Alma Mater Studiorum · Università di Bologna

SCUOLA DI SCIENZE

Corso di Laurea Magistrale in Informatica

Un Laboratorio Virtuale Distribuito per il test di applicazioni

Relatore:
Chiar.mo Prof.
Vittorio Ghini

Presentata da:
Federica Renzi

Sessione II
Anno Accademico 2013-2014

Dedico questa tesi
a coloro che da sempre
hanno creduto in me

Introduzione

In questo elaborato andremo a descrivere, sia a livello strutturale che implementativo, il progetto sperimentale da noi ideato volto alla creazione di un *laboratorio virtuale di testing*. Lo scopo principale del laboratorio è appunto quello di delocalizzare la fase preliminare di testing di un'applicazione; nel nostro caso specifico siamo partiti dallo scenario riguardante la rete ferroviaria ed abbiamo preso in esame il software utilizzato per la gestione del traffico.
Il vincolo principale che grava su questo progetto è che l'applicazione da collaudare non deve minimamente essere modificata, e ciò significa che occorre creare intorno ad essa un ambiente di testing conforme alle caratteristiche in cui il software si aspetta di operare.
Per realizzare tale progetto è stato necessario compiere uno studio approfondito riguardo a due delle tematiche più popolari in questi anni: la virtualizzazione e la delocalizzazione nel cloud. Inoltre, ci siamo persino spinti verso un concetto ancora relativamente giovane e non del tutto conosciuto: la nested virtualization.
Al fine di far comprendere al meglio al lettore tutte le scelte implementative da noi effettuate per la creazione del nostro laboratorio virtuale di testing, abbiamo strutturato la tesi in modo da presentare prima ogni macroargomento singolarmente, per poi concludere con una descrizione accurata dei vari passaggi necessari alla resa in opera del sistema.

Indice

Elenco delle figure

Elenco delle tabelle

Capitolo 1

Struttura della tesi

Data la vastità degli argomenti toccati ed inoltre la complessità di certe nozioni presentate, abbiamo deciso di strutturare la tesi in modo tale da dividere accuratamente ogni tema, per poterlo affrontare singolarmente.
Inizieremo dedicando diversi capitoli alla spiegazione dettagliata dei principali concetti su cui verte il documento di tesi, al fine di fornire una conoscenza basilare sufficiente per comprendere i vari tasselli di cui si compone il nostro laboratorio virtuale di testing; poi ci addentreremo nella descrizione dello scenario ed infine illustreremo la soluzione implementativa che abbiamo scelto per realizzare la nostra idea iniziale.

1.1 Contenuto dell'elaborato

La Virtualizzazione

La Virtualizzazione è un tema molto vasto e le sue implicazioni pratiche sono numerose. In questo capitolo verrà sviscerato il significato del termine e verranno illustrate le principali tecniche esistenti per la sua realizzazione.

Virtualizzazione nel Cloud

La parola "cloud", sebbene molto diffusa, assume svariati significati in base al contesto di utilizzo. In questo capitolo parleremo di al-

cuni dei più noti servizi Cloud, come l'IaaS, il PaaS e il SaaS; inoltre presenteremo anche un concetto ancora relativamente nuovo, che verrà da noi sfruttato per la realizzazione del laboratorio virtuale: la virtualizzazione annidata.

Il tempo nei Calcolatori

Esistono molti modi per misurare il trascorrere del tempo in un calcolatore, ed inoltre esistono diversi timer device atti a questo scopo. In questo capitolo osserveremo nel dettaglio il concetto di tempo dal punto di vista dei principali sistemi operativi esistenti, sia illustrando la loro scelta nell'utilizzo dei timer device esistenti, che descrivendo la prospettiva che assumono nei confronti del tempo.

Virtualbox

Virtualbox è uno dei più conosciuti software di virtualizzazione ed è sicuramente lo strumento principale da noi utilizzato nella realizzazione del progetto. In questo capitolo descriveremo le principali caratteristiche di questo software, soffermandoci in particolar modo su quelle utili ai fini del nostro contesto di applicazione.

Il laboratorio virtuale distribuito

È in questo capitolo che presenteremo lo scenario e che illustreremo l'idea da cui è partito il progetto di questa tesi. Cercheremo di descrivere a grandi linee cosa vorremmo ottenere e quali sono i vincoli che lo scenario ci impone. Inoltre anticiperemo quali sono i concetti chiave su cui si basa la progettazione del sistema e che poi verranno presentati più nel dettaglio nei capitoli successivi.

Il laboratorio nel Cloud

Il primo passo per realizzare il laboratorio virtuale di testing è quello di instanziare delle macchine in un claud che offra il servizio di IaaS. In questo capitolo presenteremo il provider da noi scelto, cioè ∼Okeanos, e poi spiegheremo le scelte da noi effettuate per predisporre le macchine del cloud ad ospitare il nostro progetto di tesi.

Modifica del clock

La modifica del clock è l'operazione cardine attorno a cui gira tutto il nostro laboratorio virtuale di testing. In questo capitolo spiegheremo innanzitutto il motivo per cui l'abbiamo ritenuta necessaria, e successivamente illustreremo come Virtualbox si comporta rispetto a questo particolare aspetto della virtualizzazione. Infine esporremo i due possibili approcci implementativi per compiere tale operazione.

Le configurazioni di Virtualbox

In questo capitolo vedremo, passo dopo passo, quali sono i comandi che abbiamo utilizzato per configurare Virtualbox e rendere il nostro laboratorio operativo, spiegando anche le ragioni delle nostre scelte implementative. Concluderemo esponendo e commentando i risultati dei nostri test di verifica sul funzionamento del sistema.

Capitolo 2

La Virtualizzazione

Il concetto di virtualizzazione di un sistema di elaborazione non è recente: uno dei primi esempi, introdotto dalla IBM, risale agli anni '60. Tuttavia, nell'ultimo ventennio, si è assistito ad un forte incremento dell'uso di questa tecnica in diversi ambiti della programmazione, col principale scopo di ottimizzare le risorse contenendo però i costi di gestione.
Essendo una nozione dal vasto contenuto e dalle innumerevoli implicazioni, il suo significato genera spesso confusione e imprecisioni. Perciò in questo capitolo cercheremo, dopo aver fatto un breve excursus storico, di effettuare una classificazione delle varie tecniche esistenti, cercando di specificare pregi e difetti di ognuna.

2.1 Cenni Storici

Come anticipato nell'introduzione, il primo caso di virtualizzazione è stato introdotto da IBM intorno agli anni '60 col sistema di elaborazione denominato inizialmente CP/CMS ed in seguito diventato VM/370 [2].
La componente CP (Control Program) consisteva in un primordiale livello di astrazione, il cui compito era quello di fare vedere ai livelli superiori (il sistema operativo) diverse macchine logiche; la virtualizzazione offerta da questa interfaccia consisteva semplicemente in una replica dell'hardware fisi-

co, senza l'apporto di alcun ulteriore servizio agli utenti. La componente CMS (Conversational Monitor System) rappresentava invece il sistema operativo, interattivo e monoutente, che girava su ogni macchina virtuale.
Il CP/CMS è stato ideato con l'obiettivo di creare un sistema time-sharing. La sua adozione, nella versione VM/370, da parte di IBM fu una diretta conseguenza del sostanziale fallimento del sistema time-sharing TSS/360 che si rivelò non adeguato alle aspettative perché troppo complesso e poco efficiente [1]. Viceversa, l'idea architetturale concepita con CP/CMS consentiva ad ogni utente di avere una propria partizione sul disco e di supportare lo sviluppo dei suoi programmi sfruttando completamente l'hardware a disposizione sulla macchina fisica. Inoltre l'implementazione di tale idea risultava più semplice da gestire rispetto al tradizionale sistema time-sharing in quanto risultavano separate, e quindi sviluppabili indipendentemente, le due componenti essenziali al concetto di virtualizzazione: la suddivisione dell'uso delle risorse fisiche tra gli utenti e il mascheramento per l'utente delle peculiarità dell'hardware.
Uno dei principali vantaggi era che, poiché la macchina virtuale a disposizione dell'utente rispecchiava perfettamente le caratteristiche dell'hardware fisico, di fatto era possibile utilizzare sopra di essa qualsiasi sistema operativo compatibile alle specifiche fisiche; rendendo perciò potenzialmente possibile avere un unico calcolatore con installati differenti sistemi operativi. Inoltre, un altro punto di interesse era la solidità; infatti, visto che la struttura garantiva alle macchine virtuali una esecuzione separata, vi era la garanzia che un errore in uno dei sistemi operativi non avrebbe influenzato l'esecuzione degli altri, rendendo il sistema estremamente affidabile.
È chiaro che questi sono dei vantaggi enormi, soprattutto se si considera che in quel periodo storico non esistevano ancora sistemi operativi multitasking interattivi.
A partire dagli anni '70 però, sono arrivati sul mercato i primi sistemi operativi multitasking ed, inoltre, nei successivi anni si è assistito ad un colossale crollo del costo dell'hardware, accompagnato da un notevole incremento delle

sue potenzialità. I mainframe sono gradualmente scomparsi, lasciando il posto a dei calcolatori di dimensione molto inferiore. Ciò condizionò i progettisti informatici, facendoli progredire verso un paradigma del tipo "un server per ogni applicazione".
Ovviamente, in un contesto come quello descritto, lo sviluppo delle tecniche di virtualizzazione non poteva che subire una brusca interruzione, tanto che le architetture non hanno più fornito l'hardware necessario per implementarle in modo efficiente.
Si è così giunti, intorno ai primi anni 2000, ad avere una proliferazione di hardware nelle sale server, le quali risultavano essere colme di calcolatori decisamente sottoutilizzati rispetto alle potenzialità di cui disponevano. Ciò, oltre ad essere uno spreco di risorse informatiche, si rivelava essere anche uno dispendio di spazio e di denaro per il mantenimento. Per questi motivi, già dalla metà degli anni '90, è stato riportato alla luce il concetto di virtualizzazione, con l'obiettivo di ottimizzare quelle risorse disponibili ma non pienamente sfruttate.

2.2 Definizione e Classificazione

Partendo da una spiegazione il più generica possibile sul significato di virtualizzazione, possiamo dire quanto segue [3] :

> *"La virtualizzazione è la separazione di una risorsa o di una richiesta di servizio, dalla metodologia con cui questa viene effettivamente servita a livello fisico"*

Dalla definizione sopracitata risulta chiaro che l'enorme varietà di interpretazioni che derivano da questo concetto non sono altro che la conseguenza diretta dell'esistenza di varie modalità di implementazione della virtualizzazione, le quali dipendono soprattutto dalla scelta del livello in cui porre l'astrazione delle risorse.
La Fig. 2.1, redatta da Phelp J.R. e Dawson P. [5] e successivamente riadat-

tata [6] , mostra una classificazione delle diverse tecniche di virtualizzazione esistenti.

Si può subito identificare una divisione in tre macro categorie:

Partitioning Virtualization la virtualizzazione mediante partizionamento permette di configurare l'hardware fisico al fine di predisporlo per ospitare più macchine logiche, ognuna con il proprio sistema operativo. Questa categoria comprende i concetti come:

Hardware Partitioning: che consiste in una suddivisione fisica dell'hardware della macchina, permettendo così di definire le risorse da attribuire ad ogni sistema di elaborazione. Non essendoci una effettiva "condivisione" delle risorse, ma piuttosto una allocazione, si ha il vantaggio di non avere *overhead* per la gestione della virtualizzazione in quanto i confini di ogni macchina sono definiti a livello hardware.

Hypervisor: cioè quello strato di software che permette solamente l'astrazione dell'hardware fisico, non integrando però alcuna funzionalità di gestione per le macchine virtuali. Grazie ad esso è possibile far percepire alle macchine virtuali installate sul sistema, un hardware persino differente, sia dal punto di vista qualitativo che quantitativo, da quello che è realmente presente nella macchina.

In base a dove viene collocato avremo una bare-metal virtualization, se si trovi a diretto contatto con l'hardware, o una OS-hosted virtualization, se invece è ospitato da un altro sistema operativo (vedi Fig. 2.2 e Fig. 2.3).

Virtual Machine Monitor (VMM): è il software che si occupa di gestire sia la condivisione delle risorse del sistema che il ciclo di vita di ogni VM presente sulla macchina fisica.

OS Virtualization: questa tecnica, a differenza delle altre, inserisce lo strato di virtualizzazione fra sistema operativo e applicazioni.

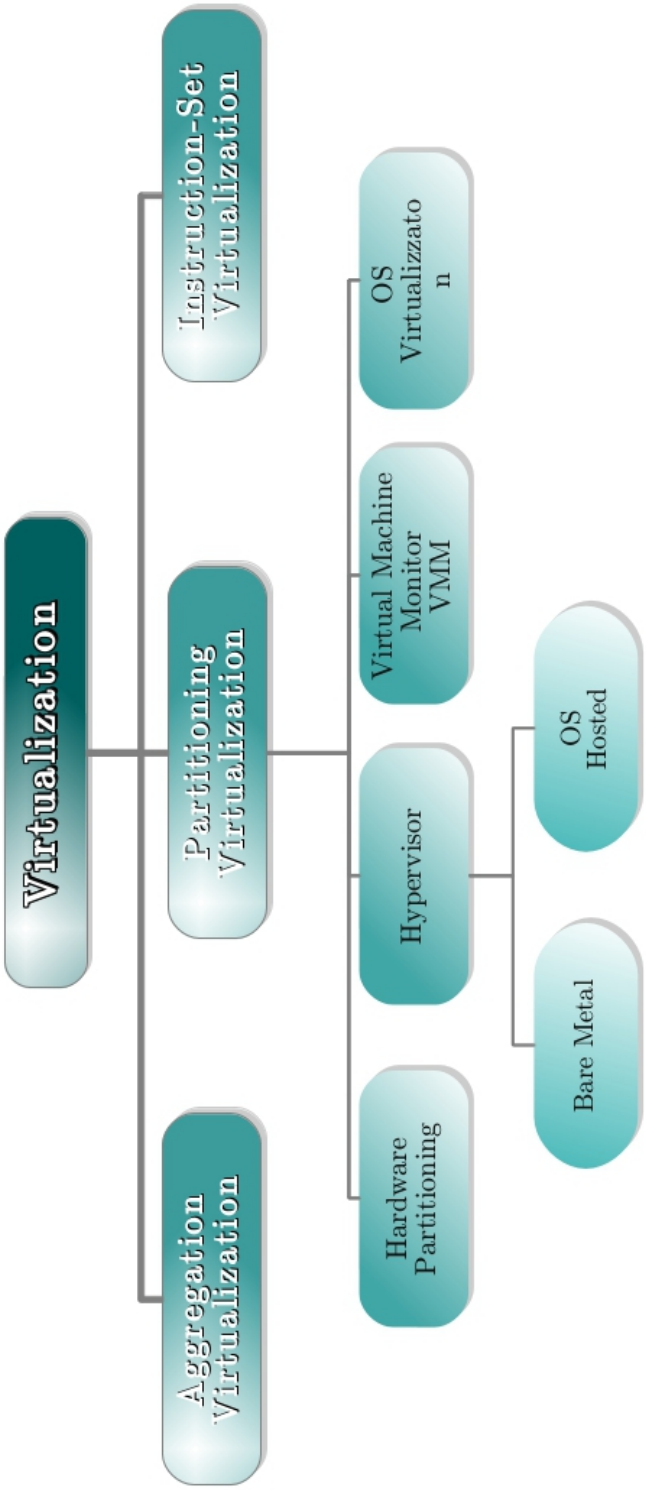

Fig. 2.1: Classificazione delle tecniche di virtualizzazione

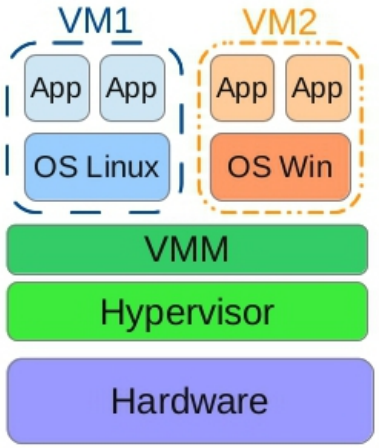

Fig. 2.2: Bare-Metal Virtualization

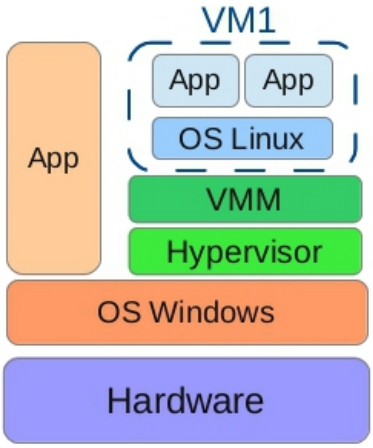

Fig. 2.3: OS Hosted Virtualization

Instruction-Set Virtualization questa tecnica permette di emulare un diverso sistema hardware/software traducendo le istruzioni originali in altre, comprensibili dal sistema ospitante. Alcuni esempi noti sono:

- gli ambienti di sviluppo per dispositivi mobili (Android, Symbian, ecc...), che permettono di eseguire su un normale PC le applicazioni dirette a tali device, agevolando enormemente il lavoro di debugging del codice;
- MAME (Multiple Arcade Machine Emulator) il quale rende possibile giocare ai videogiochi classici degli anni '80 eseguendo il codice originale direttamente sul PC
- Java Virtual Machine (JVM) che effettua una traduzione "on the fly" del byte code java in istruzioni eseguibili in qualsiasi sistema, rendendo il linguaggio di programmazione Java incredibilmente portabile.

Aggregation Virtualization in questo caso, a differenza degli altri, la virtualizzazione viene sfruttata per astrarre la complessità del servizio fornito, il quale viene realizzato suddividendo il "compito" principale in

task più semplici ed indipendenti tra loro, ognuno dei quali viene assegnato ad un diverso nodo dell'infrastruttura di calcolo. Alcuni esempi tipici sono il *GRID Computing* [1], o più in generale tutte le tecniche di calcolo distribuito, e il *Virtual Infrastructure* ideato da VMware riportato nella Fig. 2.4 (riprenderemo questo argomento nel capitolo successivo).

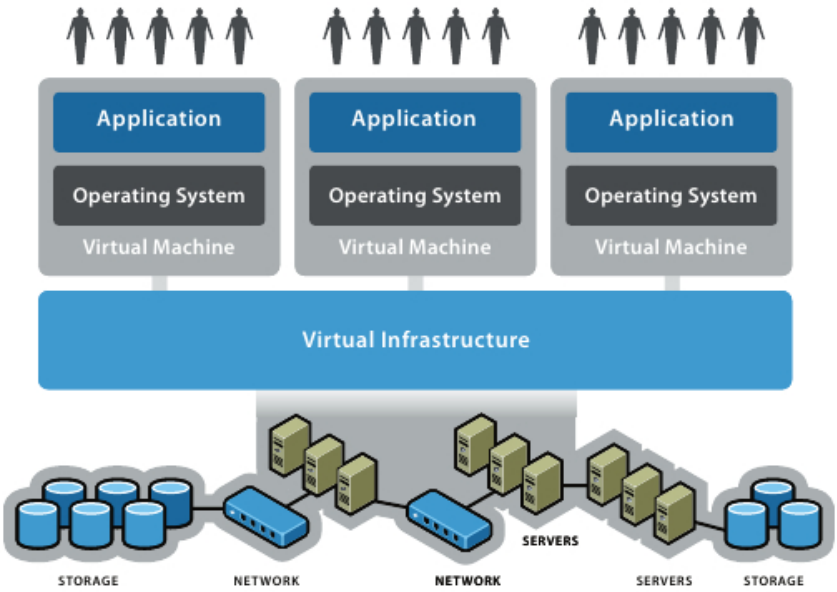

Fig. 2.4: Virtual Infrastructure[2]

Sebbene questa possa sembrare una classificazione netta e le diverse tecniche sembrino totalmente differenti tra di loro, in realtà ci si trova spesso ad usare una versione ibrida delle varie implementazioni, al fine di soddisfare completamente tutti i vincoli di sistema che si vengono a creare durante la progettazione del software di virtualizzazione.
Al di là dei diversi modi con cui si può progettare un VMM, esso deve comunque soddisfare alcune condizioni essenziali di servizio:

[1] http://www.gridcomputing.com/

- ambire ad ottenere la migliore efficienza possibile di elaborazione, il più possibile conforme alle potenzialità della macchina fisica
- garantire un certo isolamento del software che gira al suo interno, in modo tale da non compromettere in alcun modo il sistema ospitante
- fornire al sistema ospitato tutte le funzionalità che avrebbe se potesse girare direttamente sull'hardware fisico

2.3 Implementazione della Virtualizzazione

Un altro tipo di classificazione della virtualizzazione si fonda invece sull'approccio implementativo del VMM e sui principi che governano il dialogo tra esso la macchina ospitata. In base a tale classificazione, esistono due tipi di implementazione possibile:

1. *Full Virtualization* (Virtualizzazione Completa)
2. *Paravirtualization* (Paravirtualizzazione)

Prima di proseguire occorre speficiare alcune nozioni importanti: normalmente viene indicato con il termine *host* la piattaforma di base sulla quale girano le macchine virtuali, che comprende la macchina fisica, l'eventuale sistema operativo e il VMM; si indica invece con il termine *guest* tutto ciò (applicazioni e sistema operativo) che si trova a girare nella macchina virtuale.
Il concetto chiave per distinguere queste due differenti implementazioni sta nell'identificare se il sistema operativo guest si renda o meno conto di essere su una macchina virtuale. Andiamo ora ad analizzare i diversi approcci più nel dettaglio.

2.3.1 Full Virtualization

In questo caso, il sistema operativo guest viene indotto a credere di essere l'unico presente sulla macchina fisica e di aver pieno possesso delle risorse

hardware. Perciò tale tecnica di virtualizzazione prevede, per essere realizzata, che il VMM sia in grado di mostrare all'OS guest, tutto l'hardware di cui necessita: il BIOS, la/le CPU, la memoria, i dispositivi di I/O, ecc... devono essere tutti presenti sotto forma virtuale.
Il principale vantaggio di questo paradigma riguarda la possibilità di eseguire su tale sistema di virtualizzazione tutti i sistemi operativi tradizionalmente conosciuti, senza che questi debbano minimamente essere modificati. Ciò si traduce in una elevatissimo grado di portabilità del sistema guest il quale, per essere eseguito, non richiede la presenza di un hardware particolare alla macchina host, ma ha come unico vincolo quello di dover essere virtualizzato dallo stesso VMM da cui è stato creato.
È facile intuire, d'altro canto, che la tecnica della virtualizzazione completa porta ad una maggiore complessità nell'implementazione del VMM, ed inoltre ad una maggiore dipendenza dal tipo di CPU. Infatti, come vedremo successivamente, essa condiziona in modo decisivo la stabilità del sistema ospitato e da essa dipende l'ammontare dell'overhead necessario al VMM per virtualizzare tutte le operazioni eseguite dal guest.
Quando un sistema operativo "crede" di avere pieno possesso delle risorse hardware, esso le gestisce in maniera diretta e privilegiata. Per capire meglio questo concetto occorre fare una piccola premessa per ricordare il funzionamento di un'architettura CPU: essa opera secondo diversi livelli di protezione detti *ring*, i quali decretano una gerarchia tra gli utilizzatori della macchina fisica (vedi Fig. 2.5). Per semplicità possiamo considerare una suddivisione su due livelli: uno superiore (detto anche *ring 0*) su cui risiede il sistema operativo e uno denominato utente. All'interno del ring 0 viene ammessa l'esecuzione di tutte le istruzioni della CPU, privilegiate e non, permettendo così una totale gestione dell'hardware fisico; viceversa, nel livello utente vengono eseguiti gli applicativi i quali, non potendo eseguire alcuna operazione privilegiata, devono interfacciarsi con il sistema operativo host per poter operare.

Nel caso in cui un'applicazione in user space tenti di eseguire un'istruzione

Fig. 2.5: Struttura architetturale dei livelli di privilegio di una CPU

privilegiata, la CPU può comportarsi in due modi:

- ignorare la richiesta di esecuzione
- avvisare il sistema operativo tramite un *trap*[3], il quale interpreterà la richiesta intraprendendo gli opportuni provvedimenti.

Quando una CPU è in grado di intercettare tutte istruzioni privilegiate e di notificarle con un trap, viene definita *naturalmente virtualizzabile*.
Si può facilmente intuire come una CPU di questo tipo abbia un duplice vantaggio sui VMM in termini di:

Efficienza Il VMM potrà permettere alle macchine virtuali di eseguire direttamente il codice sulla CPU fisica senza alcun controllo e, perciò, senza aggiungere alcun overhead;

Stabilità Nel caso in cui il sistema guest tenti di eseguire un'operazione potenzialmente dannosa, la CPU potrà bloccarla e notificarla al VMM il quale agirà nel modo più adatto alla situazione[4].

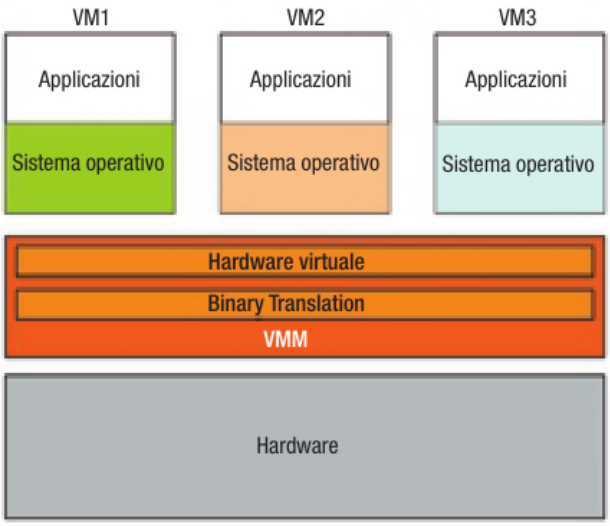

Fig. 2.6: Full Virtualization [1]

Purtroppo però non tutte le CPU sono naturalmente virtualizzabili, ed in questi casi le VMM dovranno prevedere una fase di *parsing* delle istruzioni che il guest intende eseguire, al fine di poter intercettare preventivamente quelle potenzialmente nocive (vedi Fig. 2.6). Tale operazione viene chiamata *binary translation* e si occupa di tradurre le istruzioni da un instruction set source ad un instruction set target, a scapito ovviamente di una perdita di efficienza nella virtualizzazione.
Un esempio di architettura che necessita di questo tipo di operazione è il caso dell'architettura x86 che descriveremo in seguito.

