di host e spegnere quelli che rimangono senza VM. Chiaramente questo meccanismo viene avviato soltanto sugli host di list_under e soltanto nel caso in cui il numero di host che la compongono sia superiore ad uno.

Se invece list_over non risulta essere vuota, per le condizioni imposte prima, sappiamo che ogni host appartenete a questa lista ha in esecuzione un numero di VM che sicuramente sarà superiore ad uno, quindi per cercare di abbassare l'overload di queste macchine, si procede alla migrazione di una VM da questi host agli host di list_under, sempre che quest'ultima contenga qualche host. In caso contrario si controlla la list_off e se dispone di qualche macchina, se ne accende una, per poi effettuare la migrazione verso di essa.

C'è da precisare, che quando si concretizza la condizione per la quale risulta possibile la migrazione, per cui abbiamo un host sorgente con n VM in esecuzione ed un host destinazione, si presentava il problema del modo in cui scegliere la VM da migrare. Per far ciò viene utilizzato un apposito metodo chiamato "*min_guest*" che preso come argomento l'intera lista di VM dell'host sorgente, restituisce il

[11] condizione che si verifica quando le VM in esecuzione su tale host vengono migrate ad altri host

[12] metaforicamente rappresenta il cervello che decide sulle migrazioni da fare.

nome della VM più leggera, intesa come la VM che al momento, risulta avere la percentuale media di utilizzo cpu più bassa[13]. In questo modo si cerca di evitare il problema per cui migrando una VM da un host in overload ad uno in underload, non si fa altro che ribaltare i ruoli e quindi c'è il rischio che si inneschi una serie di migrazioni incontrollate.

Facendo un passo indietro, la fase della compattazione viene effettuata in più cicli e ad ogni ciclo viene analizzata list_under e viene effettuata una migrazione della VM "più leggera" tra l'ultimo host della lista ed il primo, tra il penultimo ed il secondo e cosi via, fino ad arrivare agli host centrali. A quel punto si rigenerano le 4 liste e si verificano di nuovo le condizioni. In questo modo si cerca di adattare le soluzioni, all'evoluzione dell'intero cluster per gradi, valutando se continuare con la scelta iniziale o modificare la rotta.

In linea puramente teorica, se le fasi di compattazione vengono portate tutte a termine, si arriva alla soluzione ideale in cui rimangono accesi solo 8 host ed ognuno di esso esegue, mediamente, 8 virtual machine. I restanti 54 host vengono spenti ed inseriti in list_off, pronti ad essere riaccesi, qualora se ne presentasse la necessità. Infatti costantemente lo script Manager provvede a controllare il carico di lavoro di tutti gli host accesi e nel caso in cui si dovesse verificare le idonee condizione, provvede alla riaccensione di uno o più host ed alla successiva migrazione di VM.

Si ricorda che per riaccendere un host dalla list_off, viene utilizzato il metodo "*poweron*" che sfrutta la funzionalità data dal pacchetto *"ethtool"*, la quale grazie a due semplici comandi, permette di configurare un host alla ricezione del cosiddetto *"MagicPacket"* ed un altro host all'invio dello stesso. La tecnologia Magic Packet consiste nell'invio di un pacchetto di dati direttamente ad un sistema. Questo pacchetto è formato dalla ripetizione dell'indirizzo di livello MAC (media access control) di sistema per 16 volte. L'indirizzo MAC è specifico per l'adattatore di rete nel sistema in modo che il Magic Packet risvegli solo il sistema che si vuole avviare. Quando l'adattatore di rete riceve e decodifica questo pacchetto, invia al sistema un segnale di evento di gestione dell'alimentazione

[13] valore che vine ricavato dal file xml dell'host in questione

(PME, power management event) che lo riaccende completamente e lo avvia. Il valore del mac-address dell'host da riaccendere è facilmente recuperabile dal file xml relativo a quell'host.

3.3 Normalizza

Come abbiamo già detto ogni 15 minuti viene aggiornato l'intero cluster, rigenerate le quattro liste che abbiamo visto prima e gestite le eventuali migrazioni, tutto questo viene effettuato fino al raggiungimento dell'orario di riapertura del laboratorio. Mezz'ora prima di tale orario, lo script Manager viene interrotto e viene eseguito un apposito script "*Normalize*" che, come suggerisce la parola, si occupa di riportare la situazione di funzionamento del laboratorio alla normalità, cioè accendere tutti gli host e verificare che su ogni host sia in esecuzione almeno una VM.