2.3.2 Paravirtualization

La parola *"Paravirtualizzazione"* sfrutta il prefisso "para", una preposizione derivante dal greco che assume il significato di "al fianco", "al di

[3]Consiste in un'eccezione causata dal software che si traduce in un'interruzione sincrona che assegna il controllo al sistema operativo.

[4]Per semplificare la situazione abbiamo ipotizzato che il VMM si trovi nel ring 0, assieme al sistema operativo host. In realtà però le CPU hanno più di due livelli di protezione.

fuori di". Perciò il significato effettivo della parola potrebbe essere "virtualizzazione affiancata", riferendosi al tipo di comunicazione che si instaura tra il sistema operativo guest ed il VMM [4].
A differenza di quelli descritti in precedenza, i VMM paravivirtualizzati non emulano interamente l'hardware della macchina fisica, ma definiscono e implementano un'interfaccia applicativa tra il VMM e il guest, nota anche come *Virtual Hardware API*. Da ciò possiamo facilmente dedurre che, per poter utilizzare questo tipo di approccio, il sistema operativo presente sulla macchina virtuale deve essere consapevole di risiedere in un ambiente virtualizzato e di non essere l'unico sistema operativo presente sulla macchina fisica; ne consegue che il sitema operativo guest non è installato nella sua versione originale ma deve essere opportunamente scelta una sua versione modificata che sia in grado di interfacciarsi con il VMM.
Per applicare tale paradigma perciò, è necessario effettuare il *porting* del

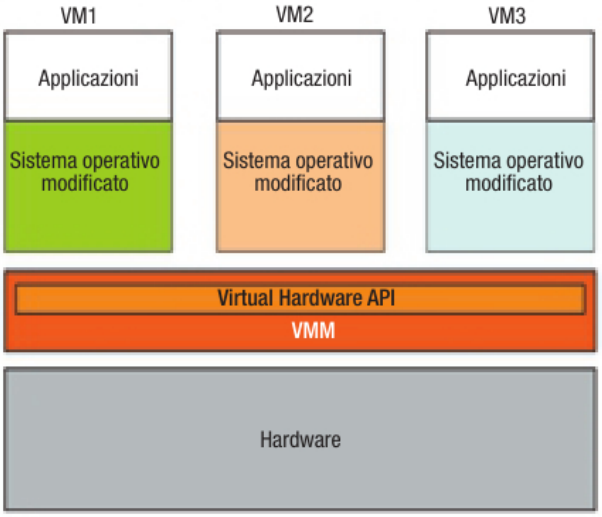

Fig. 2.7: Paravirtualization [1]

sistema operativo che si intede usare e ciò costituisce un enorme svantaggio alla portabilità della VMM perché non tutti gli utenti possono avere accesso a tali risorse. Viceversa però, questo tipo di virtualizzazione include anche

una serie di vantaggi:

- l'implementazione dei VMM è più snella e semplice da realizzare;
- non è più necessario prevedere e gestire le CPU non naturalmente virtualizzabili;
- maggiore interattività tra VMM e sistema operativo guest, consentendo una maggiore efficienza nella gestione delle risorse condivise.

Uno dei progetti più noti orientati alla paravirtualizzazione è XEN[5], sviluppato presso il Computer Laboratory dell'Università di Cambridge e rilasciato con licenza opensource, sul quale sono stati effettuati diversi porting di sistemi operativi come Linux, BSD, WindowsXP, ...

2.4 Il caso del processore x86

Come abbiamo precedentemente accennato, non tutte le architetture possono essere definite "naturalmente virtualizzabili" e, tra queste, vi è anche l'architettura x86 la quale, realizzata nell'epoca del boom del personal computer, non è stata minimamente concepita tenendo presente le condizioni per una semplice virtualizzazione.
Questa assenza di una collaborazione hardware da parte della macchina fisica, comporta necessariamente che tutto il processo di virtualizzazione venga gestito in modo software, rendendo il VMM più pesante ed il sistema guest meno performante.
Affrontiamo il problema in base ai tre punti critici precedentemente citati:

- CPU
- Memoria
- I/O

[5]http://www.xenproject.org/

2.4.1 Virtualizzazione della CPU

Ciò che complica particolarmente il lavoro del VMM è che, in questo particolare sistema, le istruzioni privilegiate eseguite nello spazio utente non provocano un'interruzione tramite trap (come spiegato nella sezione 2.3.1), ma vengono semplicemente ignorate e questo non consente in alcun modo al VMM di intervenire in modo trasparente (vedi Fig. 2.8).

Inoltre, un'altra problematica di questo tipo di architettura è il cosiddetto

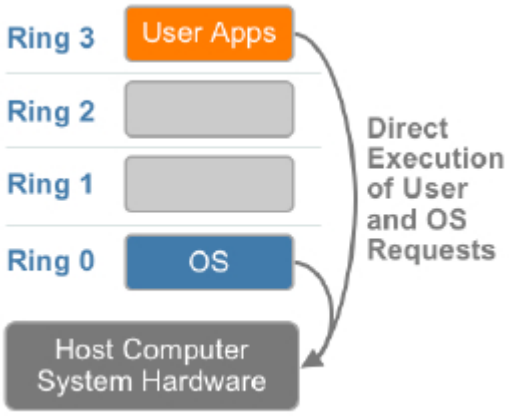

Fig. 2.8: Architettura x86[4]

ring aliasing: esistono delle istruzioni che, pur non essendo delle istruzioni privilegiate (perciò richieste dal ring 3 in *user mode*), consentono di accedere in lettura a registri di sistema le cui informazioni andrebbero invece mascherate ad una macchina virtuale. Ad esempio possiamo citare il *code segment register*, un registro che punta ad una tabella del sistema operativo il cui contenuto segnala il livello di privilegio corrente; risulta evidente che la lettura di questo registro da parte del sistema operativo guest (eseguito ovviamente nello spazio utente invece che nel ring 0) segnalerebbe al medesimo un'anomalia di collocazione e porterebbe a dei malfunzionamenti [1].

Le difficoltà appena citate portavano persino a credere che la virtualizzazione dell'architettura x86 fosse quasi impossibile da implementare. Chiaramente,

la mancanza di un sostegno da parte dell'hardware obbliga il VMM a cercare degli stratagemmi per aggirare il problema e garantire comunque il funzionamento della virtualizzazione completa. Gli inconvenienti possono essere risolti, ma il prezzo da pagare è un aumento della complessità di implementazione per il VMM ed una riduzione delle prestazioni nella virtualizzazione. Fu VMware a riuscire a vincere per prima la sfida di virtualizzare l'architettura x86, e lo fece nel 1998 [4]. E lo fece sfruttando una tecnica detta *dynamic binary translation* per mezzo della quale riusciva a sostituire i blocchi di codice conteneti le istruzioni che potevano causare le problematiche precedentemente citate, con blocchi equivalenti dal punto di vista funzionale ma contenenti istruzioni per la notifica di eccezioni, favorendo così l'entrata in gioco del VMM (vedi Fig. 2.9). Tali blocchi venivano passati in esecuzione direttamente ed inoltre venivano conservati in una particolare cache al fine di permetterne un eventuale riuso futuro, risparmiando così il costo di ulteriori translazioni.

Affinché il sistema possa funzionare completamente e senza errori, questo

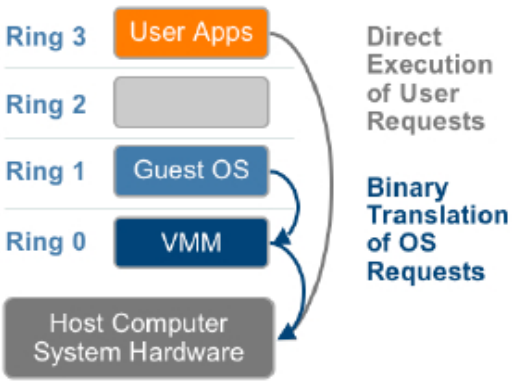

Fig. 2.9: Virtualizzazione di Architetture x86 ideata da VMware[4]

processo di traslazione va applicato all'intero kernel del sistema operativo guest, per essere certi che tutte le istruzioni che potrebbero causare dei problemi vengano opportunamente intercettate e gestite.

2.4.2 Virtualizzazione della memoria

L'unità di gestione della memoria, definita in inglese con l'acronimo di MMU (Memory Management Unit), è una classe di componenti hardware che gestisce le richieste di accesso alla memoria generate dalla CPU su sollecitazione dei processi.
Già di per sè la memoria ha una sua forma di virtualizzazione, infatti i processi non si curano dello spazio di indirizzamento fisico che utilizzano, ma si appoggiano completamente alla MMU per la gestione di queste problematiche. Possiamo perciò dire che la "memoria virtuale" governata dalla MMU consiste nello spazio di memoria lineare visto dall'ottica di un processo (vedi Fig. 2.10). Per svolgere questa gestione a favore dei processi la MMU si av-

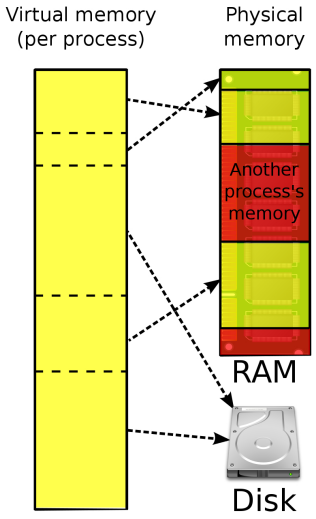

Fig. 2.10: Memoria virtuale percepita dai processi [7]

vale di due strutture dati: il Translation Lookaside Buffer (TLB) e la Paging Table (PT). Prima di tutto lo spazio di memoria virtuale viene catalogato in pagine di dimensioni prefissate (Paging Table) le quali vengono poi utilizzate per mappare gli *indirizzi virtuali* in indirizzi fisici. Il TLB invece consiste in un buffer[6] della Page Table, cioè tiene traccia solamente un sottoinsieme del

[6]In realtà, in alcune implementazioni, esso è rappresentato da una cache della CPU

suo contenuto e viene utilizzato per velocizzare ulteriormente la traduzione degli indirizzi virtuali in indirizzi fisici.
Quando entra in gioco la virtualizzazione di un sistema operativo la questione si complica. Infatti il sistema operativo guest suppone di possedere una propria MMU, ma la memoria fisica utilizzata da quest'ultimo è diversa dalla memoria fisica della macchina host. Perciò diviene necessario un secondo livello di traslazione ad opera del VMM, chiamato *Shadow Page Table* (SPT).
L'SPT è una tecnica software atta a mantenere una versione "fantasma" della Paging Table, specifica per il sistema guest. L'idea è quella di mantenere due PT, entrambe operative durante una transazione: la PT del guest (gPT) e la PT fantasma (sPT). Quando il guest è attivo, l'hypervisor forza il processore ad utilizzare la sPT, invisibile al sistema operativo guest, per realizzare un ulteriore traduzione degli indirizzi (vedi Fig. 2.11).
Il VMM fa in modo di mantenere sincronzzate le shadow pages con le guest

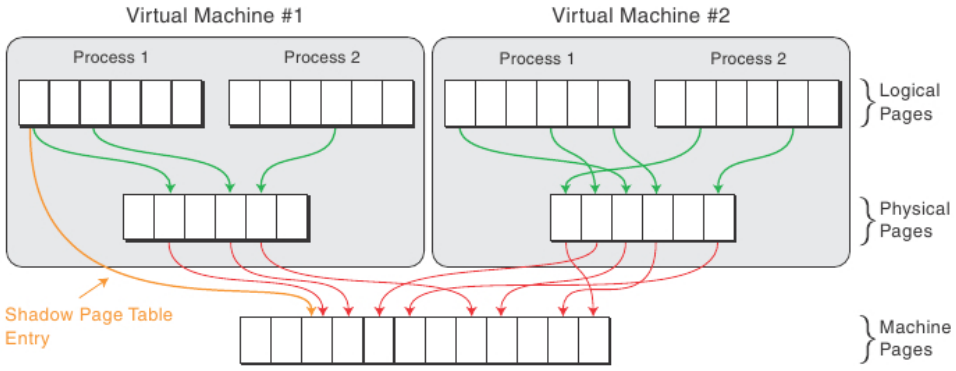

Fig. 2.11: Shadow Paging Table [9]

pages e questa operazione introduce overhead nel momento in cui il guest effettua un update della sua MMU.

2.4.3 Virtualizzazione delle periferiche

Le scelte effettuate riguardo alla virtualizzazione delle periferiche hanno come scopo principare quello di garantire la più ampia portabilità al sistema virtualizzato. Perciò, piuttosto che rendere il VMM e il guest dipendendi dai driver specifici per le periferiche della macchina fisica, ciascuna macchina virtuale viene fornita di un insieme di periferche il più possibile standard (tra cui DVD, tastiera, scheda video, USB, scheda di rete, scheda audio, ...), tutte emulate dal VMM.
È compito del VMM gestire l'interazione tra le periferiche reali e quelle emulate e lo fa seguendo la seguente procedura [12]:

1. la periferica fisica comunica con il VMM e scrive su di un suo buffer i dati da trasferire; per fare ciò utilizza il DMA (Direct Memory Access) grazie al quale è possibile accedere direttamente alla memoria interna per scambiare i dati generando un singolo interrupt per blocco trasferito

2. quando la copia dei dati raggiunge il termine, l'interrupt generato viene catturato dal VMM

3. il VMM esamina i pacchetti, individua la macchina virtuale di destinazione e copia i pacchetti nel suo specifico buffer virtuale

4. la periferica emulata lancia un interrupt avvisando il relativo sistema operativo guest che dei dati sono arrivati.

2.4.4 In breve

Concludendo il discorso dell'architettura x86 possiamo dire che risulta evidente al lettore che la completa assenza di supporto dal punto di vista hardware non solo non facilita i compiti del VMM ma comporta un considerevole overhead per ogni operazione compiuta dal sistema guest.
Fortunatamente, grazie all'aumento di popolarità subito dalla virtualizzazione,

soprattutto nel corso dell'ultimo decennio, anche i produttori hardware si sono attivati al fine di sviluppare delle tecniche fisiche che permettano una più agevole collaborazione tra il sistema host e il VMM.

2.5 Hardware Assisted Virtualization

L'Hardware Assisted Virtualization è la prova tangibile di quanto sia divenuta importante, specialmente nell'ultimo decennio, la virtualizzazione e di quanto l'informatica sia sempre più alla ricerca di modi diversi per sfruttarne al massimo i punti di forza.
I più noti produttori di CPU come Intel e AMD si sono giustamente resi conto del potenziale insito in questo trend ed hanno deciso di trovare degli accorgimenti hardware che permettessero di agevolare il lavoro del VMM, rendendo così tutto il sistema maggiormente solido e performante. Gli accorgimenti consistono, come suggerisce il nome, in alcuni supporti hardware, come l'aggiunta di un nuovo gruppo di instruction set o di nuove strutture logiche, che assistono il VMM nell'opera di virtualizzazione.
Fondamentalmente, tali modifiche sono atte ad avvantaggiare la continua supervizione che il VMM deve svolgere nei confronti del guest, al fine di permettergli più facilmente di catturare e gestire personalmente tutte quelle operazioni che coinvolgono in qualsiasi modo l'architettura del sistema host. Dal punto di vista del guest però, tutto questo non ha alcun impatto diretto, e ciò significa che il sistema operativo sulla VM non si accorge minimamente di essere virtualizzato (Full Virtualization).
I punti su cui Intel e AMD si sono concentrati, riguardano le tre aree di maggior criticità nel lavoro di virtualizzazione: la gestione della memoria del guest, l'isolamento di alcune istuzioni della CPU e il controllo dell'I/O. Sebbene l'implementazione delle tecniche hardware differisca tra i due produttori, le linee guida su cui essi si basano coincidono. In seguito andremo a descriverne i punti chiave, prendendo in esame principalmente le scelte fatte da Intel.

Il processore

La tecnologia di virtualizzazione messa a disposizione dalla Intel è disponibile in due versioni: VT-x per il processore x86 a 32 bit e VT-i per Itanium (ad esempio IA-64) a 64 bit; AMD invece ha sviluppato la tecnologia conosciuta come AMD-V. Per non fare confusione, in questa sezione parleremo solo del VT-x ma, come già accennato in precedenza, queste tecnologie sono pressoché identiche.
Il supporto alla virtualizzazione fornito dal processore consiste in un set di operazioni macchina chiamate VMX, le quali si dividono in due categorie:

- **VMX root operation** che identifica una modalità in cui un software può agire con maggiori privilegi;
- **VMX non-root operation** che identifica i classici livelli di protezione, dal ring 0 al ring 3.

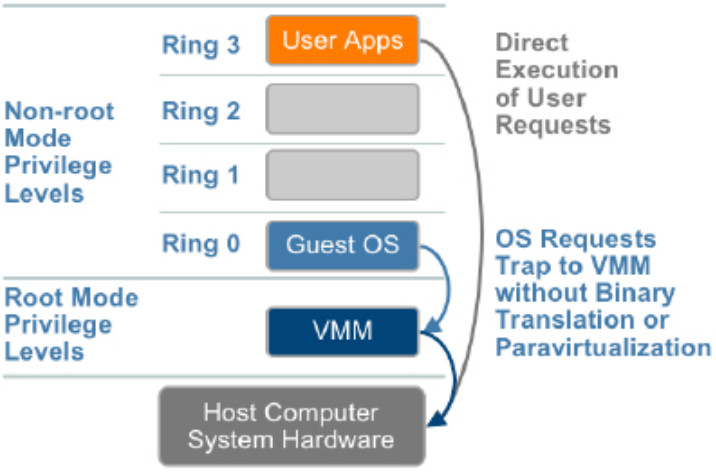

Fig. 2.12: I livelli di privilegi del processore

In breve possiamo dire che la modalità root è riservata al VMM, mentre tutto il software del guest, compreso il sistema operativo, gira in modalità

non-root (vedi Fig. 2.12 [7]).
Il passaggio da una modalità all'altra viene chiamato *VMX transition*; quando si accede al livello non privilegiato (cioè quando viene attivato il guest), si ha un *VMX entries*, mentre se si esce da questa modalità, attivando i privilegi di root, si parla di *VMX exits*.
Il comportamento del processore in VMX root operation è sostanzialmente uguale a quello di una normale CPU, con due differenze importanti: l'aggiunta di un nuovo set di istruzioni (*VMX instructions*) e l'esistenza di alcuni vincoli che impongono una riduzione dei valori caricabili in alcuni registri di controllo.
Viceversa, il comportamento del processore in VMX non-root operation viene opportunamente modificato per facilitare la virtualizzazione; le operazioni che si possono compiere in questa modalità, infatti, sono appositamente limitate al fine di evitare che il guest effettui delle richieste illecite alla macchina fisica. Sono proprio tali limitazioni a permettere al VMM di assumere il controllo della situazione: ogniqualvolta si verifica una violazione da parte del guest, cioè quando questo tenta di eseguire un'istruzione che non gli è permessa, si solleva un interrupt che causa l'operazione di VM exit, con conseguente attivazione della modalità privilegiata per il VMM.
Grazie a questo accorgimento il VMM può provvedere a gestire personalmente l'istruzione lanciata dal guest ed inoltre, trovandosi di fatto a girare nel ring 0 (sebbene con permessi di non-root), il sistema operativo ospitato non si accorge minimamente di trovarsi su una macchina virtuale.
È facile intuire come tutto ciò simpifichi notevolmente il lavoro del VMM.
La figura 2.13 illustra il ciclo di vita di un VMM e dei guest da esso controllati:

1. Il software entra in modalità privilegiata utilizzando l'istruzione VMX-ON, la prima delle VMX operation;

[7] L'immagine è tratta da [4] e rappresenta l'Hardware Assisted Virtualization in generale, senza entrare nello specifico con Intel o AMD

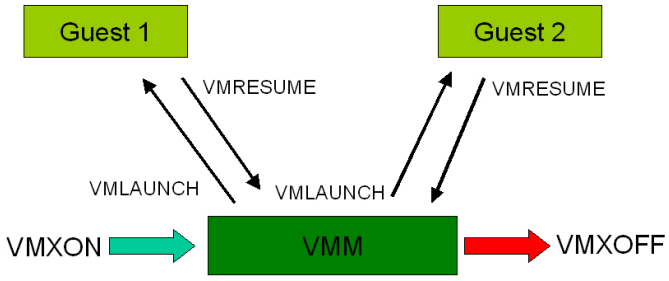

Fig. 2.13: VMX operation

2. Attraverso le VM entries il VMM avvia il sistema guest nella macchina virtuale (ovviamente uno per volta), e ciò avviene laciando un'istruzione VMLAUNCH;

3. quando il guest tenta di effettuare un'operazione illecita, le VM exits restituiscono il controllo al VMM, il quale può gestire la causa della violazione e poi restituire il controllo al guest;

4. infine, quando il VMM viene spento, si esegue l'ultima delle VMX operation attraverso l'istruzione VMXOFF.

La Memoria

La tecnica hardware che permette di ottimizzare la gestione della memoria offrendo un secondo livello di paginazione viene generalmente definita come SLAT (Second Level Address Translation) oppure come *nested paging*. L'implementazione di Intel della SLAT è conosciuta con il nome di *Extended Page Table* (EPT) e fu introdotta nella microarchitettura di Nehalem dai processori della serie Core i3, i5 e i7. AMD invece implementa la SLAT attraverso la tecnologia chiamata *Rapid Virtualization Indexing* (RVI).
Attraverso l'uso dell'EPT, il sistema operativo del guest può continuare ad effettuare il mapping tra le guest pages e le physical pages, ma il VMM viene agevolato nella sua opera di sincronizzazione con le machine pages attraverso un ulteriore livello, chiamato appunto nested page tables. Così le gPT

mappano gli indirizzi logici e fisici del guest, mentre le nPT mappano gli indirizzi fisici del guest con quelli fisici della macchina host. Diversamente dalla tecnica di shadow paging, le modifiche alle varie tabelle non vengono gestite dal VMM: il guest si occupa di gestire la sua memoria, mentre il MMU della macchina host si occupa di gestire le nPT (vedi Fig. 2.14). Questo ovviamente riduce moltissimo l'overhead del VMM dovuto alla gestione del shadow paging.
Purtroppo però non si è completamente sicuri dei benefici di questa tecnica in termini di prestazioni in quanto l'aggiunta di un livello di indirizzamento comporta un doppio accesso alla memoria, che si traduce in un aumento della latenza; sono stati svolti diversi studi al riguardo, volti a scoprire in quale circostaza sia preferibile utilizzare il supporto hardware o la tecnica software [10].

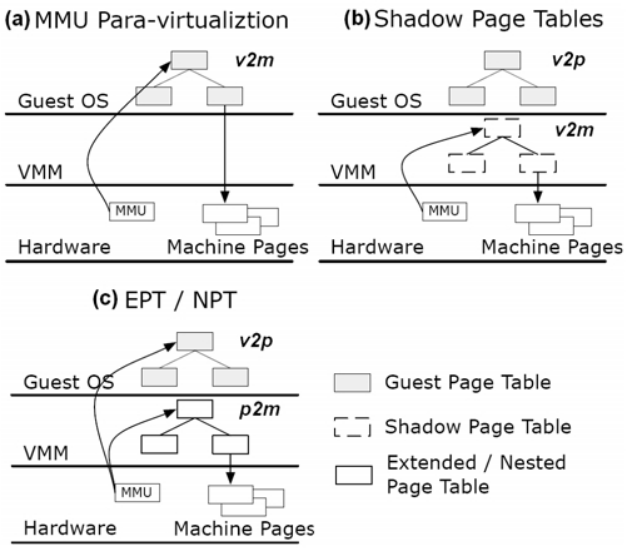

Fig. 2.14: Comparazione tra tecniche di virtualizzazione della memoria [10]

Le periferiche di I/O

Uno dei requisiti chiave per quanto riguarda la virtualizzazione delle periferiche di I/O consiste nell'abilità di salvaguardare l'isolamento e di limitare gli accessi alle risorse. La tecnologia di Intel chiamata "Directed I/O" fornisce al VMM i mezzi per controllare direttamente l'assegnamento del device di I/O alla macchina virtuale, una funzionalità definita *DMA remapping* che facilita il lavoro di traduzione degli indirizzi per il Direct Memory Accesses ed inoltre l'*interrupt remapping* per garantire la separazione tra i device usati dalla macchina virtuale e quelli usati dalla macchina host.
In particolare, è il DMA remapping a ricoprire il ruolo principale. Esso infatti fornisce un supporto hardware per isolare l'accesso del dispositivo alla memoria e permette ad ogni device del sistema di essere assegnato ad una specifica macchina virtuale attraverso un distinto insieme di pagine. Quando il dispositivo tenta di accedere alla memoria, il DMA-remapping intercetta tale accesso e utilizza le page tables per determinare se tale accesso può essere permesso o meno.
I limiti della virtualizzazione software dell'I/O possono essere aggirati attraverso l'assegnamento diretto delle risorse alle macchine virtuali. Grazie a questo approccio i driver di tale risorsa gira unicamente nella VM a cui è stato assegnato e ciò permette al guest di interagire direttamente con il dispositivo, con il minimo coinvolgimento del VMM (vedi Fig. 2.15).

2.6 I benefici della virtualizzazione

Come anticipato nell'introduzione di questo capitolo, si è ricorsi alla virtualizzazione principalmente per far fronte allo spreco di hardware sottoutilizzato ed agli elevati costi di gestione che la precedente politica aveva causato. Ora non ci resta che descrivere l'impatto che la virtualizzazione ha avuto sui sistemi, cercando di capire quali sono i benefici introdotti da questa nuova tecnica di gestione:

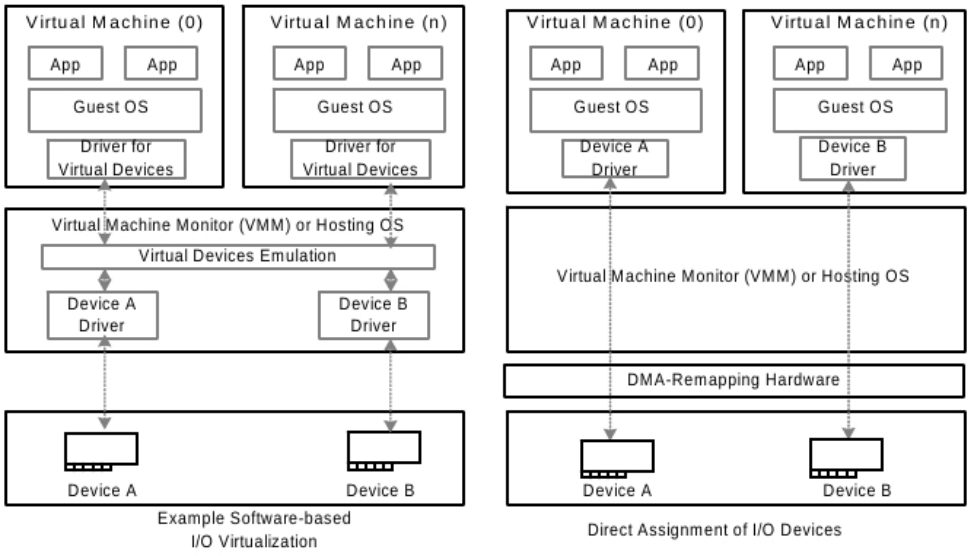

Fig. 2.15: Comparazione tra tecniche di virtualizzazione dell'I/O [11]

- **Isolamento**: tutto ciò che viene eseguito all'interno di una macchina virtuale non intacca minimamente il sistema host, perciò l'esistenza di bug in alcune applicazioni o addirittura nel sistema operativo ospitato non si ripercuote nè sulla macchina ospitante nè sulle eventuali altre macchine virtuali esistenti.

- **Flessibilità**: ogni macchina virtuale può essere spenta, messa in pausa, avviata e recuperata come qualsiasi macchina fisica. Ma soprattutto è persino possibile adattare l'hardware virtualizzato in base alle necessità.

- **Disponibilità**: se l'host su cui è eseguita una macchina virtuale necessita di manutenzione, attraverso la migrazione "a caldo", è possibile spostare la VM su un altro host fisico, pur mantenendola sempre attiva.

- **Scalabilità**: un sistema basato su macchine virtuali comporta meno restrizioni sulla scalabilità in quanto, nel caso in cui fossero richieste maggiori capacità al sistema, basterà inserire un nuovo nodo fisico con

una installazione di base, suddividendo poi il carico del lavoro anche su di esso.

- **Utilizzo dell'hardware**: ora viene sfruttata la quasi totalità dell'hardware presente in un nodo fisico.

- **Sicurezza**: l'isolamento comporta anche dei vantaggi in termini di sicurezza nel servizio offerto da una VM, nei confronti di quelli offerti dalle altre VM del sistema.

- **Costi**: come anticipato, i costi di gestione e manutenzione vengono proporzionalmente ridotti rispetto alla quantità di hardware ora sfruttato.

- **Applicazioni Legacy**: non è raro trovare nelle sale macchine delle aziende dei vecchi server con sistemi operativi obsoleti (come, ad esempio, DOS) che non possono essere spostati su macchine nuove a causa delle numerose incompatibilità. Gli ambienti di virtualizzazione rendono comunque eseguibile tale sistema operativo, permettendo però ai responsabili IT di liberarsi del vecchio hardware.

- **La creazione di ambienti di Test**: capita frequentemente che su un sistema in produzione si debbano apportare delle modifiche rischiose, che potrebbero causare delle spiacevoli conseguenze; la virtualizzazione permette la replica immediata di una macchina virtuale al fine di poter effettuare tutti i test necessari nella più totale sicurezza.

D'altra parte non si può fare a meno di citare anche gli svantaggi che la virtualizzazione comporta:

- **Overhead**: anche nel caso in cui l'hardware disponga di tecniche che agevolano la virtualizzazione, vi è comunque un percepibile calo delle prestazioni rispetto al sistema nativo;

- **Single Point of Failure (SPOF)**: nonostante la risorsa virtuale venga disaccopiata da quella fisica, il funzionamento dipende comunque dall'hardware.

2.7 Conclusione del capitolo

In questo capitolo abbiamo affrontato il concetto di virtualizzazione, partendo dalla sua definizione più generica, per poi addentrarci nella spiegazione delle varie tecniche di realizzazione. Ogni tecnica ha i suoi punti di forza e i suoi difetti perciò per poter ottenere dei buoni risultati occorre sempre cercare la soluzione più adatta per lo scenario in cui si intende operare.

Capitolo 3

Virtualizzazione nel Cloud

Il termine "cloud" può assume moltissime sfaccettature, infatti viene anche utilizzato come metafora per la rete Internet. Questa proliferazione di significati è avvenuta in modo relativamente graduale, sia a causa della generalità del concetto espresso che dall'uso improprio del termine da parte della società.
La parola *cloud* (letteralmente "*nuvola*") infatti è nata con lo scopo principale di astrarre la descrizione un sistema informatico strutturalmente complesso al fine di nasconderne i dettagli tecnici implementativi, ponendo invece enfasi nell'esposizione delle caratteristiche e delle funzionalità globali. Così facendo si focalizza l'attenzione dell'interlocutore sulla valutazione dei requisiti funzionali del sistema informatico e non sulla sua realizzazione tecnica, nascondendo nella "nuvola" tutti i particolari implementativi non necessari.
In questo caso però vogliamo riferirsci ad un concetto molto più specifico: il *Cloud Computing*.
Qui di seguito andremo a spiegare proprio il significato di questo concetto ed a descrivere le tipologie di servizi che ne conseguono. Inoltre vedremo anche quello che può essere definito come il logico passo successivo nella strada dell'evoluzione tecnologica: la *nested virtualization*.

3.1 Cloud Computing

Parlando in termini basilari, l'espressione cloud computing è utilizzata per descrivere una serie di scenari in cui le risorse computazionali sono fornite come servizio per mezzo di una connessione alla rete. Perciò può essere visto come un mezzo per fare affidamento sulla condivisione di un insieme di risorse fisiche e/o virtuali, piuttosto che sull'istanziazione di tali risorse in proprio o in locale.
Ed è proprio dicendo così che ritroviamo il concetto di "nuvola": l'obiettivo del cloud computing, infatti, è di svincolare l'amministrazione delle risorse computazionali necessarie alla realizzazione di un sistema da quello che è l'utilizzo del sistema stesso.
Le conseguenze di tutto ciò, in termini di vantaggi, sono innumerevoli:

Flessibilità. È uno strumento utile sia a realtà aziendali di piccole dimensioni che alle multinazionali. Le piccole imprese che non hanno i mezzi per sviluppare un'infrastruttura informatica completa in locale, affidandosi al cloud, riuscirebbero ad ottenere ciò che vogliono ma ad un costo notevolmente inferiore, sia in termini di investimento iniziale che in termini di spese di gestione e mantenimento.
Dal punto di vista delle multinazionali invece, il problema non consiste nella realizzazione dell'infrastruttura informatica, ma piuttosto nella volontà di sfruttare appieno le capacità computazionali di cui si dispone già, mettendone una parte a disposizione di altri utenti della rete.

Scalabilità. Un sistema informatico localizzato in un claud e basato su un'infrastruttura condivisa, offre un funzionamento simile a quello dei servizi pubblici: l'utente paga solo per ciò che intende adoperare e si disinteressa di come il sistema che utilizza sia organizzato nella realtà. In questo modo, si ottiene un'enorme libertà di scelta nel definire il quantitativo di risorse da allocare per il proprio scopo, soprattutto perché vi è la certezza che, qualora tale quantitativo si rivelasse inadeguato (sia

in difetto che in eccesso), si potrebbe tranquillamente riequilibrare la situazione in qualsiasi momento, con le naturali conseguenze sul costo del servizio.

Sicurezza. Delegando al gestore del cloud il controllo dell'architettura del sistema, si ottengono anche delle tutele che garantiscono al consumatore una certa garanzia sull'affidabilità e sulla disponibilità continua del servizio.

Localizzazione. Essendo un servizio fornito attraverso la rete, è possibile raggiungerlo in qualsiasi situazione, purchè si disponga di un collegamento ad Internet.

3.1.1 Le tipologie di servizi

In generale il cloud computing può essere catalogato in tre modi, in base al tipo di servizio offerto:

IaaS

IaaS sta per Infrastructure as a Service. Come abbiamo detto in precedenza, questi servizi si basano sul fornire delle risorse virtuali ai propri clienti per mezzo della rete.
IaaS, nello specifico, si occupa di gestire le risorse hardware presenti nel cloud al fine di offire agli utenti la possibilità di avere delle macchine virtuali complete, comprensive perciò di un sistema operativo, disponibilità di storage, connessione alla rete, indirizzi IP pubblici o privati, ed infine aventi delle potenzialità computazionali di alto livello.
Per poter provvedere a tutto questo, il provider, cioè il fornitore del servizio di IaaS, controlla e gestisce uno vasto sistema architetturale, composto da un gruppo di server distribuiti all'interno di un data center.
In genere, oltre alle macchine virtuali, vengono forniti anche dei pacchetti di utility che permettono agli utenti dotati di una certa esperienza in ambito

informatico, di poter interagire in modo più accurato con i propri dispositivi virtuali.

PaaS

Il PaaS, o Platform as a Service, può essere anche visto come un'estensione del IaaS. Esso rappresenta una categoria di cloud computing che fornisce agli sviluppatori software un ambiente di programmazione in cui scrivere applicazioni, sempre accessibile attraverso la rete.
In aggiunta alle fondamentali risorse computazionali, le postazioni appartententi al PaaS danno anche in dotazione dei software e delle utility per le configurazioni (conosciute anche con il nome di *solution stack*) necessarie per creare una piattaforma su cui i clienti possono creare le loro applicazioni. Inoltre, la gestione dell'ambiente di sviluppo concede ai programmatori un ampia possibilità di personalizzazione, sia aggiungendo o togliendo pacchetti applicativi, che organizzando e suddividendo il processo di sviluppo software in vari team.

SaaS

Tra le diverse forme di cloud computing, probabilmente il SaaS (Software as a Service) è il servizio più diffuso. Esso infatti consiste proprio nello sfruttare un provider per delocalizzare un software allo scopo di metterlo poi a disposizione degli utenti attraverso la rete. Praticamente si basa sul concetto di non far pagare agli utenti l'intero software, ma solo il suo utilizzo. La versatilità e la semplicità di questa tipologia di servizio hanno reso questo sistema di distribuzione del software molto popolare negli anni.

3.1.2 I servizi esistenti

Per quanto riguarda il SaaS, possiamo citare il software Office 365, che consente l'utilizzo dei prodotti come Word o Excel. Ma anche i vari servizi di posta elettronica web, come Gmail o Outlook, ne sono altri esempi.

In ambito del PaaS, invece, possiamo segnalare Windows Azure di Microsoft, grazie al quale è possibile progettare applicazioni con la piattaforma .NET; oppure, come non menzionare il Cloud Platform di Google con la quale, il colosso della ricerca su internet, mette a disposizione le proprie infrastrutture allo scopo di fornire un ambiente software per altri programmatori.
Infine, passando alla forma di cloud computing più di interesse per lo scopo di questo elaborato, possiamo citare la famosissima piattaforma di Amazon EC2, disponibile sul mercato dal 2006. Essa offre ai propri utenti la possibilità di creare un sistema cloud server scalabile (con valori differenti di CPU e RAM), comprensivo di storage e networking, ma concedendo la possibilità di pagare il servizio esclusivamente in base alle singole ore di computazione. Oltre ad Amazon, esistono molti altri provider tra cui Ganeti, Aruba, HostingSolution, ecc...
Per poter creare le varie macchine virtuali, i provider di servizi di IaaS utilizzano, solitamente, i più comuni software di virtualizzazione: VMware, Hyper-V, XEN e KVM.

Una limitazione esistente in questo tipo di infrastrutture virtuali è la loro estrema dipendenza dall'ambiente virtuale con cui sono stati creati. Infatti, proviamo a prendere nuovamente ad esempio il servizio offerto da Amazon, in cui la parte riguardande la virtualizzaiozne viene affidata al software Xen. In Amazon EC2 ci si aspetta che i sistemi guest che girano nell'infrastruttura siano stati creati utilizzando un particolare formato immagine: il cosiddetto Amazon Machine Image (AMI) format. Questo formato costituisce l'unità fondamentale di creazione all'interno di Amazion EC2 e, per realizzare una propria macchina virtuale, si può scegliere una tra le miriade di diverse configurazioni preconfigurate usufruibili, basate sulle diverse combinazioni di sistemi operativi e applicazioni preinstallate disponibili.
L'esistenza di questo "formato di macchina virtuale", strettamente dipendente dal software di virtualizzazione che lo ha generato, può diventare un ostacolo per un utente del cloud, specialmente nel caso in cui egli decida di

migrare[1] la propria VM, ad esempio passando da un'infrastruttura privata ad una pubblica.

3.2 Nested Virtualization

Fino a circa dieci anni fa, quando la parola "cloud" non era ancora un termine cosí diffuso, non si poteva certo immaginare quanto sarebbe diventato popolare. Ma, come avviene spesso in campo tecnologico, i servizi cambiano e progrediscono in base alla loro validità e alla richiesta di mercato.
Analogamente è avvenuto per le tecnologie di virtualizzazione le quali, sebbene siano nate con lo scopo di ottimizzare le risorse hardware a propria disposizione, adesso sono perlopiù orientate a fornire agli utenti, servizi di cloud computing su larga scala, concedendogli l'utilizzo di risorse senza però che questi ne abbiano il pieno possesso.
Al giorno d'oggi, il passo evolutivo successivo in un contesto comprendente

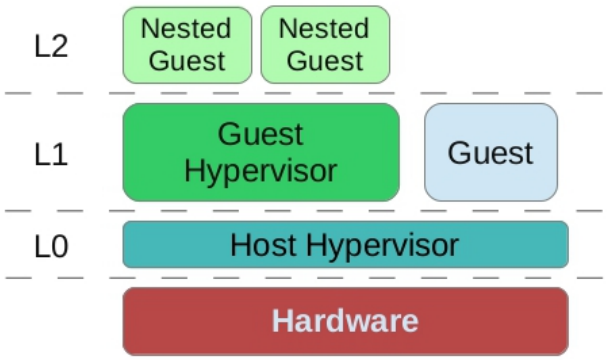

Fig. 3.1: Versione semplificata di un'architettura con Nested Virtualization

il cloud e la virtualizzazione è sicuramente la Nested Virtualization.
In una situazione classica di virtualizzazione abbiamo un hypervisor che permette a più sistemi operativi di girare simultaneamente su una macchina

[1]La migrazione consiste nello spostare in blocco un'intera macchina virtuale in esecuzione da un server fisico all'altro, preferibilmente senza downtime.

fisica. Nell'ottica della virtualizzazione annidata, invece, un hypervisor può permettere di eseguire altri hypervisor, ognuno associato con le proprie macchine virtuali (Fig. 3.1).
Grazie alla virtualizzazione annidata, un provider di IaaS potrebbe dare ai propri utenti la possibilità di far girare all'interno delle proprie macchine virtuali del cloud un hypervisor, e perciò anche una macchina virtuale ad esso collegata, che sia sotto il suo diretto controllo. In questo modo si avrebbe un duplice vantaggio per i soggetti in gioco:

- l'utente del cloud avrebbe la possibilità di maneggiare direttamente una propria macchina virtuale, scegliendo il suo software di virtualizzazione preferito;
- il provider del cloud potrebbe avere un'ulteriore attrattiva nei confronti di quegli utenti interessati ad approfittare di questa possibilità.

L'implementazione della nested virtualization porta con sé molte conseguenze dal punto di vista della crescita informatica:

◊ Creare un sistema che permetta lo sviluppo e il test di software cluster bastato sulla virtualizzazione, utilizzando semplicemente una singola macchina fisica;

◊ Creazione di framework per il debugging di hypervisor;

◊ Facilitare lo sviluppo e il testing di software distribuito permettendo la creazione di un cluster virtuale su una singola macchina virtuale;

◊ Rendere possibile la migrazione di vm anche tra cloud provider che usano diversi formati di macchina virtuale.

3.2.1 Possibili Implementazioni

Per poter realizzare la virtualizzazione annidata potremmo utilizzare una delle tecniche descritte nel capitolo precedente (vedi Cap. 2), ma non tutte

sarebbero applicabili allo stato attuale dell'arte. In alcune infatti occorrerebbe effettuare delle modifiche al fine di rendere possibile una compatibilità tra hypervisor.

Dynamic Binary Translation

Come spiegato in precedenza (vedi Sez. 2.4.1), grazie a questa tecnica è possibile utilizzare la virtualizzazione anche su un sistema non naturalmente virtualizzabile, e cioè su una macchina che non dispone neanche del meccanismo di trap (vedi Sez. 2.3.1) per catturare e arginare le istruzioni privilegiate lanciate dal sistema operativo guest.
Questa tecnica si basa sul realizzare l'intero ambiente virtualizzato utilizzando solo mezzi software, traducendo "on the fly" le istruzioni privilegiate in un nuovo blocco di codice che vada ad agire sulla macchina virtuale, e non sull'host. In questo modo, né il guest né l'host, si accorgono l'uno della presenza dell'altro, verificando così la Full Virtualization.
In base a quanto appena detto, in teoria, si potrebbe realizzare una virtualizzazione annidata utilizzando la DBT sia nell'hypervisor di base che in quello nested. Ovviamente, il vincolo basilare di tale contesto è che i sistemi virtualizzati uno dentro l'altro, devono essere conformi all'architettura x86; ciò vuol dire che non è possibile installare su di essi un sistema operativo a 64 bit.
Tale implementazione, sebbene con bassi livelli prestazionali, renderebbe possibile l'implementazione della nested virtualization senza dover minimamente modificare nessuno degli hypervisor in gioco, né la macchina ospitante.
Una variante consisterebbe nell'utilizzare come nested guest un sistema operativo paravirtualizzato, in modo da evitare l'onero della gestione della *nested* shadow page tables (vedi Sez. 2.4.2).

Paravirtualization

Ricordiamo che in un sistema paravirtualizzato, il sistema operativo guest è perfettamente a conoscenza di non trovarsi direttamente sopra un hardware

fisico ma su di uno virtuale (vedi Sez. 2.3.2). Esso infatti sfrutta le *hypercalls* per comunicare con il VMM[2] al fine di gestire in modo ottimale la virtualizzazione.
Ciò significa che, nel contesto della paravirtualizzazione, tutto il codice del guest contenente istruzioni non virtualizzabili, necessita di subire delle opportune modifiche per essere in grado di dialogare con il VMM.
Nell'ottica della virtualizzazione annidata, perciò, il nested hypervisor dovrebbe necessariamente essere modificato per potergli permettere di interagire con queste hypercalls. Se si ha a disposizione un bare-metal hypervisor, o anche detto nativo, lo si dovrebbe adattare modificandone tutto il codice contenente istruzioni privilegiate. Mentre, nel caso di un hypervisor di tipo hosted, occorrerebbe fare in modo che, il modulo che viene caricato nel kernel dal sistema operativo host, sia in grado di poter lavorare in un ambiente paravirtualizzato.
Di conseguenza, per rendere possibile la virtualizzazione annidata tramite la paravirtualizzazione, le compagnie che svilluppano tali software dovrebbero creare prodotti in grado di rendere i loro hypervisor compatibili anche nel caso di esecuzione all'interno di un guest paravirtualizzato.
Attualmente, non sono presenti sul mercato degli hypervisor che implementano tali funzionalità.

Hardware Assisted

Grazie a questo strumento fornito, come si intuisce dal nome, direttamente dall'hardware della macchina fisica, è possibile ottimizzare notevolmente il processo di virtualizzazione, per mezzo di accorgimenti che velocizzano l'interazione tra guest e host e che migliorano la gestione della memoria e dell'I/O (per maggiori dettagli, vedere Sez. 2.5). Inoltre, come con la Dinamic Binary Translation, il sistema operativo guest non necessita di essere modificato, in quanto non si rende conto di girare su un hardware virtualizzato.

[2]Virtual Machine Monitor

In questo caso, per poter sfruttare i vantaggi dell'hardware assisted virtualization anche nella nested guest, è assolutamente fondamentale che la macchina virtuale più vicina all'hardware, e perciò l'hypervisor di base, venga modificato al fine di permettere una condivisione di una parte delle funzionalità garantite da questa tecnologia con l'hypervisor annidato.

3.2.2 VMM e Nested Virtualization

Non tutti i software di virtualizzazione esistenti sul mercato dispongono di meccanismi per agevolare la nested virtualization. L'implementazione di tali funzionalità risulta infatti particolarmente complicata e dispendiosa. Di conseguenza, solo quelle compagnie che hanno successo in ambito commerciale hanno deciso di addentrarsi in questo settore.
Tra i vari software di virtualizzazione che si sono cimentati nell'implementazione della nested virtualization, il primo fra tutti che è riuscito a realizzarla è stato KVM, all'interno del Turtles Project di IBM. Anche per questo motivo abbiamo deciso di approfondire la sua implementazione.

KVM

Il progetto Turtles di IBM [15] prevede la realizzazione della la virtualizzazione annidata tramite l'utilizzo dell'hypervisor di KVM e servendosi delle tecnologie hardware di virtualizzazione della Intel.
Teoricamente esso è in grado di ospitare più guest hypervisor uno dentro l'altro, ognuno con al suo interno il sistema operativo guest. Dal punto di vista pratico, sono stati effettuati dei test in cui si prendevano in esame come guest hypervisor sia lo stesso KVM che un altro noto software come VMware Server (entrambi in versione non modificata[3]) e due comuni sistemi operativi come Linux e Windows come nested vm. Visto che tali entità annidate

[3]Con versione non modificata si tende dire che i due hypervisor non sono coscienti di essere "guest".

non sono state modificate in alcun modo in quanto prese in blocco, *si può supporre* che il progetto Turtles possa permettere l'esecuzione di una nested virtualization di qualsiasi sistema dotato di architettura x86.

Innanzitutto occore precisare che, in base al diverso livello di supporto fornito dall'architettura sottostante alla nested virtualization, esistono due tipi di implementazioni possibili:

Supporto architetturale multi livello. In base a questa tecnica, ogni hypervisor gestisce tutti i trap causati dalle istruzioni privilegiate del guest hypervisor posizionato esattamente sopra di esso. Questo comporta una grossa collaborazione da parte del sistema hardware che deve conoscere la presenza di tutti gli hypervisor e segnalare opportunatamente l'interrupt a quello che ha la competenza di gestirlo. Per fare ciò mette a disposizione tanti "hypervisor mode" quanti sono i livelli di annidamento. Un esempio di questo modello è implementato con successo nell'IBM System Z [14].

Supporto architetturale a singolo livello. In questo caso si ha a disposizione un solo "hypervisor mode" e le interruzioni provenienti da uno qualsiasi dei livelli annidati, vengono gestiti da esso. Questo modello è implementato sia da Intel che da AMD nelle loro rispettive estenzioni per la virtualizzazione, VMX e SVM (vedi Sez. 2.5).

In questo caso, come accennato in precedenta, si sta sfruttando come hardware quello dell'architettura Intel, e perciò viene fatto uso del secondo modello.
Visto che Intel mette a disposizione un livello di virtualizzazione singolo, un solo hypervisor può effettivamente utilizzare le istruzioni VMX per eseguire il proprio guest. Di conseguenza, per poter permettere ad un guest hypervisor di utilizzare tali istruzioni privilegiate, l'hypervisor posizionato nel livello L0, deve necessariamente emulare le funzionalità delle VMX all'hypervisor di livello annidato.

Ovviamente si può ipotizzare che, utilizzando un guest hypervisor che a sua volta emuli le stesse funzionalità, tale modello possa fonzionare anche in modo ricorsivo.
Per riassumere brevemente l'idea teorica alla base di questa implementazione, è sufficiente dire che si basa tutto su un lavoro di multiplexing: i trap causati da una qualsiasi delle macchine virtuali annidate vengono catturati dall'hardware emulato e inoltrati al livello sottostante, fino a giungere al principale bare-metal hypervisor.
È possibile capire meglio il processo appena descritto, osservando l'immagine della Fig. 3.2. Possiamo infatti riscontrare come, permettendo solo all'hypervisor principale di gestire le istruzioni VMX, di fatto, equivale a posizionare il livello L2 subito sopra di esso.
Concludiamo, precisando che il progetto Turtles è sicuramente da consider-

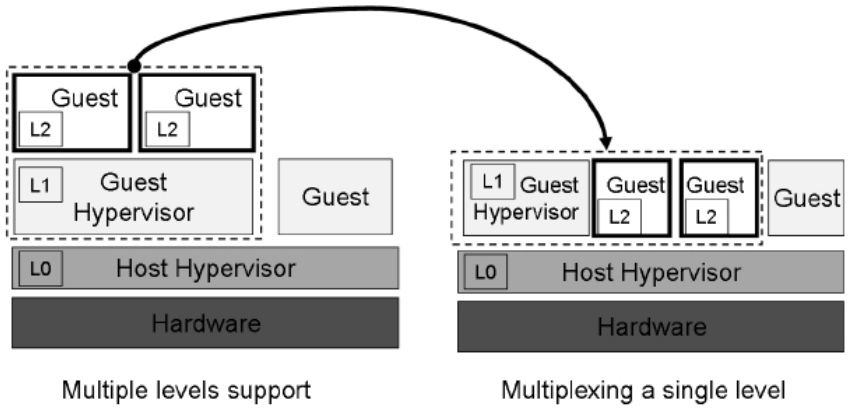

Fig. 3.2: Progetto Turtles di IBM con KVM [15]

arsi maturo: esso è stato testato eseguendo più hypervisor simultaneamente, supporta il SMP, sfrutta la tecnica di page table a due dimensioni messa a disposizione dall'hardware per implementare una nested MMU virtualization, ed infine è persino capace di far uso di un multi-level device assignment per rendere più efficente la nested I/O virtualization.