Queste operazioni vengono semplicemente realizzate in due fasi: prima vengono accessi tutti gli eventuali host spenti, con l'apposito metodo visto prima "poweron" si accendono tutti gli host presenti in list_off e si inseriscono in una lista temporanea gli indirizzi IP, successivamente vengono esaminati tutti gli host e quando si incontra un host che ha in esecuzione più di una VM, la si migra su un host presente nella lista temporanea e contestualmente si elimina l'host da tale lista. In tal modo, alla fine della scansione dell'intera lista degli host, si avrà la situazione in cui tutti gli host sono accessi ed eseguono almeno una VM.

È facile costatare che questo sistema permette di avere subito una nuova funzionalità che consiste nel fatto che anche durante il periodo di apertura del laboratorio, nel caso in cui un host fisico si dovesse guastare, avremo la possibilità che il suo clone possa essere in esecuzione su un altro host ed anche se fisicamente non è disponibile, rimane disponibile da remoto attraverso il suo clone.

A questo punto si attende la nuova chiusura del laboratorio, per rieseguire lo script Manager e la conseguente ottimizzazione nei consumi.

3.4 Utility

Oltre ai moduli su descritti, esistono altri due moduli in cui sono compresi tutti i metodi di utilità impiegati dagli altri script. Il primo modulo è chiamato *"util_xml"* e si occupa esclusivamente della gestione dei file xml, fornendo i metodi necessario a parsare tali file, crearne nuovi, aggiungere o eliminare tag ed infine salvare un documento dom in un file xml. Proprio per quest'ultima utilità che prevede il salvataggio di una stringa in un file xml, ho inserito un formattatore che si occupa di gestire le giuste indentazione del codice xml prima di salvarlo in un file, in modo tale da dare un aspetto esteticamente e funzionalmente più corretto a tali tipi di file.

Il secondo modulo invece raggruppa una serie di funzioni che assolvono ai più disparati utilizzi, ad esempio si hanno metodi che ricavano informazioni relative alla macchina su cui sono eseguiti, come: hostname, indirizzo IP, mac-address, loadavg15 etc; poi abbiamo altri metodi che si occupano della validazione dei dati immessi, quando si vuole creare una nuova virtual machine; infine abbiamo il metodo che si occupa di eseguire un comando su un host remoto, tramite una connessione ssh, tale metodo è chiamato *"remotecmd"*

3.4.1 remotecmd

Questo metodo è molto importante e viene utilizzato in modo massiccio in tutti i moduli di questo progetto, esso utilizza la libreria python "paramiko" che consente di gestire una connessione ssh in modo abbastanza semplice, inoltre per aumentarne la sicurezza ed evitare che vengano inviate in chiaro le password dei vari host, tale libreria consente di effettuare una connessione ssh sfruttando una coppia di chiavi privata/pubblica, in questo modo viene prima generata tale coppia di chiavi sull'host che dovrà eseguire tale metodo, poi verrà inviata la chiave pubblica all'host da contattare e da quel momento si può instaurare una connessione ssh senza la necessità di inserire alcuna password.

Per realizzare tale condizione, premesso che, come abbiamo detto prima, non abbiamo una macchina che viene definita Master, ma volta per volta potrebbe es-

sere un qualsiasi host del cluster ad assolvere tale compito, generiamo la suddetta coppia di chiavi su ogni host, quando viene eseguito per la prima volta una connessione ssh tra due host, il metodo remotecmd verifica se le chiavi sono state già scambiate, in caso negativo provvede allo scambio. Questa soluzione ci consente di non inserire nel codice la password di connessione in chiaro, con gli evidenti problemi di sicurezza.

Ritornando al metodo in esame, esso è costituito da due parti, in cui la prima si occupa di effettuare una connessione sicura ssh tra l'host che esegue questo metodo e l'host target e la seconda parte si occupa di eseguire il comando che gli viene passato come argomento sull'host target. Entrambi queste parti utilizzando appunto la libreria *paramiko* per assolvere le loro funzioni.

3.5 Cluster_shell

Questo ultimo metodo è stato inserito per creare una interfaccia tra l'utente ed il sistema, in modo da semplificarne e guidarne l'utilizzo. In pratica sfruttando le funzionalità della libreria python "*cmd*" viene realizzata una shell "*cluster*>" che permette di eseguire tutti i comandi previsti per la gestione del cluster, utilizzando i metodi visti prima. Inoltre prevede un comando "*help*" attraverso il quale possiamo conoscere in che modo invocare tali metodi, grazie ad una sintetica lista di comandi, con una breve descrizione.

In questo modo, attraverso un unico script da eseguire, si da la possibilità all'utente di interagire con le varie funzionalità che questo sistema propone, inoltre semplifica la gestione del controllo dei dati inseriti dall'utente, in modo da limitarne l'uso improprio.

C'è da sottolineare una funzione di questa shell chiamata "*list*" che in pratica stampa a video una lista degli host che compongono il cluster specificandone le caratteristiche principali, se però alla stessa funzione vengono passati determinati argomenti si ottengono altre informazioni utili sullo stato di tutte le macchine che compongono il cluster. Ad esempio "*list VM*" restituisce la lista di tutte le virtual machine in esecuzione su tutti gli host, in questo modo è possibile avere una vi-

sione sul numero e il nome delle VM in esecuzione sul cluster; "*list nome_del_host*" oppure "*list ip_del_host*" invece restituisce tutte le caratteristiche relative all'host passato come argomento, compresa la lista delle VM in esecuzione su di esso; ancora "*list Off*" lista tutti gli host presenti nel cluster, ma che sono spenti.

Un altra utile funzione nella gestione di questo cluster è rappresentata da "*findVM*" che permette di cercare nell'intero cluster la presenza della VM passata come argomento ed in caso positivo viene fornito anche l'indirizzo IP dell'host che la sta eseguendo. In tal modo in caso di malfunzionamento o di manutenzione specifica si può cercare su quale host è in esecuzione una data VM ed agire di conseguenza.

In questo script in definita, sfruttando le funzionalità già presenti, risulta molto facile aggiungere nuove funzionalità dettate da particolari esigenze che potrebbero insorgere nell'utilizzo di questo sistema, in modo da facilitarne la manutenzione. Ad esempio potrebbe essere utile inserire un sistema di logging del cluster, per realizzarlo basterebbe semplicemente prendere periodicamente il risultato del comando list ed inserirlo in un file.

Capitolo 4

Conclusioni e Sviluppi Futuri

Questo lavoro di tesi, ha permesso di analizzare la possibilità di sfruttare la virtualizzazione per riconfigurare i computer in dotazione al laboratorio informatico della facoltà, con l'obiettivo del risparmio energetico, realizzabile in particolari condizioni, consentendo lo spegnimento fino all'80% delle macchine fisiche presenti nello stesso.

Il punto di partenza di questo progetto si basa sulla tecnica conosciuta come "*Server consolidation*[1]" che permette di avere su un unica macchina più servizi, proprio sfruttandone la virtualizzazione, per poi ampliarne le funzionalità, passando da una visione statica del problema ad una dinamica. Non c'è bisogno di decidere quali macchine fisiche tenere accese e di quante virtual machine si ha bisogno per garantirne le stesse funzionalità, ma autonomamente il sistema, in base al carico di lavoro, decide di migrare virtual machine da un host all'altro, al fine di garantire gli stessi servizi, ma con un numero minore di macchine fisiche.

Gli esperimenti effettuati grazie a questo progetto hanno permesso di valutare alcuni risultati ottenuti. Analizzando la fase di attività del progetto in condizioni di basso carico, considerando che nel laboratorio ci sono 62 macchine con 2GB di Ram ciascuna ed assegnando ai 62 cloni virtuali 256MB di Ram, si ottiene che ogni macchina può eseguire fino ad 8 virtual machine con una fluidità di esecu-

[1] è un approccio all'uso più efficiente delle risorse nell'ottica della riduzione del numero totale di server necessari ad una azienda

zione accettabile. Inoltre i tempi necessari alla migrazione di una virtual machine da un host all'altro sono relativamente bassi, nell'ordine di qualche secondo, anche se dipende molto dal tipo di rete che collega i due host e dal carico di lavoro che sta eseguendo la VM durante la migrazione. Comunque mediamente in una decina di minuti si passa dalla situazione in cui tutti gli host sono accessi a quella in cui si ha il massimo del risparmio energetico con solo 8 macchine fisiche accese che eseguono 8 VM ciascuna. Chiaramente questi risultati sono stati ottenuti nell'ambiente di test descritto all'inizio del capitolo precedente e potrebbero aver dato una visione ottimistica rispetto ai risultati che materialmente si otterrebbero in un ambiente reale, ma nonostante ciò tali risultati dimostrano che l'adozione di questo sistema comporterebbe notevoli vantaggi sotto il punto di vista del risparmio energetico.