3.3 Conclusione del capitolo

In questo capitolo abbiamo affrontato il concetto di Cloud, ponendo particolare attenzione alle possibili tipologie di servizi disponibili ed ai conseguenti vantaggi per gli utenti.
Inoltre, abbiamo anche presentato una nuova forma di design architetturale basato sulle macchine virtuali: la nested virtualization. Essendo ancora un concetto relativamente giovane, non si dispone ancora di sufficienti informazioni sulla compatibilità dei vari software di virtualizzazione, ma dimostra già di essere un promettente inizio per realizzare nuovi e complessi sistemi architetturali.

Capitolo 4

Il tempo nei calcolatori

Il concetto di "tempo" in un calcolatore è molto vasto e può essere identificato da diversi punti di vista: clock, time-of-the-day, interrupt, tempo di sistema, ecc... Sebbene questi concetti siano correlati tra loro, ognuno di essi ha un significato differente.
Prima di tutto occorre premettere che esistono due grandi modi per scandire il passaggio del tempo: uno hardware e uno software. Come si può facilmente immaginare, l'orologio hardware gira continuamente perché alimentato da una sua batteria, mentre l'orologio software si accende quando il computer si avvia e si spegne con esso. Inoltre, mentre l'orologio hardware viene implementato per mezzo del timer device presente nel sistema, il clock software dipende soprattutto da come il sistema operativo utilizzato decide di sfruttare l'hardware di cui dispone.
Ora cercheremo per prima cosa di fare una rapida panoramica illustrando singolarmente alcuni dei concetti chiave.

4.1 Il clock

Il processore è composto da vari circuiti logici preposti ad eseguire operazioni diverse ed, inoltre, ad interagire tra loro scambiandosi informazioni. Affinché tale scambio di informazioni avvenga correttamente, è necessario

che ad ogni circuito sia indicato il momento esatto in cui può ritenere validi i segnali che riceve in ingresso. Il clock è il segnale che si occupa di questo ed è condiviso tra tutti i circuiti.
Il clock, dal punto di vista elettronico, non è altro che un oscillatore[1] che genera un segnale di frequenza nota e stabile, generalmente un'onda quadrata, dal quale si ricavano impulsi a intervalli di tempo regolari chiamati *clock signal*. L'intervallo di tempo tra i fronti corrispondenti di due impulsi consecutivi è detto tempo di *ciclo del clock*, mentre la frequenza, misurata in Hertz, indica il numero di impulsi nell'unità di tempo.
La CPU richiede un numero fisso di cicli di clock per eseguire ogni operazione, perciò possiamo dire che più velocemente gira il clock e più istruzioni per secondo possono essere eseguite dalla CPU. Ad esempio, un processore che ha una frequenza di 3 GHz è in grado di generare $3 * 10^9$ tick al secondo.
Tutti i timer device esistenti potrebbero essere rappresentati rudemente utilizzando il diagramma a blocchi della Fig.4.1. Nella figura possiamo notare

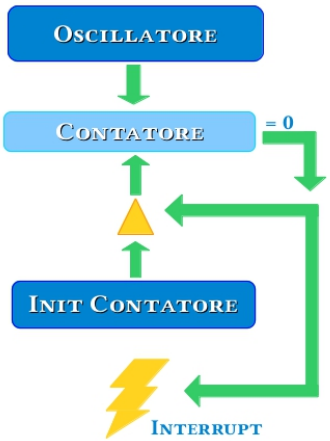

Fig. 4.1: Diagramma generico di un clock.

tre entità chiave:

Oscillatore dispositivo che si occupa di fornire degli input ad una certa frequenza fissata, la quale puó essere definita dal dispositivo stesso

[1]Solitamente si usa un oscillatore in quarzo per la sua stabilità di scillazione

oppure misurata dal sistema operativo durante l'avvio della macchina.

Contatore in generale può essere scritto/letto mediante software e viene decrementato (o incrementato in base ai casi) di un unità ogni ciclo dell'oscillatore; quando raggiunge lo zero genera in output un segnale (interrupt).

Init Contatore (facoltativo) è un registro che mantiene il valore di inizializzazione del contatore, una volta che questo ha raggiunto lo zero; ciò permette di controllare a livello software il periodo di tempo tra gli interrupt.

Esistono diversi tipi di timer device e qui di seguito ne descriveremo alcuni tra i più conosciuti e implementati negli odierni calcolatori.

4.1.1 Timer Device

PIT

L'acronimo sta per Programmable Interval Timer ed è uno dei più antichi tra i timer device. Usa un oscillatore crystal/controlled a 1.193182MHz, un contatore a 16 bit e un registro. La frequenza dell'oscillatore non è stato scelto per ragioni di convenienza ma solamente perché era una frequenza ad uso comune e disponibile nel periodo in cui il primo PC fu ideato[2].
Il dispositivo PIT in realtà contiene tre timer identici, connessi in modi diversi al resto del computer:

- *Timer 0* può generare interrupt ed è idoneo per il timekeeping del sistema;
- *Timer 1* viene storicamente usato per fare il refresh della RAM ed è tipicamente programmato ad un intervallo di 15μs;
- *Timer 2* è collegato agli speaker del PC per la generazione del suono.

[2]La frequenza dell'oscillatore è pari ad un terzo dello standard video NTSC

Il device APIC (Advanced Programmable Interrupt Controller), che vedremo più avanti, consiste in una rivisitazione in versione più recente offerta dai sistemi Intel del PIT. Viene preferito al PT soprattutto nei kernel Linux dalla versione 2.6.18 in poi.

CMOS RTC

Il CMOS viene anche chiamato *nonvolatile BIOS memory* in quanto è un dispositivo sostenuto a betteria che ha il compito di mantenere memorizzati i settaggi del BIOS del PC stabili, mentre si sta spegnendo. Il nome deriva dalla tecnologia a basso consumo energetico complementary metal-oxide semiconductor, utilizzata in elettronica per la progettazione di circuiti integrati, con cui era originariamente implementato.
Ovviamente, più che la memorizzazione di dati del BIOS, a noi interessa la parte il relazione al Real Time Clock. A tal proposito, ci sono due principali funzinalità in relazione al tempo realizzate in questo dispositivo:

- la prima consiste nel TOD, un orologio perennemente in attività, atto a mantenere il èmphtime of day nel formato anno/mese/giorno ora:minuti:secondi;

- la seconda riguarda la generazione di interrupt periodici ad un rate di ogni potenza di due tra 2Hz e 8,192Hz.

Per quanto riguarda gli interrupt, vogliamo sottolineare che il timer è comunque conforme al modello nella Fig.4.1 con la differenza che il contatore non può essere letto o scritto e che il valore di inizializzazione può essere solo una potenza di due.
Inoltre, possono essere attivati anche altri due tipi di interrupt: l'interrupt di update e quello di allarme. Il primo avviene una volta al secondo e si ipotizza che rifletta il giro di orologio del TOD al secondo successivo, mentre il secondo si verifica quando una determinata data del time of day corrisponde esattamente con un valore specificato o con un pattern impostato.

Local APIC Timer

Come abbiamo precedentemente accennato, il sistema APIC progettato dalla Intel è l'evoluzione in chiave odierna del PIT, ideato però per essere utilizzato in sistemi di elaborazione che prevedono sistemi multiprocessore simmetrici. Il local APIC è uno dei due componenti di cui il sistema APIC è composto, insieme all'I/O APIC. Entrambi gestiscono gli interrupt ma provenienti da dispositivi diversi: il LAPIC per i processori e l'I/O APIC per tutte le periferiche.
Nelle tecnologie multiprocessore, esiste un LAPIC per ogni processore ed inoltre, nei processori più recenti, esso è persino stato integrato all'interno del chip del processore. Il loro compito è quello di gestire tutte le interruzioni esterne per conto del processore di cui fanno parte ed, inoltre, sono anche in grado di accettare e generare IPI, Interruzioni Inter-Processor, tra diversi LAPIC.
L'intervallo di tempo dopo il quale il counter decrementa il suo valore dipende, tipicamente, dalla frequenza del front-side bus[3] (FSB) divisa per il valore del Divide Configuration Register. Di conseguenza, a differenza del PIT, il valore del rate del counter non è conosciuto a priori, ma varia in base alla conformazione della macchina.
Questo timer risulta essere particolarmente costante e preciso, ed inoltre dispone di un contatore maggiore del PIT o del CMOS; ma purtroppo, non essendo nota la sua frequenza, l'unico modo per determinare la misura esatta del timer del Local APIC è di misurarlo utilizzando un altro dispositivo (appunto il PIT o il CMOS) e poi "approssimarne" il risultato.

ACPI Timer

Il timer ACPI è di fatto un elemento aggiuntivo, chiesto formalmente come parte delle specifiche ACPI [16] , in cui viene anche chiamato Power Management (PM) timer oppure chipset timer. Il suo compito principale è

[3]Un interfaccia di comunicazione, utilizzata da Intel, per permettere ai diversi processori situati nel back-end di comunicare con l'esterno

quello di fornire un accurato valore del tempo, il quale viene usato dal sistema software per misurare e delineare l'eventuale inattività della macchina. È dotato di un contatore a 24 bit che viene incrementato ad una frequenza di 3.579545MHz (tre volte quella del PIT) che può essere programmato per generare un interrupt quando il suo bit più significativo passa da valore 0 a 1 (e viceversa); non necessita di un registro di inizializzazione in quanto è stato ideato per resettarsi automaticamente ogni volta che raggiunge il suo valore massimo, ricominciando poi a contare da zero.
Questo timer ha il grosso pregio di continuare a girare anche quando la macchina si trova in modalità di risparmio energetico, mentre gli altri timer vengono invece bloccati o rallentati. C'è anche da precisare che la lettura del suo valore è piuttosto lenta (tipicamente 1-2μs).

TSC

Il Timer Stamp Counter è un contatore ciclico presente nei processori x86 sino dal Pentium. Esso non può generare interrupt e non necessita di alcun registro che ne contenga il valore iniziale.
Viene letto a livello software tramite l'istruzione `rdtsc`, la quale pur essendo normalmente disponibile anche in user mode, potrebbe essere resa inattiva a scelta del sistema operativo.
Sebbene sia considerato uno dei timer device più affidabile e di conveniente accessibilità, esso presenta diversi svantaggi:

- esattamente come il timer local APIC, non esiste un modo affidabile per misurare correttamente tramite software la sua frequenza. Generlamente l'unico modo è quello di misurarlo approssimativamente usando qualche altro device.

- esistono diverse tecnologie che operano sulla velocità del clock del processore, variandolo dinamicamente in base alle necessità; ciò fa modificare, in modo più o meno evidente, anche il rate dell'oscillatore del TSC.

- alcuni processori bloccano il TSC quando si trovano nello stato di risparmio energetico.

- in caso di macchine a bus condiviso, perciò quelle aventi più processori, tutti i TSC girano in base ad un comune oscillatore; di conseguenza, in assenza di casi particolari, essi possono essere sincronizzati talmente accuratamente da comportarsi come un unico enorme dispositivo[4].

A dispetto di questi svantaggi, il TSC viene utilizzato molto di frequente come punto di riferimento per il tempo, sia dai vari sistemi operativi che dalle applicazioni.

HPET

L'High Precision Event Timer è uno strumento relativamente nuovo, Esso fu sviluppato unitariamente tra Intel e Microsoft e venne incorporato nei PC dal 2005 circa con lo scopo di evitare la confusione tra tutti i timer multimediali introdotti nel Multimedia Extensions a Windows 3.0 [18].
In realtà molte macchine non dispongono di questo dispositivo ed i sistemi operativi non ne richiedono obbligatoriamente la presenza, però se ne fa comunque uso se ne viene riscontrata la presenza.
L'HPET possiede un unico contatore centrale, a 32 o 64 bit, che gira continuamente (a meno che non venga bloccato a livello software) ed il suo valore di inizalizzazione può essere letto dal registro. Esso fornisce dei timer multipli, ognuno dei quali consiste in un registro di timeout che viene comparato con il valore del contatore centrale; quando un registro corrisponde col valore del timer, scatta l'interrupt. Se il timer è stato impostato per essere ciclico, l'hardware del HPET aggiungerà automaticamente il proprio valore attuale al registro, al fine di impostarlo per il successivo interrupt.
Anche l'HPET ha alcuni svantaggi, infatti le specifiche di questo dispositivo non richiedono in alcun modo che esso abbia una grana particolarmente fine,

[4]Ciò non vale per le macchine IBM X-Series NUMA e derivate, nelle quali c'è un oscillatore per ogni nodo NUMA [17]

o che mantenga un basso drift, o che sia veloce da leggere, perciò le implementazioni variano. Una tipica costruzione usa un contatore con frequenza di 18MHz e richiede circa 1-2μs per essere letto (come il timer ACPI).
Uno svantaggio di questo design generico è che il settaggio di un timeout è influenzato dalla "corsa" del contatore. Ad esempio, se un programma decidesse di settare un timeout breve ma, per una qualche ragione, la sua scrittura sul HPET venisse ritardata di un tempo sufficiente da permettere al contatore "sorpassare" il valore atteso, tale timeout verrebbe in realtà programmato per scattare in un lontanissimo futuro (dopo circa $2^3 2o2^6 4$ unità).
Il dispositivo HPET fu concepito per sostituire il PIT e il CMOS, al fine di guidare gli interrupt che solitamente erano dediti a gestire, ma attualmente ancora la maggior parte delle macchine hanno sia il timer PIT che il CMOS e non necessitano del HPET [19].

4.2 Tecniche di Timekeeping

Ovviamente, i primi ad essere interessati alla gestione e al mantenimento accurato del tempo in un calcolatore, sono i sistemi operativi.
Fino ad ora, abbiamo analizzato il funzionamento dei vari timer device i quali, pur essendo implementati con tecnologie differenti, sono accumunati dallo stesso funzionamento: l'incremento del contatore. In realtà però i sistemi operativi necessitano di tenere traccia anche di un valore più reale e "assoluto" del tempo, tale valore viene chiamato *wall-clock time*. Generalmente, il sistema operativo sfrutta il boot della macchina per leggere il valore iniziale del wall-clock time e poi, sfruttando appunto i timer device di cui dispone, fa in modo di tenerlo sempre aggiornato.
Per mantenere adeguato il proprio orologio interno mentre la macchina è attiva, i sistemi operativi generalmente misurano la quantità di tempo trascorso dal boot, usando una delle due seguenti tecniche di *timekeeping*:

Tick counting il sistema operativo delega ad un dispositivo hardware il compito di lanciare periodicamente degli interrupt, chiamati *tick*, che

sfrutterà al fine di determinare quanto tempo è trascorso;

Tickless l'hardware della macchina dedito alla funzione di timer effettuerà comunque l'aggiornamento periodico del contatore, ma senza lanciare alcun interrupt; sarà il sistema operativo a leggere il valore del contatore quando ne avrà la necessità.

Ora per prima cosa analizzeremo queste due tecniche, per capire i vantaggi e gli svantaggi di ognuna, poi andremo a fare una panoramica sui principali sistemi operativi in circolazione soffermandoci sulla loro specifica tecnica di gestione del tempo.

4.2.1 Tick Counting

È sicuramente la tecnica più sfruttata dai sistemi operativi ma, sfortunatamente, è anche quella più difficile da implementare in una macchina virtuale.
In generale le macchine virtuali condividono l'utilizzo dell'hardware con il sistema operativo della macchina host (vedi Cap. 2 per maggiori dettagli) e ciò comporta che il sistema guest debba coesistere insieme ad altre applicazioni del sistema host o, addirittura, insieme ad altre macchine virtuali che risiedono sulla stessa macchina fisica. Perciò, nel momento in cui la macchina virtuale dovrebbe generare un interrupt, essa potrebbe addirittura non essere in stato di "running", oppure potrebbe non essere attualmente schedulata dal sistema operativo host[5], portando la VM ad accumulare un *backlog* di molti interrupt.
Visto che la cognizione del tempo che il sistema operativo guest ha è data proprio dal conteggio degli interrupt, è ovvio che quando si è in presenza di un backlog la misurazione del tempo non può essere conforme al "tempo reale".
Una possibile soluzione consiste nel notificare più velocemente gli interrupt

[5]Molti sistemi operativi host non forniscono direttamente un modo per la VM di richiedere un timer interrupt ad un rate richiesto

al sistema guest, fino ad esauire il backlog; questa soluzione và gestita con molta cura in quanto, se si notifica un interrupt prima che il sistema operativo abbia terminato di gestire quello precedente, si verifica il fenomeno del *lost tick*. Oppure, un'altra soluzione più drastica, adottata ad esempio da VMware [19], consiste nel bloccare la ricezione degli interrupt di ritardo (settando il backlog a zero) e attivare la funzionalità di clock synchronization[6], al fine di gestire i casi in cui il backlog abbia raggiunto il limite dei 60 secondi di ritardo, aggiornando così il tempo di sistema.
Un'ulteriore questione da tenere in considerazione è la scalabilità, cioè come gestire gli interrupt nel caso in cui sulla stessa macchina fisica girino più macchine virtuali. Anche nel caso in cui una VM fosse in stato *idle*, essa dovrebbe comunque avere dei brevi intervalli di attività ogni volta che riceve un interrupt. Per capire bene le conseguenze che questa tecnica di timekeeping ha sulla macchina fisica, facciamo un breve esempio matematico: se anche ogni vm richiedesse 100 interrupt al secondo e ci fossero N macchine virtuali accese nello stesso momento, processare gli interrupt per ognuna di esse porterebbe ad avere un carico di lavoro di 100xN context switches al secondo, pur nel caso in cui le VM fossero tutte in stato idle.

4.2.2 Tickless

Un crescente numero sistemi operativi si sta affacciando a questa tecnica di timekeeping, e ciò avvantaggia di molto la gestione delle macchine virtuali. Grazie a questa procedura, infatti, non sarà più necessario intercettare gli interrupt del sistema e non sarà necessario neanche tenere un backlog, con l'enorme beneficio di risparmiare cicli di CPU, la quale in caso contrario, dissiperebbe gran parte della propria attività per gestire gli interrupt accumulati.
Per poter usufruire delle agevolazioni che questa tecnica comporta, occorre però identificare se il sistema operativo emulato dalla macchina virtuale la

[6]Questa funzionalità si attiva solo nel caso in cui VMware Tools sia installato nel sistema operativo guest

stia utilizzando o meno. Innanzitutto occorre definire una regola di partenza: *si suppone sempre la tecnica del tick counting, fino a prova contraria.* Ciò sta a significare che la macchina virtuale deve, di default, considerare che il sistema operativo faccia uso della tecnica tick counting, fino a che non abbia prova certa che il sistema sia tickless, perché altrimenti si troverebbe ad aver scartato erroneamente gli interrupt generati dalla macchina host.
Detto questo, non resta che definire quali sono le costatazioni che permettono di dedurre esattamente quale tecnica sia attualmente in uso dal software emulato, sebbene esse variino in base al software di virtualizzazione che si sta adoperando. Una seconda regola di deduzione potrebbe essere la seguente: *se il sistema guest non ha programmato alcun timer virtuale per generare interrupt, si può assumere con certezza che non utilizzerà il tick counting.* Purtroppo però anche i sistemi operativi tickless si trovano, per vari motivi, a programmare uno o più device per lanciare interrupt e ciò complica ulteriormente le cose. Molti software per la virtualizzazione, come ad esempio VMware [19], sfruttano anche il tipo di sistema operativo per dedurre la tecnica che utilizza; oppure è direttamente il software guest ad informare tramite hypercall la macchina virtuale che intende adoperare la tecnica tickless[7].

4.2.3 Mantenimento del Wall-Clock Time

Ogni sistema operativo, che sia su una macchina fisica o virtuale, ha comunque il compito di mantenere corretto il valore di "tempo assoluto" inteso come data e ora corrente. Come si è già accennato precedentemente, tale valore viene inizializzato durante il boot e poi aggiornato accuratamente man mano che il tempo passa.
Per quanto riguarda l'inizializzazione, la macchina virtuale opera esattamente come quella fisica: usa il dispositivo virtuale a batteria CMOS oppure sfrutta la scheda di rete virtuale per collegarsi ad un time server presente nella rete. L'operazione di mantenimento invece è più complicata, infatti risulta essere

[7]Notare che questo si verifica solo se il sistema operativo guest è pienamente consapevole di girare in un ambiente virtuale e non su una macchina fisica.

una sfida persino per le macchine fisiche. I timer device infatti tendono a subire dei drift causati o da un erroneo conteggio del counter, magari dovuto ai frequenti cambi di velocità della cpu (ad esempio con il Turbo Boost [20]), oppure originati dal variare della temperatura interna. A tutto ciò si va ad aggiungere un ulteriore problema: quando una macchina virtuale viene attivata dopo una sospensione o una pausa, il suo orologio interno rimane impostato al valore che aveva al momento precedente la sospensione. Tutto ciò può essere ovviato attivando nel programma di virtualizzazione una sincronizzazione periodica con un particolare orologio di riferimento, che sia in rete o nell'host.
Per vari motivi, l'amministratore di sistema potrebbe avere la necessità di mantenere gli orologi delle varie macchine virtuali svincolati da ogni valore esterno, di conseguenza in genere i software di virtualizzazione dispongono anche di un metodo per disabilitare tale sincronizzazione.

4.3 Timekeeping nei Sistemi Operativi

In questa sezione verranno presi in esame le tre principali macro categorie di sistemi operativi al fine di descrivere il loro approccio al mantenimento del tempo corrente, il loro uso dei timer device ed eventuali particolarità da tenere in considerazione quando questi SO vengono virtualizzati.

4.3.1 Windows

Il sistema operativo Microsoft Windows adopera la tecnica di timekeeping del tick counter, contando perciò gli interrupt generati timer device. Il tipo di dispositivo utilizzato per lanciare interrupt, però, varia in base sia alla specifica versione di Windows che al *Windows hardware abstraction layer* (HAL)[21] installato. Alcuni vecchi processori (ad un solo core) utilizzavano il PIT come device, ma successivamente sono passati al CMOS come device di riferimento per lanciare interrupt periodici. Nei sistemi che fanno uso del PIT il rate è circa di 100Hz (anche se Windows98 ha un rate di 200Hz), in-

vece nei sistemi che sfruttano il timer CMOS l'interrupt rate di base è circa 64Hz [19].
Occorre precisare che Microsoft Windows è dotato anche di una funzionalità particolare chiamata il *multimedia timer API* che, quando è attiva, può velocizzare la frequenza del clock fino a 1.024Hz (o 1.000Hz su sistemi che usano il PIT). Tale opzione entra in gioco sia per far fronte ad una maggiore necessità computazionale richiesta dalle applicazioni multimediali, sia nel caso in cui il sistema stia eseguendo un'applicazione Java. Alcuni software di virtualizzazione, come ad esempio VMware[19], esercitano questa funzionalità a proprio vantaggio per gestire il caso in cui una virtual machine necessiti di un interrupt a cadenza maggiore per recuperare il divario con il clock fisico.
Viene inoltre utilizzata una ulteriore misurazione del tempo, accessibile tramite la system call `QueryPerformanceCounter` [22]. Il nome, in verità, è fuorviante perché tale chiamata non accede direttamente ai *performace counter register* della CPU, ma legge semplicemente il contatore di uno dei timer device, permettendo così di ottenere un valore di tempo più accurato che contando gli interrupt di sistema. Anche in questo caso, la scelta del dispositivo da cui leggere il contatore dipende dalla versione del sistema operativo e dal HAL in uso, vediamo alcuni esempi:

- in alcune versioni, specialmente quelle multiprocessore, modificano il registro TSC a zero durante l'avvio della macchina, in parte anche per assicurarsi che tutti i processori siano sincronizzati; inoltre il TSC viene utilizzato anche per misurare la velocità di ogni processore, confrontando il suo valore con quello gli altri timer device[8]

- in alcune versioni, si utilizza il local APIC timer per generare un interrupt al secondo, mentre in altre tale device non viene minimamente considerato

- in alcune versioni, se il sistema è dotato di più processori, Windows inoltra l'interrupt principale del sistema a tutti i processori come un

[8] Anche questa operazione viene fatta durante l'avvio

broadcast, mentre in altre l'interrupt viene segnalato solo al processore primario, avvalendosi però di interruzioni inter-processor per schedulare l'avvicendarsi dei vari processori

Per effettuare l'inizializzazione del tempo assoluto, Microsoft Windows legge il clock a batteria autonoma CMOS TOD, il quale, occasionalmente, viene persino modificato in accordo a quelle regioni del mondo in cui esistono i concetti di "ora solare" e "ora legale". Inoltre, in alcuni modelli della famiglia Windows-NT (dalla versione 4.0 in poi), esiste un deamon che controlla con cadenza oraria il valore del tempo assoluto mantenuto dal sistema e il valore presente del clock CMOS TOD; se la differenza supera i 60 secondi, esso ristabilisce il tempo corretto sincronizzando il tempo di sistema con il dispositivo.