Come abbiamo già accennato nei capitoli precedenti, ogni vantaggio porta con se i suoi compromessi ed anche in questa circostanza ne dobbiamo tener conto. Considerando la situazione iniziale, su cui è stato impostato il lavoro di tesi, siamo in presenza di un laboratorio con 62 computer, la premessa per il funzionamento del sistema è che su ognuno di esso, venga eseguita una virtual machine con caratteristiche simili a quelle dell'host fisico sottostante, che garantirà la connessione con un utente remoto. Risulta evidente il passaggio da una situazione in cui era necessario gestire 62 macchine, ad una in cui bisognerà gestirne 124 (62 reali e 62 virtuali), con i vari problemi di aggiornamento e manutenzione vari. E' pur vero che gestire una virtual machine e molto più semplice che gestirne l'equivalente reale ed inoltre, esistono vari tool che permettono un automatismo per queste operazioni. Comunque questa rappresenta, sicuramente, una complicazione che bisogna prendere in esame.

Bisogna tener presente anche, che un utente remoto non interagirà più con una macchina fisica, ma con una virtuale, che avrà un livello prestazionale sicuramente più basso, ma c'è da dire che oltre alla velocità di esecuzione della macchina remota, ciò che ne limita maggiormente ed in maniera predominante le prestazioni è la rete di collegamento. In definitiva, si potrebbe aver a disposizione un computer potentissimo, ma se un utente si collega a tale computer attraverso

una rete lenta otterrà un rallentamento anche nella sua operatività.

Infine, valutando i vari fattori che potrebbero condizionare il funzionamento del progetto, abbiamo analizzato che statisticamente non esiste un grande traffico di utilizzo delle macchine del laboratorio da remoto, questa condizione fa si che durante il periodo di chiusura del laboratorio, in presenza di una bassa o nulla utilizzazione delle macchine da remoto, si ottenga una drastica riduzione di macchine fisiche accese, con i conseguenti vantaggi. Situazione che non risulterebbe vera in altre condizioni, ad esempio se tutte le macchine fossero sottoposte ad un carico costante, non si concretizzerebbero mai le condizioni adatte alla migrazione e quindi non si otterrebbe vantaggio alcuno.

Come prerequisiti del suo funzionamento, questo progetto, richiede una semplice connessione TCP/IP tra le macchine fisiche, per consentire la migrazione di una virtual machine ed uno storage condiviso. Quindi si può aggiungere una macchina al cluster semplicemente se questa è raggiungibile dalle restanti macchine del cluster ed ottenerne cosi una massima scalabilità del sistema. Ad esempio potrebbero essere aggiunte al cluster le macchine che fanno parte del laboratorio “Ranzani”, senza apportare grandi modifiche a tali macchine.

Gli sviluppi futuri a questo lavoro di tesi sono molteplici, ad esempio, se venisse modificata la struttura attuale del laboratorio, si potrebbe pensare di creare una situazione secondo cui ad ogni utente che si connette ad una macchina (localmente o da remoto), venga assegnata una virtual machine, che lo stesso potrebbe personalizzare a suo piacimento. Potrebbe poi, salvarne l’immagine su una chiavetta usb e continuare a casa a lavorare sulla sua VM, inoltre sfruttando una connessione VDE, potrebbe addirittura lavorare alla sua VM sulla rete del laboratorio come se fosse fisicamente connesso alla stessa.

Ancora si potrebbe pensare ad un professore, che dovendo assegnare un progetto ai propri studenti, il quale prevede l’utilizzo di particolari strumenti o di particolari sistemi operativi installati, crei una virtual machine adatta alle sue esigenze e la distribuisca agli stessi, in modo tale che quest’ultimi possano concentrarsi sulla realizzazione pura del progetto, trascurando le necessarie configurazioni delle loro macchine.

Questo nuovo modo di utilizzare il laboratorio, sfrutterebbe ancora di più il lavoro di questa tesi, in quanto non avendo un limite minimo di macchine da rendere raggiungibili da remoto, si potrebbe raggiungere l'obiettivo di spegnere tutte le macchine fisiche ad eccezione del nodo manager, negli orari di chiusura dei laboratori, lasciando comunque la possibilità ad un utente remoto di avviare la sua VM e solo quando le risorse del cluster vanno in overload, accendere un ulteriore macchina e cosi via.

Chiaramente si potrebbe ottimizzare maggiormente la situazione cercando di prevedere i maggiori afflussi di utenti remoti, ad esempio in particolari fasce orarie e anticipandone le azioni, avviare un numero maggiore di macchine fisiche, in modo tale da ridurre il tempo di attesa per la riaccensione di una macchina, quando necessario.

Rigraziamenti

Ringrazio i miei genitori che hanno creduto in me ed hanno avuto la pazienza di aspettare tutto questo tempo; Valentina che ha dovuto subire tutti i miei sbalzi d'umore ed ascoltarmi dire cose per lei incomprensibili (e forse veramente incomprensibili!!); mia sorella che c'è passata prima di me e mi ha dato ottimi consigli; Francesco che ha fatto in modo che questa tesi avesse un struttura più scorrevole ed infine tutte le persone che nel bene o nel male hanno contribuito al raggiungimento di questo mio obiettivo.

Qualcuno ha detto: *"...gli esami non finiscono mai"*, bene c'è la *specialistica* che mi attende... Scappi chi può!

Bibliografia

[1] GERALD J. POPEK AND ROBERT P. GOLDBERG. *Formal requirements for virtualizable third generation architectures*. Commun. ACM, 17(7):412-421, 1974.

[2] SMITH, J.E. AND RAVI NAIR. *The architecture of virtual machines. IEEE Computer, vol. 38, no. 5, pp. 32-38, May 2005.*

[3] VMWARE. *Understanding full virtualization, paravirtualization, and hardware assist*. 2007.

[4] Libvirt. http://libvirt.org/. 2010

[5] KEITH ADAMS AND OLE AGESEN. *A comparison of software and hardware techniques for x86 virtualization*. SIGOPS Oper. Syst. Rev., 40(5):2-13, 2006.

[6] DAVID CHISNALL. *The definitive guide to the xen hypervisor*. Prentice Hall Press, Upper Saddle River, NJ, USA, 2007.

[7] DARREN ABRAMSON ET AL. INTEL R. *virtualization technology*. Intel Technology Journal, 10, 2006.

[8] AVI KIVITY ET AL. *Kvm: the linux virtual machine monitor. In Proceedings of the Linux Symposium*, pages 225 - 230, 2007.

[9] *AMD-V Nested Paging*, 2008.

[10] *A performance comparison of commercial hypervisors*. Technical report, Xen Source, Inc., 2007.

[11] ANDREA BEGGI. *La virtualizzazione*, http://www.andreabeggi.net/2007/03/19/la-virtualizzazione/, 2007

[12] *Microsoft Virtual PC*, http://www.microsoft.com/windows/products/winfamily/virtualpc/, Microsoft, 2010

[13] Network Attached Storage, http://it.wikipedia.org/wiki/Network Attached Storage, Wikipedia

[14] Xen, http://www.cl.cam.ac.uk/research/srg/netos/xen/, University of Cambridge Computer Laboratory

[15] VirtualBox, http://www.virtualbox.org/

[16] ALFONSO V. ROMERO. *VirtualBox 3.1: Beginner's Guide*. PACKT, April 2010

[17] OpenVZ, http://wiki.openvz.org/Main_Page

[18] Python, http://www.python.it/

[19] R. DAVOLI. *VDE: Virtual Distributed Ethernet. In Proceedings of the First International Conference on Testbeds and Research Infrastructures for the DEvelopment of NeTworks and COMmunities* (TRIDENTCOM'05). http://ieeexplore.ieee.org/iel5/9524/30172/01386197.pdf

[20] WILLIAM VON HAGEN. *Professional Xen Virtualization.* Wrox. 2008

[21] JEANNA N. MATTHEWS, ELI M. DOW, TODD DESHANE, WENJIN HU, JEREMY BONGIO, PATRICK F. WILBUR, BRENDAN JOHNSON. *Running Xen: A Hands-On Guide to the Art of Virtualization.* Prentice Hall. 2008

[22] CHRIS TAKEMURA, LUKE S. CRAWFORD. *The Book of Xen: A Practical Guide for the System Administrator.* 2010

[23] *Virtualizzazione*.http://it.wikipedia.org/wiki/Virtualizzazione. 2010

Elenco delle figure

Indice analitico