4.3.2 Linux

Nel corso della sua storia, Linux ha subito numerose modifiche riguardo alle tecniche utilizzate per monitorare ed aggiornare il tempo di sistema. Infatti, in alcune versioni precedenti i kernel possedevano dei bug specifici che rendevano il sistema operativo particolarmente inadatto alla virtualizzazione (ad esempio la richiesta di interrupt ad alto rate che comportavano un abbassamento delle performance e un carico di lavoro eccessivo per la macchina host, anche quando il sistema era in stato *idle*). In questa sezione elencheremo alcune delle modifiche più significative avvenute negli anni.
Il kernel Linux fino alla versione 2.4 ed anche la prima versione del kernel 2.6 utilizzava esclusivamente la tecnica tick counting, identificando nel PIT 0 la principale fonte degli interrupt. Di recente però, si sono susseguiti una serie di cambiamenti atti a riscrivere totalmente la filosofia di mantenimento del tempo di sistema, primo tra i quali c'è l'aggiunta di un livello di astrazione chiamato *clocksource*. Esso consiste, di fatto, nell'implementazione di una interfaccia di supporto per la lettura da parte del kernel dei vari timer device, permettendo così la transizione del sistema alla tecnica tickless. In generale, il kernel effettua la scelta di quale device utilizzare come predefinito, durante

il boot del sistema.
Un ulteriore modifica al kernel, completata nella versione 2.6.21 nei 32-bit e nella 2.6.24 nei 64-bit, ingloba un aggiuntivo livello di astrazione denominato *clockevents*. È stata perciò inserita una nuova opzione di configurazione del kernel identificata con `NO_HZ` la quale, se abilitata durante la compilazione del kernel, ne modifica la tecnica di timekeeping, portandolo ad utilizzare un interrupt periodico per le callback, per lo scheduling e la gestione dei processi.
Su Linux, le applicazioni a livello utente possono persino richiedere ulteriori interrupt utilizzando il dispositivo `/dev/rtc`, i quali sfruttano o il timer periodico CMOS oppure l'HPET. Questa funzionalità viene sfruttata sia dai software multimediali che da alcuni software di virtualizzazione per richiedere interrupt ad un rate maggiore di quelli offerti dalla macchina host.
Per quanto riguarda il wall-clock time, la maggior parte delle distribuzioni di Linux sono impostate per inizializzare il tempo del sistema al clock CMOS TOD durante l'avvio della macchina e sovrascriverlo durante lo spegnimento. Inoltre, in alcuni casi, il kernel di Linux viene programmato per modificare periodicamente (ad intervalli di 11 minuti) il CMOS TOD in base al time server di riferimento [23].

4.3.3 Solaris

In Solaris 10 la tecnica di timekeeping utilizzata è quella tickless.
Come abbiamo spiegato precedentemente, tale tecnica consiste nel far si che sia il sistema operativo a leggere uno dei vari contatore hardware (solitamente il TSC è quello impostato di defaut) al fine di ottenere l'ammontare grezzo di tempo passato dal boot della macchina. Il wall clock time viene letto invece dal CMOS, sempre durante il boot.
Solaris, come ulteriore accorgimento, legge periodicamente il clock CMOS TOD e sfrutta tali informazioni per correggere o rifinire le misurazioni effettuate dal TSC, modificando la sua frequenza in base all'occorrenza.
Anche il servizio di callback di Solaris non adopera alcun interrupt periodico;

piuttosto mantiene un interrupt one-shot per svegliare il sistema in tempo per la successiva callback. Da notare che questa caratteristica rende Solaris molto simile a Linux con il clocksource e la funzionalità `NO_HZ` attiva, e ciò le rende una buona accoppiata guest/host.
Questo sistema operativo, quando si trova a girare su una macchina virtuale, potrebbe portare alla luce due particolari problematiche:

1. si verifica nel caso in cui stia girando su un sistema multiprocessore; Solaris tenta sempre di sincronizzare il TSC durante il boot misurando quanto ogni processore si discosta dal clock, calcolando l'offset per ciascuno di essi ed applicando tale offset a livello software come correzione ogniqualvolta che una sottosequenza tenta di leggere il TSC. Purtroppo però, se il sistema operativo si trova su una macchina virtuale, non necessariamente i processori della macchina fisica sono simultaneamente schedulati durante il boot della VM, perciò le misurazioni effettuate potrebbero essere erronee[9].
   In realtà questo problema non si verifica solo in Solaris, ma gli altri sistemi operativi che tentano di sovrascrivere il clock TSC, lo fanno attraverso l'hardware, dando così modo al software di virtualizzazione di intercettare l'operazione e prendere opportuni provvedimenti.

2. un'altra piccola considerazione da fare è che Solaris può occasionalmente rendersi conto se il clock CMOS TOD contiene un valore estremamente diverso da quello che si aspetterebbe di trovare, concludendo così che il dispositivo sia rotto. La innocua conseguenza di ciò consiste nella visualizzazione di un messaggio di warning nella console log di Solaris.

[9] Tale problematica è stata corretta in Solaris 10, con l'update 8.

4.4 Conclusione del capitolo

In questo capitolo abbiamo studiato in maniera approfondita il concetto del tempo all'interno di un calcolatore, inteso sia come "tempo assoluto" definito con data ed ora, che come percezione fisica del tempo trascorso dal momento del boot di sistema.

Capitolo 5

Virtualbox

Virtualbox è sicuramente considerato tra i più famosi software per la creazione di macchine virtuali, come lo sono ad esempio VMware o Microsoft Virtual Server. Tuttavia, a differenza di quelli appena citati, pur essendo di proprietà della Oracle, Virtualbox dispone di una licenza *open source* (con una versione ridotta distribuita secondo i termini della GNU General Public License) e ciò ci permette di poter visionare direttamente il codice sorgente. Virtualbox è un VMM che permette l'installazione di un sistema operativo completo su una macchina fisica avente già un proprio sistema operativo, facendo sì che il software localizzato nella macchina virtuale non si renda minimamente conto di non risiedere direttamente sull'hardware.
In questo capitolo cercheremo di presentare le potenzialità di tale software di virtualizzazione, cercando però di focalizzarci nelle funzionalità che sono più di nostro interesse ai fini del progetto.

5.1 Le principali caratteristiche

La possibilità di poter analizzare direttamente il codice sorgente grazie alla licenza open source è sicuramente uno dei maggiori vantaggi che Virtualbox può vantare nei confronti dei suoi concorrenti, ma non è l'unico punto di forza che possiede. Qui di seguito elencheremo solo alcune delle più note

proprietà che lo caratterizzano. Per avere maggiori dettagli su come queste vengono realizzate o su quali impostazioni occorre modificare per poterle attivare consiglio di visitare la sezione corrispondente della documentazione ufficiale, nel manuale online [24].

Portabilità

Virtualbox può essere installato su un gran numero di sistemi operativi, sia a 32 bit che a 64 bit.
Essendo un software di virtualizzazione con *"hosted" hypervisor*, non gira direttamente sull'hardware ospitante ma su di un altro sistema operativo, ed inoltre non richiedendo alcuna collaborazione al sistema operativo guest, realizza perfettamente la tecnica di Full Virtualization (vedi Sez. 2.3.1).
È importante sottolineare che , il software di virtualizzazione che viene installato sulle diverse piattaforme ospitanti è pressoché identico nelle sue caratteristiche e ciò permette agevolmente di realizzare una "migrazione" delle macchine virtuali tra host diversi aventi Virtualbox installato. Inoltre, grazie alla compatibilità con il formato OVF (Open Virtualizatio Format [1]), permette persino di importare VM create con uno degli altri software di virtualizzazione.

Non richiede l'hardware virtualization

Per la realizzazione della virtualizzazione non necessita della presenza di un hardware sulla macchina fisica in grado di supportarlo nella gestione del guest. Ciò significa che, se la macchina host è recente e dispone delle agevolazioni dell'hardware assisted (vedi Sez. 2.5) allora Virtualbox le sfrutta, ma altrimenti può benissimo utilizzare altre tecniche di virtualizzazione orientate più all'emulazione software (Dynamic Binary Translation, vedi Sez. 2.4.1), potendo così lavorare anche su sistemi non naturalmente virtualizzabili (vedi Sez. 2.4).

[1] È uno standard aperto per la creazione e la distribuzione di applicazioni virtuali o più comunemente di software che possa essere eseguito su macchine virtuali.

Ovviamente si può anche "forzare" Virtualbox a lavorare in Dynamic Binary Translation, semplicemente disabilitando l'opzione hardware assisted. Come vedremo più avanti, nella Sez. 7.3.1, questa particolare configurazione ci permetterà di realizzare una nested virtualization tra KVM e Virtualbox.

Guest Additions

Le cosiddette *Virtualbox Guest Additions* sono pacchetti software che possono essere installati all'interno dei sistemi guest supportati al fine di migliorare le performance o al fine di fornire ulteriori funzionalità di integrazione e comunicazione con il sistema host. Grazie all'installazione di questi pacchetti sarà possibile usufruire di diversi vantaggi: un'automatica modifica della risoluzione video, l'accellerazione grafica 3D, l'opzione *seamless windows* che permette di gestire la VM col mouse come se fosse una delle tante finestre del sistema operativo ospitante, ecc...
Un esempio importante di Guest Additions consiste nella "cartella condivisa", grazie alla quale è possibile trasferire file comodamente tra host e guest.

Hardware support

Tra le molte altre cose, Virtualbox supporta:

- Guest Multiprocessing (SMP). Virtualbox può rendere possibili fino a 32 CPU virtuali per ognuna delle sue VM, indifferentemente da quanti core sono presenti nella macchina fisica ospitante.

- Supporto USB. Virtualbox implementa un controller virtuale USB che permette all'utente di connettere un numero arbitrario di dispositivi USB alla propria VM senza però dover installare alcun driver specifico sull'host.

- Compatibilità hardware. Virtualbox rende possibile la virtualizzazione di un vasto numero di dispositivi, tra i quali ci sono anche i

più tipici device tipicamente forniti dalle altre piattaforme di virtualizzazione. A tal proposito possiamo citare i controller di hard disk IDE, SCSI e SATA, moltissime schede di rete e audio, porte virtuali seriali e parallele ed inoltre il Input/Output Advanced Programmable Interrupt Controller (I/O APIC), disponibile in moltissimi sistemi odierni. Ovviamente tutto ciò contribuisce alla portabilità delle VM create.

- Supporto ACPI. L'acronimo sta per "Advanced Configuration and Power Interface" e consiste in uno standard industriale aperto che definisce interfacce comuni per il riconoscimento dell'hardware, la configurazione e la gestione energetica di scheda madre e periferiche. Grazie ad esso Virtualbox può sempre fare un report dello stato energetico del host al sistema guest, rendendo così molto più agevole, ad esempio, il testing di applicazioni per sistemi mobili che necessitano di informare i propri utenti sullo stato della batteria.

- Risoluzione multischermo. Le macchine virtuali create con Virtualbox supportano una risoluzione dello schermo anche maggiore di quella disponibile con l'hardware a disposizione.

- Supporto iSCSI built-in. Questa funzionalità unica permette di connettere direttamente la VM ad un server di storage iSCSI[2] senza passare per il sistema host.

- PXE Network boot. La scheda di rete virtuale di Virtualbox supporta completamente il boot da remoto tramite il Preboot Executing Environment.

[2]È un protocollo molto utilizzato in ambienti SAN poiché permette di consolidare l'archiviazione dei dati su dispositivi virtuali, collegati attraverso la rete, dando l'illusione di disporre localmente di un disco fisico che invece si trova in remoto

Snapshots

Virtualbox può generare e mantenere in memoria un numero arbitrario di snapshot che rappresentano lo stato della macchina virtuale. L'utente può andare a ritroso tra di essi e ripescare un qualsiasi snapshot passato, decidendo così di far cominciare una configurazione alternativa della VM da lì; ciò permetterebbe di creare un intero albero di snapshot.
Inoltre, l'uso di snapshot permette anche di spegnere e riavviare una macchina virtuale senza però dover necessariamente interrompere il sistema operativo guest. Infatti, grazie allo spegnimento in "Safe Mode", è possibile salvare lo stato in cui si trova la vm al momento della chiusura e riprenderlo poi al momento dell'attivazione. In questo modo è possibile persino "ibernare" eventuali software che sono in fase di esecuzione durante l'attivazione della procedura di spegnimento.

I Gruppi

È permesso creare gruppi ed organizzare le VM sia collettivamente che individualmente. Le tipologie di gruppi creabili vanno da quello di base, a quelli più complessi come quelli gerarchici o innestati.
In generale, le operazioni che si possono effettuare su un gruppo sono le stesse di quelle che si possono applicare ad una sola VM, ad esempio Start, Pause, Reset, ecc...

Architettura Pulita e Modulare

La chiave di progettazione per gli sviluppatori di Virtualbox è sicuramente stata la modularità.
Grazie al design modulare, esiste una chiara separazione tra il codice client e quello server; inoltre sono stati creati diversi frontend dai quali si possono controllare tutte le VM (vedi Sez. 5.2.1), da quello più intuitivo con interfaccia grafica GUI, a quello più potente, programmabile da linea di comando.
Inoltre, grazie alla struttura della sua architettura, Virtualbox può esporre

tutte le funzionalità e tutte le configurazioni possibili attraverso un semplice Software Development Kit (SDK) il quale rende possibile la programmazione di una VM attraverso alcuni dei linguaggi di programmazione più comuni (vedi Sez. 5.3).

Remote Machine Display

La funzinalità chiamata "Virtualbox Remote Desktop Extension" (VRDE) permette di effettuare l'accesso da remoto con alte prestazioni ad ogni VM attiva. Questa estensione è compatibile con il Remote Desktop Protocol (RDP), originariamente creato per Mircosoft Windows, ma con l'aggiunta del supporto per USB.
Il VRDE non sfrutta il server RDP ma si collega direttamente allo strato di virtualizzazione, col risultato che risulta perfettamente funzionante anche con sistemi operativi guest che non sono Windows e non richiede alcun supporto da parte della macchina virtualizzata.
Tra le più importanti capacità che esso comporta ci sono:

- Extensible RDP authentication. Virtualbox già supporta Winlogon su Windows e PAM su Linux per effettuare l'autenticazione RDP; in aggiunta, mette a disposizione un SDK molto usabile che permette al programmatore di creare altre interfacce che permettano altri metodi di autenticazione

- USB over RDP. Sfruttando il canale virtuale via RDP, Virtualbox permette agli utenti di connettere dei dispositivi USB localmente e collegarli alla VM che sta attualmente girando in remoto su un server Virtualbox RDP.

5.2 Funzionalità utili per il nostro scenario

Come anticipato in precedenza, Virtualbox permette un vastissimo numero di configurazioni diverse per le macchine virtuali ed è per questo che

viene considerato un software di virtualizzazione molto versatile e potente. Dato che Virtualbox descrive tutte le sue funzionalità accuratamente nella propria documentazione [24], non staremo qui a ripeterle. Piuttosto cercheremo di concentrarci solo sul descrivere alcune delle configurazioni che possono esserci utili ai fini del nostro scenario, al fine di rendere più chiare le scelte progettuali che verranno esposte in seguito.

5.2.1 I Frontend

Con frontend si intende quella parte del software di virtualizzazione con cui andrà ad interfacciarsi l'utente.
Virtualbox, come detto in precedenza, possiede diverse tipologie di frontend, ognuno dei quali permette sia di interagire con le macchine virtuali che con le configurazioni delle stesse; purtroppo però non sono tutti semplici da usare e potenti al tempo stesso. Infatti potremmo dire che esiste una correlazione inversa tra la facilità d'uso, intesa anche per un utente non esperto, e la quantità di opzioni configurabili.

GUI L'acronimo sta per Graphical User Interface e consiste appunto nel principale frontend grafico di Virtualbox. È stato creato usando la libreria Qt toolkit con il principale obiettivo di semplificare la gestione delle macchine virtuali agli utenti più inesperti.
È strutturata in modo tale da guidarlo nella creazione e nella gestione della sua macchina virtuale attraverso alcuni accorgimenti:

- le VM vengono create consigliando le configurazioni di default, utili a permettere le funzionalità standard di una macchina virtuale come, ad esempio, avere una quantità di memoria sufficiente in base alle esigenze del SO scelto, oppure avere una connessione internet che permetta agevolmente di navigare senza limitazioni;
- nella pagina di apertura dell'interfaccia è presente un elenco delle VM create con Virtualbox e, selezionando una di esse, è possibile leggerne una versione riassuntiva delle sue specifiche impostazioni;

- le operazioni di avvio e spegnimento di una macchina virtuali sono possibili sia attraverso l'utilizzo dei pulsanti, grandi e sempre visibili nell'interfaccia, che attraverso diversi altri menù disponibili;
- le configurazioni che sono modificabili con questa interfaccia sono organizzate su diversi livelli gerarchici, in modo tale da mettere in primo piano i pannelli utili alle impostazioni basilari, ed in secondo piano le operazioni più complesse. Così da non appesantire i pannelli e di non sovraccaricare di troppe informazioni gli utenti.

Essendo questo il frontend di default di Virtualbox, la quasi totalità del suo manuale utente lo prende come punto di riferimento per le configurazioni.

VBoxManage Consiste nell'interfaccia a linea di comando di Virtualbox ed è di gran lunga il frontend più potente tra quelli disponibili, in quanto permette di ottenere delle configurazioni molto più avanzate rispetto agli altri. Questa sua enorme qualità viene compensata da un alto livello di difficoltà nell'utilizzo, che non lo rende adatto per utenti che non posseggono una formazione o delle competenze informatiche.

VBoxSDL È un'interfaccia grafica semplificata che, a differenza della GUI principale, possiede intenzionalmente un limitato numero di funzionalità disponibili e di configurazioni applicabili alle VM. Come specificato anche nella documentazione ufficiale [24], questa interfaccia viene principalemente utilizzata internamente dallo staff per il debugging e perciò non è uffialmente supportata. Potrebbe però rivelarsi utile in contesti lavorativi in cui la macchina virtuale non necessariamente viene controllata dalla stessa persona che la utilizza.

VBoxHeadless Questo è l'unico frontend che non produce alcun output visibile dell'host, ma agisce come un RDP server nel caso in cui sia stata installato il Virtualbox Remote Desktop Extension (VRDE). A differenza di tutti gli altri frontend, non necessita di alcun supporto grafico e

questo risulta essere un vantaggio molto utile se si vuole mantenere una macchina virtuale su un host particolare come, ad esempio, un server Linux non dotato di un sistema X Windows. Vedremo in seguito che questo frontend ci sarà molto utile nel nostro scenario.

5.2.2 Tipologie di connessione

Virtualbox mette a disposizione dell'utente ben otto tipi di PCI Ethernet card virtuali per ogni macchina virtuale[3] e, per ognuna di esse, permette di specificare sia la tipologia dell'hardware in uso che la modalità con cui tale card si interfaccerà con la rete fisica presente nell'host.
Per quanto riguarda le tipologie di hardware disponibili, sono ovviamente state prese in considerazione le schede di rete più diffuse: AMD PCNet PCI II (Am79C970A), AMD PCNet FAST III (Am79C973, quella di default), Intel PRO/1000 MT Desktop (82540EM), Intel PRO/1000 T Server (82543GC) e Intel PRO/1000 MT Server (82545EM). Inoltre è stata aggiunta anche un'ultima opzione speciale chiamata *Paravirtualized network adapter* (o virtio-net) che, a differenza delle precedenti, non virtualizza un hardware di rete comune, affida ad uno specifico software nell'ambiente virtualizzato il compito di occuparsene, risparmiandosi così la complessità dell'emulazione della scheda di rete e migliorando le performance del sistema.
La questione che maggiormente ci interessa però, non riguarda l'aspetto hardware ma le modalità di realizzazione della rete da parte di Virtualbox. Esistono infatti ben sette configurazioni di rete differenti ed ognuna di esse presenta dei vantaggi e svantaggi che devono essere presi in considerazione se si vuole scegliere la modalità più adatta alle esigenze del proprio sistema. È proprio per ambire a tale consapevolezza che presenteremo in dettaglio le diverse tipologie di rete possibili.

Non Connesso (*Not Attached*) Grazie a questa modalità Virtualbox notifica al sistema operativo guest che pur avendo una scheda di rete,

[3]L'interfaccia GUI, in realtà, permette di modificarne solo quattro; per configurare le altre è necessario utilizzare VBoxManage

la connessione risulta assente (cosa che potrebbe accadere in caso di cavo Ethernet non inserito, o in mancanza di Wifi a cui connettersi). L'utilità di tutto ciò sta nel poter forzare il sistema operativo guest a riconfiguare la propria connessione simulando un ritorno della connessione, usando cioè questa modalità come perno per togliere e riattivare la connessione.

NAT (*Network Address Translation*) È la configurazione impostata di default durante la creazione guidata dalla GUI di una nuova macchina virtuale. È infatti quella più semplice e comoda per permettere all'utente una normale navigazione in un contesto casalingo, essendo paragonabile ad un normale pc connesso ad un router, realizzato in questo caso dal modulo di "Virtualbox networking engine"; il compito di questo modulo è di mappare il traffico da e verso la macchina virtuale in modo trasparente e lo concretizza frapponendosi tra il sistema guest e quello host, con lo scopo principale di garantire una separazione tra di essi, massimizzando così la sicurezza.

La principale conseguenza di questo alto livello di sicurezza è che, esattamente come avviene in una rete privata nascosta da un router, la macchina virtuale risulta invisibile e irraggiungibile dall'esterno, non rendendo perciò possibile utilizzare il sistema guest come server.

I frame inviati dal guest vengono appositamente trattati da Virtualbox estraendone il pacchetto TCP/IP ed usando il sistema operativo host per fare in modo che tali dati risultino spediti da un'applicazione residente sulla macchina fisica; in questo modo, chiunque comunichi con la macchina virtuale, dovrà necessariamente intercedere per l'host, credendo addirittura di comunicare solo con esso. Inoltre, sempre per mantenere la separazione tra i due sistemi, Virtualbox fornisce alla vm anche un DHCP server, in modo da potergli assegnare uno spazio di indirizzi completamente diverso rispetto a quello dell'host.

Purtroppo presenta quattro limitazioni che lo rendono inadatto per usi più elaborati rispetto alla normale navigazione (Vedi Sez.6.3.3 della

documentazione ufficiale [24]).

Rete con NAT (*NAT Network*) Consiste in una nuova versione, considerata ancora "sperimentale", della modalità NAT; è stata introdotta in Virtualbox dalla versione 4.3 con lo scopo di sopperire ad alcune limitazioni di quest'ultima.

In generale si può dire che il comportamento risulta essere lo stesso del NAT, cioè agendo da router e raggruppando i sistemi in una rete privata, isolata dall'esterno. A differenza del caso precedente, però, questa nuova tipologia permette al guest di comunicare sia con altri guest appartenenti alla stessa rete privata, che di essere raggiunto dalle macchine esterne, utilizzando i protocolli TCP e UDP sia con indirizzi IPv4 che IPv6.

Al fine di rappresentare correttamente il concetto di "rete privata", la NAT network possiede un nome e diverse impostazioni configurabili, necessarie per permettere al guest di essere on-line. È possibile specificare lo spazio di indirizzi attribuibile ai sistemi in rete, includere un server DHCP ed, inoltre, mette a disposizione una determinata funzionalità molto utile, chiamata **port forwarding**. È grazie ad essa che le entità esterne alla rete privata possono raggiungere il guest direttamente.

Ciò avviene mediante la creazione di regole che impongono di inoltrare tutto il traffico che giunge ad una specifica porta dell'host ad un solo tra i guest della sottorete, avente come ip un indirizzo menzionato nella regola.

Scheda con Bridge (*Bridged Network*) Questa è sicuramente un'opzione più avanzata rispetto alle precedenti, sia come implementazione da parte di Virtualbox, sia come percezione della rete da parte dei sistemi host e guest.

Con la modalità bridge, Virtualbox usa uno specifico device driver presente sul sistema host che filtra i dati dall'adattatore fisico della rete;

grazie a questo driver, chiamato "*net filter*", è possibile realizzare in software un'effettiva nuova interfaccia di rete. In questo modo, dal punto di vista della macchina fisica, il guest risulta fisicamente connesso ad esso attraverso un cavo di rete, e ciò renderebbe possibile ai due sistemi di comunicare agevolmente tra di loro, ma soprattutto permetterebbe di configurare l'host affinchè agisca come router per il guest.
È importante sottolineare che questa configurazione, per essere realizzata, necessita di filtrare tutto il traffico uscente con lo scopo di modificare il MAC address dei pacchetti con quello dell'adattatore dell'host[4].

Rete Interna (*Internal networking*) L'obiettivo di questa configurazione è quello di creare una tipologia di rete software comprendente solo un sottoinsieme selezionato di macchine virtuali tra quelle presenti sulla macchina fisica, che permetta di comunicare tra loro ma gli che neghi qualsiasi comunicazione con altre entità esterne, host incluso.

Scheda solo Host (*Host-only networking*) Virtualbox ha aggiunto questa modalità dalla versione 2.2 con lo scopo di instaurare un collegamento diretto tra il sistema host e quello guest.
Per realizzarla, Virtualbox crea una nuova interfaccia di rete virtuale sulla macchina fisica (simile a quella di loopback), che si va ad aggiungere a quelle attualmente presenti. È tramite questa interfaccia che le macchine virtuali configurate con questa tipologia di rete possono cominicare tra loro e con l'host. Occorre però tener presente che, essendo un'interfaccia di rete virtuale, le virtual machine non possono raggiungere il resto della rete.

Come abbiamo avuto modo di capire, Virtualbox offre davvero un variegato numero di configurazioni di rete, ognuna adatta a scopi diversi, ma che insieme comprono davvero un ampio spettro di necessità.

[4]Ciò significa che non è esattamente identico utilizzare tale modalità quando si ha a disposizione una rete wireless o cablata, semplicemente perché non tutte le schede di rete wireless supportano la modalità promiscua.

5.2.3 Advanced topics

Virtualbox utilizza un intero capitolo della documentazione ufficiale [24], il capitolo 9, per elencare quelle che vengono definite come "Impostazioni Avanzate".
Esse consistono in un vasta gamma di configurazioni ulteriormente possibili per le macchine virtuali, adatte per gli utenti con maggiore esperienza informatica. Coprono praticamente tutti gli argomenti possibili: display, periferiche, memorie,... ma anche cose come l'automatizzazione di procedure come l'avvio del web service (vedi 5.3.2), il login nel guest, l'avvio della macchina virtuale durante il boot dell'host, ecc...
Risulterebbe fuori tema approfondire tali impostazioni in questa sede, ma va comunque citata la loro esistenza in quando, tra queste, ve ne sono alcune che verranno utilizzate nel progetto ad oggetto di questa tesi (vedi Sez. 6.4.2 e poi Cap. 8).

5.3 SDK

Virtualbox è un programma di virtualizzazione che non mira solo ad una fascia di utenti base, cioè quegli utilizzatori che lo sfruttano solo per poter avere più sistemi operativi sulla stessa macchina fisica senza però rischiare di intaccare il sistema già esistente; al contrario, offre un'ampia gamma gli strumenti utili a qualsiasi tipo di utente finale, partendo dalla semplice interfaccia GUI per gli utenti più inesperti, fino ad arrivare ad un completo ed avanzato mezzo per gli sviluppatori software: il Software Development Kit (SDK).
L'SDK viene distribuita gratuitamente da Oracle, esattamente come tutte le altri parti di Virtualbox; inoltre comprende una documentazione molto dettagliata [25] e tutti i file di interfaccia necessari ed utili per poter scrivere del codice che possa interagire con l'intero apparato di Virtualbox.
Per prima cosa faremo una panoramica sulla struttura complessiva del sistema di virtualizzazione, per capire come l'SDK si interfaccia con il resto del

software, poi andremo ad elencare i diversi approcci implementativi messi a disposizione degli sviluppatori.

5.3.1 Main API

Come abbiamo già accennato all'inizio di questo capitolo, la struttura di Virtualbox è incentrata sulla modularità e su una chiara separazione in livelli del codice sorgente. Per capire meglio tale struttura, si può osservare attentamente l'immagine in Fig. 5.1, in cui si possono chiaramente vedere i vari moduli in cui è diviso il codice e la netta suddivisione tra codice backend e quello dei frontend. Innanzitutto occorre specificare la prima macro-divisione

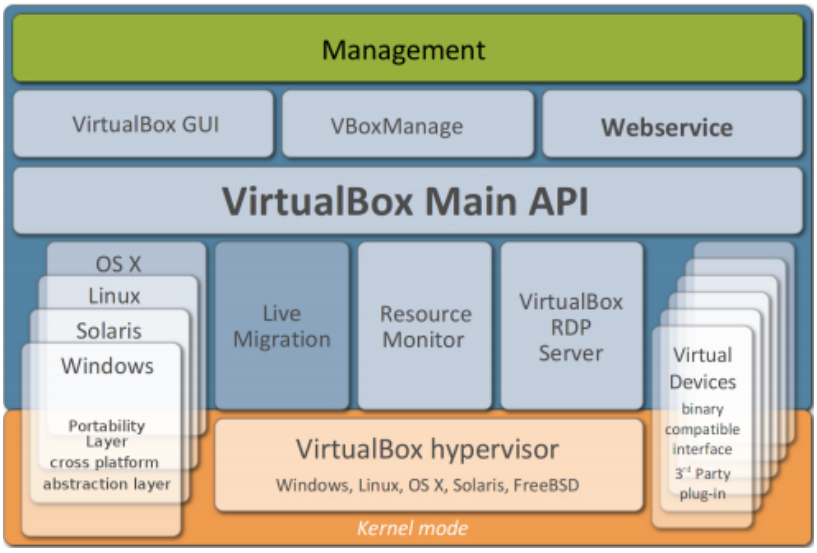

Fig. 5.1: Struttura modulare di Virtualbox [25]

dei moduli, rappresentata nella figura da due diversi colori:

- la parte in arancione rappresenta il codice che, in presenza del supporto di virtualizzazione del'hardware (Vedi Sez. 2.5), gira in kernel mode;
- la parte in blu rappresenta il codice in userspace.

Nel livello più in basso risiede ciò che Oracle chiama "Virtualbox hypervisor", che di fatto rappresenta il cuore del motore di virtualizzazione; il suo compito è quello di controllare l'esecuzione delle macchine virtuali e di controllare attentamente che non ci siano conflitti durante l'esecuzione, sia tra macchine virtuali esistenti sulla stessa macchina fisica, che rispetto all'host. Sebbene venga definito con il termine di *hypervisor*, esso comprende molteplici delle tecniche di virtualizzazione precedentemente citate (vedi Sez. 2.2); infatti Virtualbox, nella sua engine, comprende: il Virtual Machine Monitor, l'emulazione delle periferiche hardware, l'instruction-set virtualization nel caso in cui il dispositivo ospitante non possegga un hardware virtualizzabile, lo sfruttamento dell'hardware assisted virtualization, ecc...

Appena sopra all'hypervisor, troviamo alcuni moduli aggiuntivi atti a fornire funzionalità extra; tra questi citiamo il Server RDP grazie al quale è possibile usufruire del Remote Destop Protocol per connettersi all'output grafico di una macchina virtuale situata in remoto (torneremo su questo argomento più avanti, vedi Sez. 9.1.3) Ma ciò che ci interessa maggiormente osservare nella figura 5.1 dal punto di vista dell'SDK è proprio quel livello intermedio, situato appena sopra i blocchi appena citati: il Virtualbox Main API. Come possiamo intuire dal nome, le Main API costituiscono l'interfaccia principale per poter scrivere programmi che interagiscono con Virtualbox, in quanto esse permettono di esporre completamente tutte le funzionalità del motore di virtualizzazione situato sottostante.

Grazie alle Main API è possibile creare, avviare, fermare e cancellare una macchina virtuale o, persino, settarne una qualsiasi delle configurazioni disponibili. Infatti, l'enorme vantaggio di questa struttura divisa in livelli consiste soprattutto nel fatto che, internamente, persino i frontend stessi di Virtualbox operano utilizzando tali API. Da ciò possiamo evincere che, dal punto di vista di uno sviluppatore software, il codice scritto sfruttando l'SDK non solo risulterà affidabile ma potrà anche vantare una certa stabilità, in previsione di aggiornamenti futuri di Virtualbox.

5.3.2 Approcci di programmazione

L'SDK espone un elevato numero di funzioni, tutte catalogate e ben documentate [25]. Ognuna di queste funzioni può essere utilizzata in diversi modi dai programmatori in quanto Virtualbox mette a disposizione più modalità differenti di interazione col codice delle Main API.
Per poterne apprezzare le diversità ed i punti di forza, qui di seguino andremo ad analizzarle distintamente, senza però dilungarci nei dettagli implementativi (per avere ulteriori dettagli rimandiamo alla documentazione ufficiale [25]).

Web Service

I web service costituiscono un tipo particolare di interfaccia di programmazione. Infatti, mentre nel caso "standard" il programma che chiama l'interfaccia di programmazione (API) e il programma che espone tali interfacce risiedono sulla stessa macchina e, nella maggior parte dei casi, utilizzano persino lo stesso linguaggio di programmazione, con i web service si usano gli standard di Internet per comunicare, come l'HTTP e l'XML.
Per poter sfruttare con successo un web service sono necessari molti steps: il web service deve accettare la connessione, deve esserci una descrizione del servizio fornito, ed inoltre deve esserci un client connesso al servizio.
Le connessioni sono governate dal protocollo SOAP, il quale descrive il modo in cui il client ed il server debbano scambiarsi i messaggi; infine la descrizione del servizio segue lo standard di linguaggio formale in formato XML chiamato WSDL (Web Services Description Language).
Per quanto riguarda Virtualbox, i concetti appena descritti sono rappresentati da:

- L'eseguibile rappresentante il server di web service per Virtualbox: `vboxwebsrv`. Nel momento in cui tale eseguibile viene avviato, esso si comporterà come un server HTTP in ascolto su una specifica porta TCP/IP, ed i client possono connettersi ad esso.

- Il file WSDL di descrizione è incluso nell'SDK e può essere trovato nella directory `sdk/bindings/webservice`. Visto che in esso è stato descritto l'intera gamma di web service API disponibili, gli sviluppatori software possono scrivere programmi client in ogni linguaggio che disponga di un toolkit in grado di comprendere lo standard WSDL (es.Java, C++, .Net, PHP, Python, ecc...)

- Un programma client che, connettendosi col web service, voglia interagire con esso al fine di controllare le macchine virtuali di Virtualbox.

Occorre anche specificare che esistono due diversi approcci di programmazione per il client code:

1. per chi è abituato ai linguaggi orientati agli oggetti come Java o Python, l'SDK dispone di un insieme di classi, facili da usare, che permettono di utilizzare il web service appunto in modalità object-oriented. Nella documentazione si fa riferimento a questa possibilità di programmazione chiamandola *"object-oriented web service (OOWS)"*

2. in alternativa si può accedere direttamente al web service, senza il livello client object-oriented. In questo caso si parla di *"raw web service"*.

COM/XPCOM

Internamente, per questioni di portabilità e facilità di manutenzione, le Main API sono implementate utilizzando il cosiddetto Component Object Model (COM), un'interfaccia di comunicazione tra processi originariamente ideata da Microsoft nel 1993, per sistemi Mircosoft Windows.
Ovviamente, sulle macchine fisiche aventi Windows come sistema operativo ospitante, Virtualbox può utilizzare direttamente COM; viceversa però, in tutti gli altri sistema operativi diversi da Windows, Virtualbox fa uso di XPCOM, che consiste in una implementazione in versione free di COM, originariamente creata da Mozilla per i suoi browser.
Se non si ha la necessità di chiamare le procedure da remoto, oppure se si

ha già familiarità con linguaggi come Python, C++ e con COM, si potrebbe decidere di adoperare questa modalità di interazione con le Main API.
È importante fare una precisazione: i frontend di Virtualbox sono scritti in C++ ed utilizzano direttamente COM/XPCOM per richiamare le Main API, perciò, analizzando la questione in prospettiva tecnica, il web service può essere visto come un ulteriore frontend rispetto a queste COM API. Questo significa che, utilizzare COM/XPCOM per scrivere software, è di fatto il linguaggio più "naturale" per interagire con Virtualbox. Inoltre, un altro vantaggio considerevole è che COM sfrutta una chiara separazione tra l'implementazione e l'interfaccia; in questo modo i programmi esterni che ne vogliono fare uso, non hanno la necessità di sapere come le varie funzionalità sono poi implementate internamente da Virtualbox.
Visto che COM e XPCOM, pur essendo concettualmente molto simili, sono anche particolarmente diversi per quanto riguarda i dettagli implementativi, l'SDK di Virtualbox mette a disposizione dei programmatori un cosiddetto "glue layer", cioè una libreria che permetta di scrivere codice astraendo da queste differenze. Tutti coloro che utilizzeranno questo "glue layer" non solo saranno facilitati nella gestione delle funzioni, ma avranno anche il vantaggio di rendere il proprio codice portabile, in quanto completamente indipendente dalla piattaforma su cui è stato implementato. Pertanto, persino nella documentazione ufficiale dell'SDK [25], si consiglia esplicitamente di utilizzarlo.

5.3.3 Confronto

Come ogni cosa, entrambi gli approcci di programmazione hanno in sé vantaggi e svantaggi. Nella tabella 5.1, presa esattamente da [25], riportiamo una breve seppure esaustiva comparazione tra i due modus operandi.

Web Service	**COM/XPCOM**
PRO: Semplice da utilizzare con Java e Python nella versione object-oriented; compatibile perfettamente anche con ogni altro linguaggio di programmazione (C++, .NET, PHP,...)	CON: Utilizzabile solo con linguaggi in grado di dialogare con COM/XPCOM (la maggior parte dei linguaggi per piattaforme Windows, C++, Python,...)
PRO: Il client, cioè il software che si interfaccia con le Main API, può girare in remoto	CON: Il client deve necessariamente essere eseguito sullo stesso host che contiene le macchine virtuali
CON: Notevole overhead dovuto al traffico di XML nececessario per ogni singola chiamata di metodo	PRO: Overhead particolarmente basso

Tabella 5.1: Confronto Web Service e COM/XPCOM

Capitolo 6

Il laboratorio virtuale distribuito

In questo capitolo spiegheremo con maggiore dettaglio lo scenario che sta alla base del progetto, già introdotto all'inizio di questo elaborato. Poi andremo ad illustrare esattamente in cosa consiste la nostra idea di *laboratorio virtuale distribuito*, definendone le caratteristiche basilari e gli strumenti utilizzanti per la concreta realizzazione del sistema. Così facendo potremo vedere come, tutti i concetti esposti nei capitoli precedenti, andranno a collegarsi all'interno della nostra soluzione.

6.1 Scenario

Lo scenario che ci ha ispirati nella creazione del nostro progetto è quello legato alla gestione del traffico ferroviario.
È facile intuire come quello della gestione della circolazione dei treni risulti essere un problema complesso e dalle molteplici conseguenze: da un eventuale aumento della produttività grazie alla maggiore efficienza nella coordinazione tra i mezzi in circolo, fino ad importantissime implicazioni sulla sicurezza dei passeggeri nelle tratte più ad alto traffico.
Negli anni, questo sistema di coordinamento ha subito una notevole evoluzione,

grazie soprattutto all'affiancamento di nuove tecnologie telematiche ed informatiche di gestione. Infatti, fino agli anni 60-70 [26], il compito di gestire la vasta rete ferroviaria era affidato al Dirigente Centrale, una figura professionale inventata dagli statunitensi con il nome di *Despatcher*, cioè "arbitro" o "controllore". Il lavoro tipico del Dirigente Centrale era qullo di frapporsi tra i capistazione al fine di decidere, ovviamente in base a dati oggettivi, quale stazione dovesse effettuare incroci e precedenze, e cioè le operazioni base nella gestione del traffico dei treni.
Inizialmente tale lavoro di gestione era basato su pochissime attrezzature: dei telefoni collegati alle varie stazioni, un grande foglio di carta ed alcune matite. Egli infatti riceveva i dati necessari da tutte le stazioni (orari di arrivo e partenza, i minuti di anticipo o ritardo, i resoconti finali nelle stazioni di arrivo, ...) e, sulla base di queste informazioni che gli giungevano a getto continuo, il D.C. era in grado di gestire il traffico ferroviario avendo una visione completa dei movimenti dei treni nella linea a lui affidata.
Con lo sviluppo dell'elettronica, l'apparato di gestione fece un enorme passo avanti grazie al sistema C.C.L. (Controllo Circolazione Linee) e, con esso, anche la figura del Dirigente Centrale cambiò radicalmente: per seguire lo spostamento dei treni e le scelte dei capistazione nella coordinazione degli itinerari era oramai sufficiente un semplice monitor. In virtù di questo enorme cambiamento, sebbene il termine anglosassone rimase Dispatcher, in Italia il Dirigente Centrale divenne il Dirigente Centrale Operativo.
Negli anni il sistema C.C.L. venne ampliato sempre più, fino ad includere un sistema di database per tenere traccia di tutti i dati sensibili nella gestione del traffico, oltre che a all'integrazione delle tecnologie di Rete per la gestione delle comunicazioni, sia con le stazioni che lungo i vari tratti di percorrenza.

Il limite del sistema

Nonostante i numerosi passi avanti fatti negli ultimi decenni, sia dal punto di vista gestionale che dal punto di vista tecnologico, il sistema di coordinamento del traffico ferroviario ha in sé alcuni forti limiti. Primo fra tutti,

l'impossibilità di prescindere dalla figura umana del Dirigente Centrale Operativo.
Infatti, non si tratta solo un lavoro di raccolta dati che conduce a decisioni meccaniche da prendere, piuttosto entrano in gioco fattori come la competenza, l'attenzione e, spesso, anche l'ingegno. Il D.C.O. ha il compito di gestire un gran numero di treni, suddivisi in molteplici tratte diverse della stessa linea e, talvolta, necessita di dover interagire con i colleghi delle linee adiacenti per coordinare il traffico in un ottica più globale.
Per fare un semplice esempio, se si decide di bloccare un treno merci per agevolare il passaggio di un treno passeggeri che ha già subito dei ritardi, nulla vieta che altri treni non possano approfittare di questo piccolo cambiamento di programma.
Ci sono innumerevoli fattori da dover considerare: la lunghezza e la velocità dei treni, i tempi di percorrenza, ecc... Perciò si potrebbe quasi dire che la gestione del traffico viene più affidata alle capacità personali dell'operatore che a delle semplici operazioni meccaniche. Così, per adesso, il ruolo dei computer può essere considerato di "supporto", seppure non secondario, alla gestione, basato sia sull'immagazzinamento dei dati, che su alcune capacità previsionali di massima che facilitano l'operatore nel processo decisionale.
Tutto ciò ci porta al fulcro del nostro problema: il testing di eventuali modifiche al software di controllo della circolazione.
Non essendo possibile escludere il fattore umano, anche le più semplici modifiche al software di gestione necessitano di un collaudo con la figura di un qualificato Dirigente Centrale Operativo. Inoltre, a causa dell'estrema eterogeneità del sistema ferroviario, occorre specificare che il software non possiede delle caratteristiche di generalità tali da permettergli di essere testato in una sola stazione per poterne stabilire l'operatività. Ogni linea ferroviaria, infatti, ha le sue caratteristiche fisiche che la differenziano dalle altre e questo si traduce, dal punto di vista del programma di gestione, in un considerevole numero di *vincoli* di configurazione.
Perciò, l'unico modo possibile per testare fisicamente una nuova versione

del software è quello di affiancare la sua esecuzione a quella precedente, già funzionante e operativa, senza però collegarla completamente al sistema ferroviario, ma lasciando al D.O.C. il compito di costatarne la condotta in base agli input reali che raccoglie.

6.2 L'idea

Lo spunto per il nostro progetto nasce esattamente dalla costatazione del limite precedentemente descritto e dal desiderio di alleggerire sia il compito del D.O.C. in fase di testing, che di svincolare la software house dalla necessità di dover sempre svolgere le prove in locale, sulle varie tratte ferroviare, anche quando le modifiche software sono minimali.
Per prima cosa, cerchiamo di astrarre un po' il problema, in modo da inquadrarne bene la struttura, definendone i punti critici e fissando con esattezza gli obiettivi che intendiamo raggiungere.
Nel caso appena descritto abbiamo a che fare con un software molto complesso, estremamente legato a dei vincoli di configurazione che sono necessari per garantirne una corretta esecuzione, ed inoltre fortemente dipendente dal contributo umano, specialmente dal punto di vista di gestione delle situazioni di pericolo.
La nostra idea consiste nella creazione di un laboratorio virtuale distribuito che permetta di effettuare una fase preliminare di testing, eseguendo in completa autonomia il software di gestione, grazie al supporto di alcuni programmi di simulazione del traffico. Inoltre, vorremmo limitare il coinvolgimento del D.C.O solo ad un ristretto numero di casi, come ad esempio quello del verificarsi di una situazione di pericolo.
In questo modo, la software house che amministra il sistema avrebbe molta più libertà nel collaudare eventuali modifiche, oltre ad una maggiore autonomia nell'ideazione di nuove soluzioni progettuali.
Per poter esporre nel dettaglio la nostra idea, occorre procedere per gradi. Innanzitutto andremo a descrivere i vincoli principali che gravano sulla realiz-

zazione di questo laboratorio di testing, poi andremo ad inoltrarci nelle scelte procedurali da noi effettuate non solo per rendere il laboratorio conforme a tali vincoli, ma anche per conseguire gli obiettivi da noi fissati.

6.2.1 I Vincoli e gli Obiettivi di progettazione

Iniziamo definendo subito le restrizioni che lo scenario precedentemente descritto impone alla concreta realizzazione del nostro laboratorio di testing:

Vincoli di località. Necessitando di continui input di stato, e cioè delle informazioni riguardanti il traffico ferroviario, le nuove versioni del software devono necessariamente essere testate in locale, occupando così una parte delle risorse hardware disponibili;

Vincoli nelle configurazioni. Il software presente in ogni punto di controllo del D.C.O. è strettamente legato alle impostazioni locali delle macchine (indirizzo IP, mappe delle linee, ...);

Vincoli di sicurezza. Il D.C.O. ha il compito di controllare periodicamente lo stato dell'esecuzione al fine di prevenire ed, eventualmente, arginare le situazioni di pericolo.

Basandoci su tali considerazioni, abbiamo ideato una soluzione che permetta di ovviare ad ognuno di questi punti. Le contromisure da noi progettate mirano appunto a:

- utilizzare dei software che permettano di simulare efficacemente il traffico ferroviario[1];
- sfruttare le potenzialità delle macchine virtuali per poter ricreare un ambiente di testing conforme ai vincoli sulle configurazioni;
- servirsi di un servizio di Cloud per delocalizzare le operazioni di testing;
- isolare il laboratorio virtuale, ad eccezione che per eventuali interventi di sicurezza ad opera del D.O.C.

[1] In questa sede, abbiamo ipotizzato che tale programma esista già.

6.3 Dall'idea alla realizzazione

Parlando in termini più concreti, possiamo ora descrivere il nostro laboratorio virtuale di testing come un sistema distribuito, costruito su delle macchine virtuali localizzate su degli host messi a disposizione da un provider di IaaS (vedi Sez. 3.1.1). Il laboratorio deve essere strutturato in modo tale da permetterne una gestione da remoto da parte, sia dei programmatori della software house al fine di monitorare i risultati dell'esecuzione, sia da parte dell'operatore della sicurezza.

Per poter chiarire la struttura del sistema, abbiamo cercato di rappresentarlo

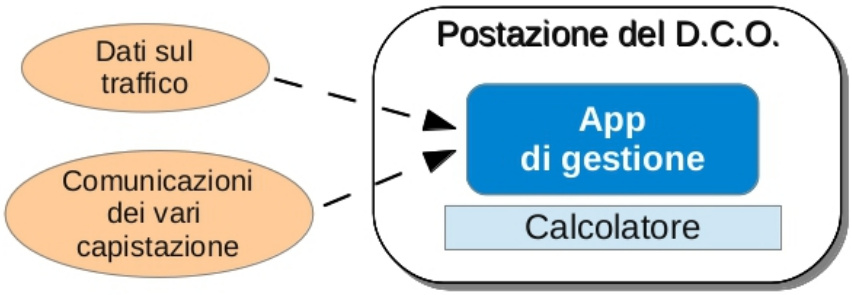

Fig. 6.1: Schema semplificato del sistema gestionale ferroviario.

in chiave sintetica nelle Fig. 6.1 e 6.2. Nella prima cerchiamo di individuare le tre entità in gioco: il flusso di dati riguardanti la situazione del traffico, le comunicazioni da parte dei capistazione ed infine il software di gestione localizzato nella postazione del D.C.O. . Come possiamo vedere, abbiamo escluso da questa rappresentazione ogni dettaglio architetturale riguardo all'applicazione, questo perché, in base ai vincoli precedentemente elencati, non abbiamo il minimo accesso all'interno del software gestionale; cosí facendo ci abituiamo da subito a considerarlo come un blocco unico e completo.

La seconda figura, invece, mostra l'aspetto schematico del nostro laboratorio virtuale di testing. Innanzitutto occorre fare una distinzione tra le entità presenti nel cloud e quelle esterne: nel cloud sono racchiusi tutti i software del sistema, sia quelli che si occupano della simulazione del traffico ferroviario

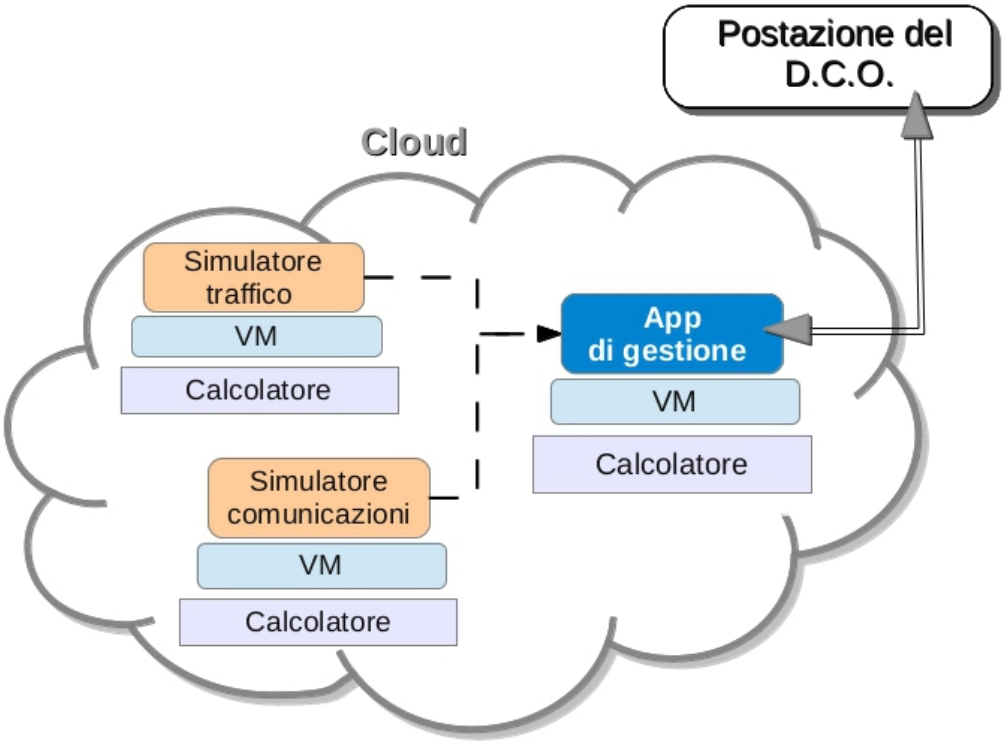

Fig. 6.2: Schema semplificato del Laboratorio Virtuale.

e delle comunicazioni, che il software principale della gestione, quello cioè che sta usufruendo del laboratorio virtuale per fare dei test di esecuzione; all'esterno del cloud, invece, si trova il D.C.O. il quale entrerà in gioco solo nel caso si verifichi un problema legato alla sicurezza durante il test.

6.3.1 I tempi di reazione

Solamente osservando la Fig. 6.2 risulta evidente la difficoltà maggiore insita nella realizzazione del nostro laboratorio virtuale di testing remoto: il problema dei tempi di reazione del D.C.O. in caso di necessità.
È ovvio pensare che esista all'interno del software un limite di tempo entro cui aspettarsi un provvedimento umano nei casi di pericolo imminente, superato il quale il programma attiverà un meccanismo automatico di blocco dei treni che impedirà il verificarsi di incidenti. Inoltre, visto che il software

di gestione è stato ideato per essere eseguito sulla stessa macchina fisica a cui si interfaccia il D.C.O., risulta ovvio supporre che tale limite di tempo sia relativamente piccolo.
Inoltre, dal punto di vista del testing dell'applicazione, si può supporre che, il superamento di tale limite con conseguente attivazione della procedura automatica di blocco, comporti il fallimento del test ed ad un invalidamento della versione del software attualmente in fase di testing. Questo ovviamente allo scopo di evitare il più possibile di giungere nella realtà ad un tale stato di pericolo.
Parlando ora nei termini del nostro progetto, l'idea di distanziare la postazione in cui viene eseguita l'applicazione gestionale da quella dell'operatore significherebbe aggiungere la latenza della rete al normale calcolo dei tempi di reazione. Purtroppo però, come abbiamo già detto in precedenza, uno dei nostri vincoli consiste proprio nel non avere modo di modificare nessuna delle configurazioni dell'applicazione.
È soprattutto per ovviare a questa problematica che entra in gioco uno dei concetti chiave della nostro progetto: la virtualizzazione. Infatti, posizionando tutti i software all'interno di delle macchine virtuali, otterremmo per ognuna di esse un'ambiente isolato e protetto, in cui l'hardware a disposizione non corrisponde a dei dispositivi fisici, ma deriva spesso da un'emulazione software ad opera del VMM (vedi Cap. 2). Sfruttando questo vantaggio, la nostra intenzione è quella di intercettare il messaggio diretto al D.C.O. e di intervenire sulla macchina virtuale rallentandone la velocità del clock. In questo modo, la concezione del tempo da parte dell'applicazione viene ingannata, così da poter garantire all'operatore il tempo sufficiente per prendere le sue decisioni.

6.4 Strumenti Utilizzati

Il nostro laboratorio virtuale di testing, per essere realizzato concretamente in tutte le sue parti, ha bisogno di due importanti strumenti di

supporto: un provider di IaaS, un software di virtualizzazione.

6.4.1 ∼Okeanos

∼Okeanos è un servizio cloud messo a disposizione dalla GRNET (Greek Research and Academic Community), una società statale operativa sotto le direttive del Ministero greco dell'educazione.
Il servizio offerto è di tipo IaaS (Infrastracture as a Service) che consiste nel mettere a disposizione una macchina virtuale, completamente personalizzabile nella scelta del sistema operativo e nelle caratteristiche hardware (numero di CPU, memoria,...), svincolando però gli utenti da tutti quei compiti quali la manutenzione o la configurazione fisica della macchina.
Grazie ad una convenzione tra l'università di Bologna e la GRNET, ∼Okeanos mette a disposizione degli studenti che ne volessero far uso, un assaggio del proprio servizio per un tempo limitato, allo scopo di agevolare le loro attività di ricerca e studio.
Analizzeremo meglio le caratteristiche di ∼Okeanos più avanti.

6.4.2 Virtualbox

Come abbiamo avuto modo di vedere (vedi Cap. 5), Virtualbox è un software di virtualizzazione molto potente e ben strutturato. Mette a disposizione dei propri utenti innumerevoli funzionalità e configurazioni, permettendo loro di personalizzare ampiamente il sistema al fine di renderlo adatto agli scopi più disparati, dalla semplice macchina virtuale casalinga, ad un complesso servizio web.
Inoltre, Virtualbox possiede due importanti caratteristiche che lo rendono molto appetibile a livello informatico: è opensource, cioè permette agli utenti di accedere al codice sorgente con lo scopo di studiare più nel dettaglio l'implementazione delle funzionalità, ed è completo di un SDK, agevolando così gli sviluppatori ad interagire col software di virtualizzazione in modo algoritmico al fine di automatizzare alcune operazioni (che siano di configu-

razione delle VM o semplicemente di attivazione o spegnimento delle stesse). Oltre a quanto appena detto, esiste un'altra motivazione che ci ha ulteriormente spinto verso l'utilizzo di questo specifico software di virtualizzazione: la funzionalità che permette la modifica della velocità del clock. Esiste infatti tra le configurazioni avanzate (Sez. 5.2.3), un'impostazione grazie alla quale si può modificare la percezione della velocità del clock da parte della macchina virtuale, consentendoci così di ovviare al problema, precedentemente esposto (Sez. 6.3.1), dei tempi rezione dell'operatore nei casi di pericolo.
Anche su questo argomento entreremo più nel dettaglio nei capitoli successivi.

6.5 Conclusione del capitolo

Da quanto abbiamo detto fino ad ora, è facile intuire la complessità e la vastità dei fattori in gioco. Così, per realizzare concretamente la nostra idea abbiamo preferito suddividere lavoro complessivo in sottomoduli, al fine di isolare uno alla volta i diversi requisiti del sistema e gestirli separatamente. Durante la realizzazione del laboratorio ci siamo scontrati con diverse problematiche inaspettate, e molto particolari perciò, per non fare confusione, sfrutteremo questa suddivisione anche nei capitoli successivi, al fine di affrontare un argomento per volta.

Capitolo 7

Il laboratorio nel Cloud

Uno degli obiettivi fondamentali nella la realizzazione del progetto è quello di poter delocalizzare il testing dell'applicazione di gestione del traffico ferroviario. A tale scopo abbiamo scelto di usufruire del servizio di cloud IaaS di ~Okeanos fornito dalla società GRNET.
Sfruttando questo servizio di IaaS è possibile delegare la parte tecnica e pratica della gestione dei calcolatori del laboratorio al provider del cloud, in questo caso ~Okeanos, ed avere a disposizione delle macchine in remoto, perfettamente customizzabili in base alle proprie esigenze, con la sicurezza che siano sempre attive e connesse alla rete.
Fortunatamente esiste anche una versione internazionale di ~Okeanos chiamta ~Okeanos global, che rende disponibili i propri servigi non solo alla comunità accademica e di ricerca della Grecia, ma anche ad altri atenei di studio internazionali, tra cui la nostra università di Bologna. Perciò, grazie all'accordo tra queste due istituzioni, ci è possibile usufruire dei servizi di IaaS semplicemente effettuando il login con l'account Unibo.

7.1 La virtualizzazione su Okeanos

~Okeanos offre ai suoi utenti accesso a macchine virtuali, reti ethernet virtuali, dischi virtuali e firewall virtuali attraverso una semplice ed intuitiva

interfaccia utente web-based. Il servizio offerto si compone di innumerevoli tasselli:

- Computer/Network Service (condificato come *cyclades*)
- File Storage Service (condificato come *pithos+*)
- Identity Management (condificato come *astakos*)
- Image Registry (condificato come *plankton*)
- Billing Service (condificato come *aquarium*)
- Volume Storage Service (condificato come *archipelago*)

i quali sono combinati allo scopo di fornire agli utenti finali un servizio completo, oltre che numerose attività di supporto (monitoraggio, gestione delle problematiche, operazioni di helpdesk,...).
Sebbene sia nato identificando il proprio target di clientela quella di utenti con ridotte competenze informatiche, di fatto risulta essere particolarmente utile e funzionale anche per gli utenti più avanzati.

7.1.1 Cyclades

Cyclades costituisce la parte di ∼Okeanos legata alla computazione e alla gestione della rete, perciò è esattamente la tipologia di servizio a cui miriamo per il nostro progetto.
La sua struttura interna comprende l'utilizzo di diverse componenti software, sia sviluppate dalla stessa GRNET che fornite da altre società note nel mercato globale. Infatti il suo design combina l'uso di Google Ganeti[1] nel backend, con delle implementazioni in Python/Django delle API nel frontend.
Per i dettagli tecnici sulla struttura e sull'implementazione delle funzionalità rimandiamo ai papers ufficiali di ∼Okeanos (vedi [27]), ma per avere un idea generale di come è strutturata l'architettura del sistema, riportiamo qui la

[1] Ganeti è un tool per il cluster management sviluppato appunto da Google

sua rappresentazione grafica (Fig. 7.1).

In particolare, quello su cui vogliamo assolutamente porre l'attenzione,

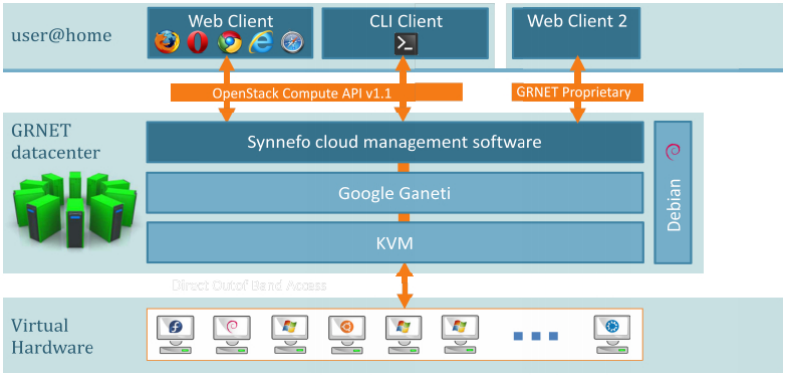

Fig. 7.1: Architettura di ~Okeanos [27].

riguarda il programma utilizzato per la creazione e la gestione delle macchine virtuali, e ciè KVM. Vedremo infatti più avanti come questa caratteristica influenzerà il nostro laboratorio virtuale di testing.

7.2 Le Nostre VM

Essendo studenti di una delle università convenzionate con ~Okeanos, abbiamo diritto ad usufruire di una versione limitata dei suoi servizi. Perciò, in prospettiva del nostro laboratorio virtuale di testing abbiamo istanziato:

- due macchine virtuali,
- due indirizzi IPv4 pubblici.

Le macchine virtuali da noi create sono dotate di un sistema operativo Debian "Wheezy" a 64bit, 2 CPU, 2048 MB di RAM e 20 GB di disco fisso. Poi, ad ognuna di esse è stato assegnato un indirizzo IPv4 pubblico.
Si è deciso di optare per un sistema operativo come Debian per diverse ragioni:

1. si tratta di uno dei sistemi basati su software libero più affidabili e completi oggi in circolazione;

2. si ha un maggiore livello di controllo delle funzionalità e delle impostazioni della macchina grazie ai permessi di root;

3. le macchine virtuali di ~Okeanos basate su kernel linux sono automaticamente dotate di un server SSH;

4. essendo una distribuzione linux dotata di kernel 3.2, adopera il meccanismo di timekeeping "Tickless" (vedere Sez. 4.2.2 e 4.3.2 per maggiori dettagli) che risulta essere più adatto ai nostri scopi.

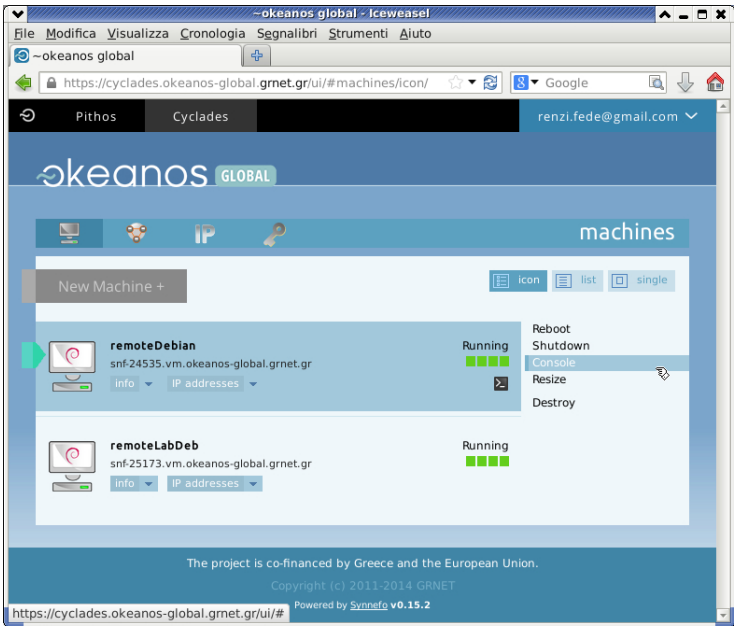

Fig. 7.2: Frontend web di ~Okeanos.

Dal punto di generale, non sono state necessarie altre configurazioni per rendere operative le macchine ai fini del nostro laboratorio virtuale, ad eccezione per alcune configurazioni di rete che analizzeremo piú avanti (Sez. 9.1.2). Terminata la procedura di creazione delle macchine virtuali possiamo ora vedere, usufruendo dell'interfaccia web, le nostre due macchine virtuali (Fig. 7.2). Osservando più attentamente la figura possiamo notare:

- in grassetto il nome scelto per le macchine virtuali, mentre sotto viene riportato il nome dell'host (es. snf-XXXXX.vm.okeanos-global.grnet.gr)

utile per collegarsi alla vm utilizzando il protocollo SSH (le credenziali per la connessione vengono fornite in sede di creazione della vm);

- subito sotto troviamo due menù a tendina utili sia per riassumere visivamente le caratteristiche fisiche e le configurazioni di rete delle vm, che per visualizzare le statistiche riguardanti la percentuale di utilizzo della rete e delle risorse computazionali;

- a destra troviamo invece il menù operativo che permette di effettuare le operazioni principali come avvio, spegnimento, ...

Tra le operazioni principali appena citate, vogliamo porre l'attenzione sull'opzione "Console", che nella Fig. 7.2 vediamo evidenziata. Effettuando il click su tale opzione verrà aperta una nuova finestra pop-up contenente la richiesta di avviare un'applicazione Java, avviata la quale, sarà possibile vedere l'output video della propria macchina virtuale di ∼Okeanos.
È però doveroso specificare che tale funzionalità, sebbene incredibilmente utile, porta con sé dei considerevoli tempi di risposta, oltre che la perdita di alcuni utili shortcouts come Ctrl-C, Ctrl-V,...

7.3 La Nested Virtualization

Una volta avviate le macchine virtuali di ∼Okeanos, la prima cosa da fare è stata installare in entrambe Virtualbox.
Prima di passare a descrivere i dettagli tecnici, è doveroso fare un'importante precisazione.

NOTA: vista la complessità della struttura che andremo a descrivere e vista l'estrema ridondanza del termine *Virtualizzazione*, allo scopo di evitare fraintendimenti o confusione nel lettore, da qui in avanti ci riferiremo alle macchine virtuali create con ∼Okeanos con il termine di HOST, mentre alle macchine virtuali nested create con Virtualbox con il ter-

mine di GUEST.

Per effettuare l'installazione del software di virtualizzazione è stato sufficiente collegarsi singolarmente in ssh agli host, utilizzando le credenziali forniteci precedentemente, e poi seguire passo passo la procedura di installazione riportata nel sito ufficiale di Virtualbox [28].
Dopodiché, sfruttando le potenzialità del frontend a linea di comando di Virtualbox, cioè VBoxManage, abbiamo eseguito l'iter necessario a creare completamente una macchina guest che si suppone ospiterà il software di gestione della rete ferroviaria. Provvederemo a descrivere dettagliatamente tale procedura nei capitoli successivi (Sez. 9.1.1), mentre qui ci limiteremo a dire che tipo di sistema operativo si è deciso di installare.
Vista la natura sperimentale di questa tesi, si è deciso di non sovraccaricare da subito il laboratorio virtuale con dei sistemi operativi complessi e pesanti. Piuttosto si è giunti alla decisione di utilizzare una soluzione snella e completa, che ci permettesse agevolmente di effettuare il collaudo del nostro sistema, senza però mettere troppa carne al fuoco. Così abbiamo scelto di utilizzare TinyCore [29].
Il "Core Project" è un sistema basato su un'alta modularità e affiancato da una attiva comunità di sviluppatori di estensioni compatibili. Si tratta di un sistema operativo particolarmente leggero, composto solamente da un kernel linux e da alcuni fondamentali pacchetti di utility; perciò è quanto di più minimale possa esiste per avere a disposizione un sistema operativo avente un ambiente desktop e l'accesso alla rete cablata.
Osservando ora la figura 7.3 possiamo capire in che stato ci troviamo nella prospettiva di realizzazione del nostro laboratorio virtuale di testing. Attualmente abbiamo semplicemente a disposizione, all'interno del cloud di ∼Okeanos, due macchine host (RemoteDebian e RemoteLabDebian), entrambe con un indirizzo IPv4 pubblico e con Virtualbox installato; inoltre, struttando quest'ultimo software, abbiamo realizzato due macchine virtuali nested denominate rispettivamente "Tiny" e "Tiny2".

Per poter rendere più semplice la comprensione di come, le varie configu-

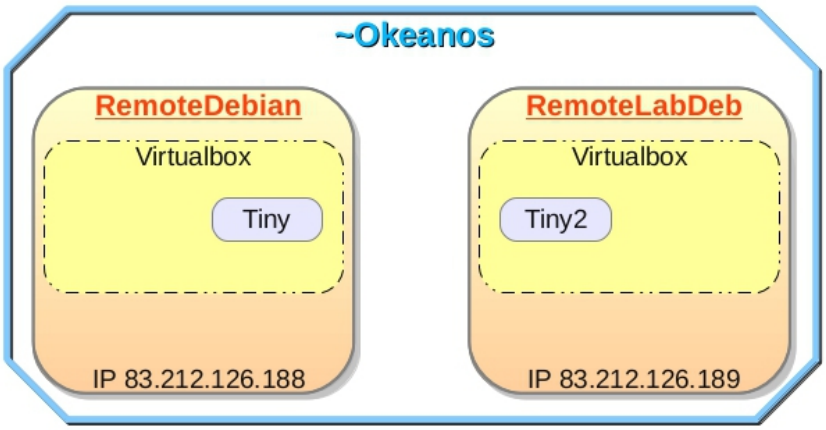

Fig. 7.3: Prima fase di predisposizione del Laboratorio Virtuale di Testing

razioni che andremo a descrivere nei capitoli successivi, contribuiranno alla realizzazione del nostro progetto, cercheremo di mantenere come riferimento schematico quello utilizzato nella Fig. 7.3, aggiungendogli via via ulteriori dettagli.

7.3.1 Il Dubbio: la compatibilità KVM e Virtualbox

Come abbiamo visto (Sez. 3.2.2), questo particolare software di virtualizzazione mette anche a disposizione le funzionalità necessarie a realizzare la Nested Virtualization in quanto riesce persino a virtualizzare il supporto di hardware assisted.
Partendo da questo presupposto abbiamo inizialmente tentato di installare una versione del sistema operativo Tiny Core a 64 bit, impostando Virtualbox in modo tale che potesse usufruire dei vantaggi della virtualizzazione hardware.
Durante la creazione della macchina virtuale nested, il software di Virtualbox non ha sollevato alcun problema: l'hardware assisted risultava presente sulla macchina host.
Purtroppo però, l'avvio del guest non ha confermato le nostre aspettative,

infatti la vm Tiny non riusciva nemmeno ad iniziare la procedura di boot; la vm si bloccava nella schermata iniziale di avvio, quella che mostra il logo di Virtualbox.
Inizialmente abbiamo supposto che il fulcro del problema fosse altrove, nelle impostazioni di Virtualbox, nella macchina di ∼Okeanos,... ma dopo aver vagliato ogni ipotesi possibile, abbiamo deciso di ritentare la creazione della vm Tiny. Per prima cosa installando un sistema a 32 bit, al posto di uno a 64 bit, mantenendo comunque attivo l'opzione hardware assisted; purtroppo anche in questo caso il sistema guest non si avvivava. Infine abbiamo optato per disabilitare persino l'utilizzo delle funzionalità di hardware assisted[2]. Con questa configurazione, la macchina virtuale nested si è avviata immediatamente, senza dare alcun tipo di problemi.

L'incompatibilità tra KVM e Virtualbox nella realizzazione della virtualizzazione annidata con hardware assisted si è rivelato per noi un problema del tutto inaspettato. Infatti non risulta sia stata ancora ufficialmente riscontrata e documentata in alcun articolo scientifico.

7.4 Conclusione del capitolo

In questo capitolo, inizialmente abbiamo approfondito le funzionalità offerte da ∼Okeanos, e poi abbiamo effettuato un primo passo nella realizzazione del nostro laboratorio virtuale distribuito.
Inoltre, ci siamo scontrati con un problema del tutto imprevisto riguardante la virtualizzazione annidata, scomprendo e dimostrando l'inconcompatibilità tra Virtualbox e KVM per quanto riguarda la virtualizzazione dell'hardware assistent.Infatti, l'unico modo per eseguire una vm di Virtualbox all'interno di una macchina di KVM è quello di disabilitare l'opzione di hardware assisted.

[2]Ricordiamo che, così facendo, si forza il software di virtualizzazione ad adoperare la tecnica del Dynamic Binary Translation, spiegata nelle Sez. 2.4 e 3.2.1

Capitolo 8

Modifica del clock

Qui parleremo più in dettaglio della procedura di modifica della velocità del clock sulla macchina virtuale creata con Virtualbox.
Ricordiamo che questa operazione risulta essere necessaria per permettere al Direttore Centrale Operativo di interagire con il nostro laboratorio virtuale nello stesso modo in cui farebbe se il test dell'applicazione di gestione dei treni fosse effettuato in locale (vedi Sez. 6.3.1).
Prima di tutto andremo a capire bene come funziona la gestione del clock in Virtualbox, e poi andremo ad affrontare i possibili approcci implementativi adottabili per conseguire il nostro scopo.

8.1 Il Clock in Virtualbox

Virtualbox, come ogni altro software di virtualizzazione, prevede nella propria engine l'emulazione e la gestione dei timer device. Inoltre, offre tutti gli strumenti necessari per permettere ai propri utenti un'accurata gestione del tempo, sia attraverso l'utilizzo di funzionalità atte a mantenere le virtual machine sincronizzate temporalmente con l'host, che attraverso l'utilizzo dei settaggi avanzati, per mezzo dei quali si rende possibile un'ulteriore livello di controllo dell'hardware virtualizzato.

In Virtualbox, per la gestione del tempo, esistono 4 tipi di orologi differenti, le cui descrizioni sono state prese direttamente dal codice sorgente, più precisamente nel file `/src/VBox/VMM/VMMR3/TM.cpp`

8.1.1 Tipologie di Clock

I clock attualmente presenti in Virtualbox sono:

1. il *Virtual* (guest);
2. il *Synchronous virtual* (guest);
3. il *CPU Tick* o TSC (guest)[1];
4. il *Real* (host).

Tra questi, i primi tre, e specialmente il secondo, risultano quelli più importanti e di maggiore interesse per questo elaborato.

Synchronous Virtual Clock

Il Synchronous Virtual Clock è legato al Virtual Clock, ma a differenza di quest'ultimo, esso tiene conto dell'eventuale ritardo causato dallo scheduling dell'host. Infatti è concepito in modo tale da mantenere tale ritardo sotto una certa soglia massima, raggiunta la quale, aumenta autonomamente la propria velocità al fine di raggiungere il valore del Virtual clock (vedi Sez. 4.2.1).
Tutti i vari device che implementano l'accesso alle informazioni temporali che poi vengono utilizzate dal sistema guest, si basano su questo clock. Ciò permette di mantenere una certa consistenza tra le varie time source.

Virtual Clock

Il Virtual Clock è implementato come un offset rispetto ad un wall-clock monotono ed ad alta risoluzione. Il wall-clock consiste in un valore reale

[1]L'uso del TSC, attualmente, è rivolto essenzialmente all'emulazione dell'istruzione *rtsc*, il cui risultato é calcolato in funzione del clock virtuale.

e assoluto del tempo specificato in un formato umanamente leggibile (cioè suddiviso in giorno, mese, anno, ora,...).
Per mantenerlo aggiornato, Virtualbox si affinda alla funzione `RTTimeNanoTS`, che restituisce il current timestamp in nanosecondi. Grazie alle sua implementazione risulta essere un orologio piuttosto preciso e compatibile con tutti gli host.
Infine, occorre puntualizzare che il valore di questo clock è legato all'esecuzione della virtual machine: se la virtual machine non è in stato di running, anch'esso si ferma.

CPU tick

Il Time Stamp Counter viene implementato in funzione del Synchronous Clock, la cui frequenza corrisponde a quella della CPU dell'host, misurata da Virtualbox durante l'avvio della macchina virtuale. Ma è anche possibile configurare la velocità della frequenza.

Per avere ulteriori dettagli su come sono stati implementati questi clock, consigliamo di fare riferimento alla tesi di Cattani [30].

8.1.2 Configurazioni Avanzate per il Clock

Tra la documentazione delle configurazioni avanzate di Virtualbox, quelle effettuabili perlopiù utilizzato il frontend a linea di comando VBoxManage, troviamo anche alcune sezioni riguardanti il clock della macchina virtuale.
Come comportamento di default, Virtualbox mantiene sincronizzati tutti i valori temporali a quelli dell'host. Infatti, durante il boot, misura la frequenza dell'host e memorizza il valore corrente di "wall clock", in modo da mantenere la percezione del tempo da parte del guest il più possibile uniforme a quella dell'host. Questo comportamento deriva dal fatto che, nella maggior parte dei casi, si vuole mantenere sincronizzata la macchina virtuale con l'ora reale.
In alcune circostanze però, potrebbe essere utile fare in modo che il time

stamp counter (TSC) rifletta la quantità di tempo effettivamente trascorsa all'interno della macchina virtuale. Per ottenere ciò occorre attivare la modalità TSC handling, specificando la macchina virtuale a cui applicare tale modifica[2], basta digitare:

```
VBoxManage setextradata "VM"
          "VBoxInternal/TM/TSCTiedToExecution" 1
```

Un'altra configurazione possibile, più esattamente, quella maggiormente di interesse per il nostro contesto, è sicuramente la seguente:

```
VBoxManage setextradata "VM" "VBoxInternal/TM/WarpDrivePercentage" 50
```

Tale opzione infatti permette di definire la velocità di frequenza del Virtual Clock nella macchina virtuale specificata, partendo dal presupposto che, essendo una percentuale, il valore di default è 100.
In questo caso, in base all'esempio sopra riportato, la virtual machine di nome "VM" subirà un rallentamento di frequenza pari al 50%, con un conseguente cambiamento della sua percezione dello scorrere del tempo. Cambiando invece il valore fornito come paramentro con 200 si otterrebbe una frequenza del clock raddoppiata rispetto a quella reale.

Sincronizzazione

È importante far notare che cambiare il rate del Virtual Clock può confondere il guest e causare comportamenti anomali. Ad esempio, nel caso in cui le Virtualbox Guest Addition fossero installate e attive, i vari meccanismi di sincronizzazione tenterebbero comunque di ristabilire la corrispondenza tra il clock del guest e quello dell'host, anche in presenza di configurazioni alternative riguardo alla frequenza del clock; per questo motivo, nel caso in cui si volesse far uso di tali configurazioni, sarebbe bene disabilitare la sincronizzazione.
Per fare ciò è sufficiente usare la seguente configurazione:

[2]La documentazione ufficiale [24] consiglia di utilizzare questa opzione solo in combinazione con l'hardware virtualization

```
VBoxManage setextradata "VM"
           "VBoxInternal/Devices/VMMDev/0/Config/GetHostTimeDisabled" 1
```

8.2 Approcci di implementazione

Il nostro obiettivo è quello di rallentare le macchine virtuali di Virtualbox presenti nel nostro laboratorio al fine di sopperire alla latenza della rete, fornendo così al Direttore Centrale Operativo il tempo necessario per interagire opportunamente col software. Tale operazione, in base al nostro scenario di riferimento, deve essere attivata solo quanto viene effettivamente coinvolto nella computazione il D.C.O., e cioè ad esempio, quando si verificano delle situazioni di pericolo.
Come abbiamo visto nella sezione precedente, Virtualbox mette a disposizione una particolare opzione che ci permetterebbe di rallentare la frequenza del clock, con conseguente impatto sulla percezione del tempo da parte del guest. Purtroppo però, allo stato attuale di Virtualbox (Versione 4.3.14), non è possibile attivare tale modifica del clock mantendendo la virtual machine in stato di running.
Per risolvere questo problema, possiamo pensare a due soluzioni:

- Modificare il codice sorgente di Virtualbox per rendere possibile questa modifica anche quando la vm è attiva;
- Automatizzare la procedura di spegnimento e riaccensione della vm in modo che si attivi quando necessario.

8.2.1 Modifica dinamica

Questo approccio implementativo è stato realizzato da Cattani nel suo progetto di tesi [30]. Viene defininita modifica “dinamica” del clock in quanto si intende attivare il cambiamento di frequenza anche quando questa è in stato di running.
Per capire meglio quanto stiamo dicendo, occorre fare prima delle precisazioni.

Ogni creazione di macchina virtuale con Virtualbox comporta, di fatto, la creazione sulla macchina host di una specifica cartella, contenente alcuni file di supporto necessari alla corretta configurazione e al regolare avvio della vm. Tra questi file, quello più rilevante per il nostro contesto, è il file di configurazione xml in quanto contiene, insieme ad altri valori specifici, anche i parametri settati grazie al comando di VBoxManage `setextradata`.
Tali valori vegnono letti da Virtualbox solo in due casi:

1. quando viene avviata una macchina virtuale

2. quando viene riattivata una macchina virtuale, in seguito allo spegnimento in "Safe mode" (vedi Sez. 5.1)

In entrambi i casi, Virtualbox legge prima di tutto i valori del file xml, poi li assegna alle specifiche variabili di esecuzione, ed infine avvia la macchina virtuale.
Sebbene sia comunque lecito effettuare una modifica utilizzando il comando `setextradata` anche fintantochè la specifica vm è attiva, tale modifica avrà effetto solo sul file xml, il quale verrà ignorato fino alla successiva attivazione della macchina virtuale. È per questo motivo che la variazione del clock utilizzando il metodo descritto nella sezione precedente, può essere definito "statico".
Col suo progetto [30], Cattani è riuscito concretamente a creare una procedura di configurazione del clock a runtime, non attraverso l'utilizzo degli strumenti di sviluppo software forniti da Virtualbox, ma realizzando una modifica diretta del codice sorgente. Ciò rende, tale tipo di soluzione, decisamente poco robusta nei confonti di eventuali aggiornamenti nel software di Virtualbox; infatti per poter mantenere aggiornato il sistema alle nuove versioni, occorrerebbe dover ripristinare e reimplementare la procedura di modifica dinamica ad ogni nuova release di Virtualbox.

8.2.2 Utilizzo dell'SDK di Virtualbox

La soluzione che proponiamo in questo elaborato, invece, si basa sullo sfruttamento del Sofware Development Kit fornito direttamente da Virtualbox, grazie al quale è stato possibile realizzare un programma che permettesse di rendere l'intera procedura di modifica del clock automatica.
Per prima cosa abbiamo dovuto scegliere l'approccio implementativo da utilizzare tra quelli disponibili (vedi Sez. 5.3.2). Osservando bene i requisiti alla base del progetto, possiamo dire che nel nostro caso:

1. abbiamo la necessità di terminare la procedura nel modo più veloce possibile, sia per non intaccare la validità del test che per non cambiare la quantità di tempo assegnata al D.C.O. per prendere provvendimenti;

2. la procedura deve attivarsi in base ad alcune condizioni all'interno della macchina virtuale di Virtualbox (torneremo su questo argomento più avanti, nella Sez. 9.1.2), perciò il programma deve necessariamente essere già presente su ∼Okeanos al momento dell'attivazione.

Così siamo giunti alla conclusione di utilizzare le librerie COM/XPCOM[3], usando però il glue layer per Java fornita da Virtualbox, in quanto permette di astrarre il codice dalla libreria, permettendoci di concentrarci solo sulle funzionalità del software di virtualizzazione.
In breve, il nostro programma effettua le seguenti operazioni:

1. chiudere in "Safe Mode" la macchina virtuale di Virtualbox (Fig. 8.1)

```
249    System.out.println("Stopping VM");
250    m.lockMachine(session, LockType.Shared);
251    IProgress p = session.getConsole().saveState();
252    progressBar(mgr, p, 150000);
253    session.unlockMachine();
```

Fig. 8.1: Codice per chiudere la vm.

[3]Nel nostro caso, avendo installato Virtualbox su una macchina avente come sistema operativo Debian (vedi 7.2) e non Windows, non possiamo disporre COM ma dobbiamo necessariamente usare XPCOM

2. modificare la configurazione della macchina rallentando la frequenza del clock (Fig. 8.2)

```
256   m.lockMachine(session, LockType.Shared);
257   String s = "VBoxInternal/TM/WarpDrivePercentage";
258   String v = "50";
259   m.setExtraData(s, v);
260   session.getMachine().saveSettings();
261   session.unlockMachine();
```

Fig. 8.2: Codice per modificare il clock.

3. riavviare la vm (Fig. 8.3)

```
264   System.out.println("Re-start VM");
265   IProgress progress = m.launchVMProcess(session, "headless", null);
266   progressBar(mgr, progress, 10000);
267   session.unlockMachine();
268   System.out.println("VM running...");
```

Fig. 8.3: Codice per riavviare la vm[4]

Una volta scritto il codice, abbiamo trasferito l'eseguibile del nostro programma su ∼Okeanos e poi ne abbiamo testato l'avvio. Le operazioni precedentemente elencate vengono brillantemente eseguite in tempi ragionevolmente brevi.

Questa tipologia di soluzione risulta essere sicuramente più robusta della precedente, in quanto si basa sulla strumentazione che Virtualbox stesso fornisce ai suoi utenti, vantando così una maggiore stabilità nei confronti di eventuali aggiornamenti futuri del software di virtualizzazione. Inoltre, il programma può essere ulteriormente perfezionato aggiungendo funzionalità di log, controlli sulle eventuali eccezioni e, soprattutto, strumenti di supporto per il debugging finale del nostro laboratorio virtuale di testing.

8.3 Conclusione del capitolo

In questo capitolo abbiamo preso in esame il problema, descritto precedentemente, dei tempi di reazione del D.C.O. (vedi 6.3.1) proponendo come

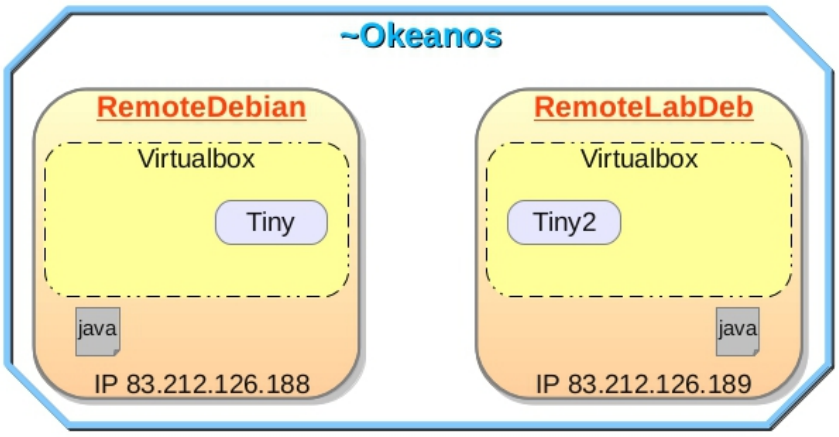

Fig. 8.4: Upload dell'eseguibile in java nelle macchine del laboratorio

soluzione l'utilizzo di una delle configurazioni avanzate presente nella documentazione di Virtualbox: la modifica della frequenza del clock virtualizzato. In questo modo, calcolando opportunamente i tempi di latenza, è possibile far in modo che l'operatore, per reagire, abbia comunque lo stesso lasso tempo che avrebbe se l'applicazione girasse in locale.
L'approccio proposto in questo elaborato consiste nell'utilizzo dell'SDK di Virtualbox in collaborazione con le librerie COM/XPCOM. In base hai test effettuati, questa soluzione risulta essere efficiente e robusta, e perciò utilizzabile nel nostro scenario.

Capitolo 9

Le Configurazioni di Virtualbox

Fino ad ora abbiamo illustrato in modo astratto la struttura del nostro laboratorio virtuale di testing, elencandone le componenti e spiegando il ruolo fondamentale di ciascuna di queste parti all'interno del sistema. Ora, invece, andremo a descrivere in modo dettagliato quelle che sono le impostazioni con cui configurare Virtualbox al fine di rendere concretamente operativo il tutto.

9.1 Gestione da remoto

Nella Sez. 7.2 abbiamo visto che ∼Okeanos mette a disposizione dei propri utenti, persino un'interfaccia web che permette di utilizzare da remoto la propria macchina, visualizzandone l'output grafico tramite browser. Questo vuol dire che si potrebbe anche sfruttare tale interfaccia per configurare Virtualbox tramite il frontend GUI, tenendo però conto degli elevati tempi di risposta.
Tuttavia, per rendere la procedura più portabile e di conseguenza meno dipendente dai servizi forniti dal provider del cloud, in questo caso ∼Okeanos, noi abbiamo preferito descrivere l'iter di configurazione in modo più tecnico: sfruttando perciò VBoxManage, il frontend a linea di comando di Virtualbox, e collegandoci alle macchine del cloud tramite la connessione in SSH.

9.1.1 Realizzare una VM

Una volta installato il software di Virtualbox seguendo la procedura indicata nel sito ufficiale [28], non ci resta che creare una macchina virtuale. In questa sezione andremo a descrivere, passo dopo passo, i comandi da sottoporre a VBoxManage per ottenere una macchina virtuale completa e funzionante.

Creare una nuova macchina virtuale

```
VBoxManage createvm --name "Tiny" --ostype Linux26 --register
```

La nuova macchina virtuale sarà fornita di un sistema operativo appartenente ad una distribuzione di linux, avente un versione del kernel di tipo 2.6 o 3.x; il nome assegnato è, appunto, "Tiny" e, grazie all'opzione `--register`, avviene la registrazione ufficiale della vm, cioè la creazione dello specifico file XML contentente tutte le impostazioni della macchina.

Definire la memoria RAM

```
VBoxManage modifyvm "Tiny" --memory "1024MB" --acpi on
```

Attraverso il comando `modifyvm` è possibile modificare le configurazioni di una vm attualmente non in stato di "running". Prima di tutto occorre specifcare il nome o l'ID della macchina virtuale a cui apportare le modifiche, in questo caso "Tiny", e poi occorre elencale. Qui abbiamo aggiunto una RAM da 1GB e poi abbiamo attivato l'opzione acpi.
Con ACPI, o Advanced Configuration and Power Interface, si intende lo standard corrente per permettere a tutti i sistemi operativi di riconoscere l'hardware, configurare la scheda madre, i device e le impostazioni riguardanti il consumo energetico. Visto che tutti i moderni PC contengono questa funzionalità e che sia Window che Linux la supportano da anni, Virtualbox nella creazione della macchina virtuale tramite interfaccia GUI, la abilita di default.

Assegnare un disco fisso

```
VBoxManage createhd --filename "Tiny.vdi" --size 8000
VBoxManage storagectl "Tiny" --name "SATA" --add sata
          --controller IntelAHCI
VBoxManage storageattach "Tiny" --storagectl "SATA"
          --port 0 --device 0 --type hdd --medium Tiny.vdi
```

Con il primo comando si crea un file .vdi (Virtualbox Disk Image) di nome Tiny e avente una dimensione di circa 8GB. Successivamente viene creato un controller SATA, automaticamente assegnato alla vm Tiny. Infine si associa il controller SATA di Tiny al file .vdi e si informa Virtualbox di considerare tale file come l'hard disk di quella macchina virtuale.

Aggiungere un lettore dvd virtuale

```
VBoxManage storagectl "Tiny" --name "IDE" --add ide
          --controller PIIX4
VBoxManage storageattach "Tiny" --storagectl "IDE" --port 0
          --device 1 --type dvddrive
          --medium /home/user/Downloads/TinyCore-current.iso
```

Per prima cosa si crea un controller IDE e lo si assegna alla vm Tiny. Poi si associa il controller alla macchina virtuale, impostandolo come dvddrive ed infine si simula la presenza di un dvd inserito, contenente il file .iso del sistema operativo TinyCore.

Il boot del sistema operativo

```
VBoxManage modifyvm "Tiny" --boot1 dvd
VBoxManage modifyvm "Tiny" --boot2 disk
```

Grazie a questi due comandi, viene specificato l'ordine di boot del sistema operativo. Ciò ci permette di costringere la macchina virtuale ad eseguire un boot da dvd durante il suo primo avvio, in modo da poter effettuare l'installazione del sistema operativo. Successivamente sarà sufficiente espellere

il dvd virtuale (oppure cambiare nuovamente l'ordine di boot) per far sí che la vm esegua il boot dal disco fisso.

Seguendo questi passaggi sarà possibile creare una macchina virtuale completa e perfettamente operativa, ma non è sufficiente per il nostro scenario. Infatti, occorre ancora impostare la connessione di rete e il remote desktop server affinchè il D.C.O. possa visualizzare l'output grafico della macchina virtuale di Virtualbox nella propria postazione.

9.1.2 Configurazione della connessione

Nella Sez. 5.2.2 abbiamo elencato tutte le tipologie di connessione possibili in Virtualbox. Ogni configurazione ha un suo obiettivo e perciò racchiude in sé un certo numero di vantaggi e svantaggi. Prima di decidere quale di queste avremmo utilizzato per il nostro laboratorio virtuale di testing, abbiamo dovuto analizzare quali fossero i nostri requisiti, riguardo specificatamente alla connessione.
I vincoli a cui dobbiamo sottostare sono:

1. le macchine virtuali dovranno avere un particolare indirizzo IP, previsto già nelle configurazioni del software di gestione che si andrà a testare[1];

2. gli indirizzi IP presenti nel sistema virtuale di Virtualbox, cioè sia quello della macchina in cui risiede l'applicazione di gestione, che quello degli altri software in gioco, sono IP realmente esistenti e, probabilmente, pubblici;

3. deve essere possibile, per la macchina virtuale, raggiungere l'"esterno" dell'host, ma rimandendo vincolata al laboratorio presente in ∼Okeanos o all'indirizzo in cui è presente il D.C.O.;

[1] Ricordiamo che uno dei nostri requisiti di sistema ci impone di non poter modificare in alcun modo il software da testare.

4. deve essere possibile, per le entità esterne alla macchina virtuale, quali i software di simulazione del traffico ferroviaro presenti su altre macchine virtuali o il D.C.O., poter raggiungere direttamente il guest;

5. vista la necessità del software di gestione del traffico ferroviario di avere una certa sicurezza nel recapito dei pacchetti, è plausibile supporre che venga utilizzato il protocollo tcp.

Al fine di soddisfare tutte queste condizioni abbiamo optato per un uso combinato di due elementi:

- la configurazione di rete Virtualbox *Rete con NAT* (o Nat Network), unito alla funzionalità di *port forwarding*, per la macchina virtuale;

- un potente firewall, integrato già nel kernel linux, chiamato *iptables*, per le macchine di ∼Okeanos.

Andiamo ora a vedere nel dettaglio il contribuito che ciascuno di questi elementi ha dato al nostro laboratorio virtuale.

Nat Network

Come abbiamo avuto modo di vedere, questa particolare configurazione di rete si basa sull'emulazione di un NAT e, all'occorrenza, anche di un server DHCP per permette di assegnare alla macchina guest un indirizzo IP qualsiasi; questo viene reso possibile perché, in ogni caso, una volta che il pacchetto viene inoltrato nella rete, verrà opportunamente modificato l'indirizzo IP del mittente, sostituendolo con quello dell'host. Così facendo abbiamo già risolto il primo dei vincoli precedentemente elencati.
Un'altra funzionalità molto utile è quella del *port forwarding*. Come spiegato nella Sez. 5.2.2, questa funzionalità permette di inoltrare direttamente al guest tutti i pacchetti che arrivano all'host su una determinata porta, purché esistano delle "regole" a specificare quali sono le entità coinvolte in questa operazione.

Iniziamo spiegando per gradi cosa occorre fare per predisporre la macchina guest con le impostazioni appena descritte. Prima di tutto, per poter utilizzare tale configurazione, occorre creare una Nat Network; ovviamente, sebbene questa sia un'operazione realizzabile anche per mezzo del frontend GUI, noi anche in questo caso utilizzeremo il VBoxManage.

Creare la Nat Network

```
VBoxManage natnetwork add --netname NatNetwork
                          --network 10.0.2.0/24
```

In questo modo viene creata una Nat Network di nome, appunto, "NatNetwork" ed avente un CIDR (Classless Inter-Domain Routing) a 24bit[2].

Inserire la regola del port forwarding

```
VBoxManage natnetwork modify --netname NatNetwork
           --port-forward-4
           "f:tcp:[83.212.126.188]:60000:[10.0.2.15]:60000"
```

Utilizzando questo comando si aggiunge alla rete precedentemente creata una regola chiamata "f", la quale impone che tutto il traffico tcp che giunge alla porta 60000 dell'indirizzo IPv4 83.212.126.188, cioè l'indirizzo pubblico della macchina di ∼Okeanos, debba essere inoltrato alla porta 60000 della macchina virtuale di Virtualbox avente indirizzo IP 10.0.2.15 .

Osservando la regola appena descritta, risulta semplice immaginare le potenzialità di questa operazione: con un unico comando è possibile

[2]Il CIDR è un nuovo schema di indirizzamento introdotto nel 1993 per sostituire lo schema classful. Permette, in un indirizzo IP, di definire quale parte indichi la sotto rete e quale gli host. In questo caso avremmo 24 bit per comporre la parte dell'indirizzo di rete e $y = 32 - 24$ bit che consentono di calcolare il numero di host nella sottorete tramite la formula $2^y - 2$.

infatti definire esattamente i soggetti a cui si riferisce, sia specificando l'indirizzo IPv4 di host e guest che precisando il tipo di traffico da inoltrare, evitando così che altre macchine virtuali appartenenti alla stessa Nat Network beneficino di tale configurazione (torneremo su questo punto più avanti).

Assegnare questa tipologia di rete alla macchina virtuale

```
VBoxManage modifyvm "Tiny" --nic1 natnetwork --cableconnected1 on
                           --nat-network1 NatNetwork
```

In questo caso, il fulcro del comando di configurazione è la macchina virtuale "Tiny", alla quale viene assegnata una particolare tipologia di rete. Tramite l'opzione `--nic1` si sceglie quale tipologia di rete utilizzare per connettere la prima PCI Ethernet card virtuale, con `--cableconnected1` si stabilisce che è una connessione wired, ed infine con `--nat-network1` si assegna il nome della Nat Network a cui si vuole agganciare il guest.

```
user     10134  3.2  6.6 530900 137412 pts/4   Sl+  12:36   0:13
 /usr/lib/virtualbox/VBoxHeadless --startvm tiny
user     10656  0.0  0.3 111288  7700 ?        Sl   12:38   0:00
 /usr/lib/virtualbox/VBoxNetDHCP --ip-address 10.0.2.3 --lower-i
p 10.0.2.4 --mac-address 08:00:27:C0:C8:EA --need-main on --netm
ask 255.255.255.0 --network NatNetwork --trunk-type whatever --u
pper-ip 10.0.2.254
user     10657  0.1  0.7 185000 14732 ?        Sl   12:38   0:00
 /usr/lib/virtualbox/VBoxNetNAT --ip-address 10.0.2.1 --netmask
255.255.255.0 --network NatNetwork --trunk-type whatever
```

Fig. 9.1: Processi che emulano la Nat Network in Virtualbox

Una volta terminata questa procedura, ogni volta che si avvierà la macchina virtuale Tiny, verranno attivati anche tutti i processi di Virtualbox necessari per emulare le caratteristiche di una rete privata con Nat. Nella Fig. 9.1 possiamo osservare quali sono esattamente i processi in esecuzione che gestiscono la NatNetwork di Virtualbox.

Ovviamente questa procedura di configurazione è stata effettuata in en-

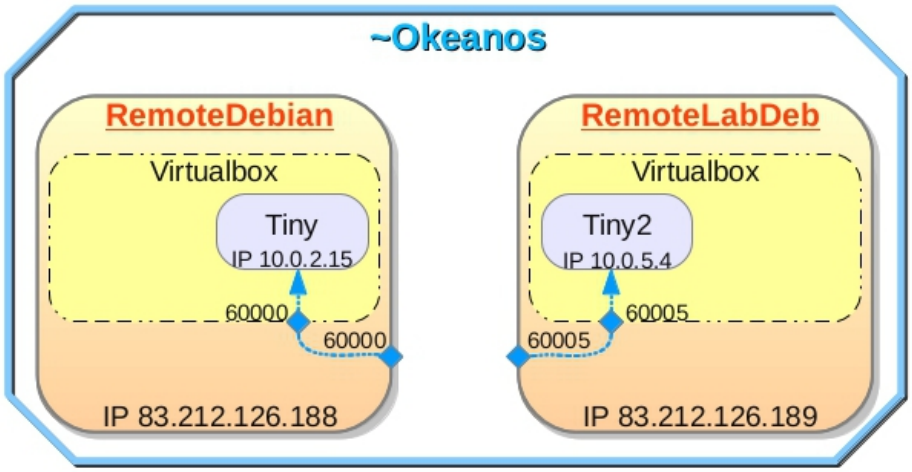

Fig. 9.2: Laboratorio Virtuale con Port Forwarding

trambe le macchine di ∼Okeanos. Possiamo vedere nella Fig. 9.3 una rappresentazione grafica della tecnica di port forwarding appena descritta.

iptables

Iptables è un'applicazione di livello utente che permette di definire delle regole e configurazioni riguardanti il traffico di rete della macchina su cui viene utilizzata. E' conosciuta come firewall in quanto permette di assumerne il ruolo, bloccando eventuale traffico indesiderato in ingresso o mascherando l'indirizzo ip del mittente dei pacchetti in uscita. Esistono moltissime opzioni da utilizzare per personalizzare le regole di iptables ma in questa sezione vedremo solo quelle utili per il nostro contesto.
Nel nostro scenario abbiamo bisogno di deviare il traffico uscente dal guest Tiny all'altro guest Tiny2, il quale risiede nella seconda macchina di ∼Okeanos da noi istanziata per il laboratorio virtuale. Inoltre, per poter fare in modo che la macchina virtuale Tiny raggiunga direttamente Tiny2, occorre inviare i pacchetti sulla specifica porta designata per il port forwarding.
Prima di mostrare l'implementazione di iptables all'interno del nostro laboratorio virtuale, ricordiamo ancora una volta che l'applicazione che gira

all'interno del guest di Virtualbox non può essere modificata in alcun modo. Ciò significa che, se spedirà dei pacchetti, questi avranno come indirizzo IP di destinazione quello effettivamente attribuito ad una delle entità presenti nel sistema di gestione del traffico ferroviario.

```
iptables -t nat -A OUTPUT -p tcp -m tcp -d 10.0.5.4 -j DNAT
         --to-destination 83.212.126.189:60005
```

Grazie all'utilizzo di questo comando, è possibile nattare tutto il traffico in uscita (`OUTPUT`) di tipo tcp e diretto all'IP `10.0.5.4`, deviandolo verso l'IP `83.212.126.189`, cioè all'indirizzo pubblico della seconda macchina di ∼Okeanos istanziata per il nostro progetto. Viene inoltre specificata come porta di destinazione quella designata per il port forwarding verso Tiny2, la `60005`.
Inserendo questa regola tra le configurazioni di iptables rendiamo effettivamente possibile la comunicazione tra le due macchine virtuali Tiny e Tiny2 (vedi Fig. 9.3).

È importante puntualizzare che la Figura 9.3 ha il solo scopo di rappre-

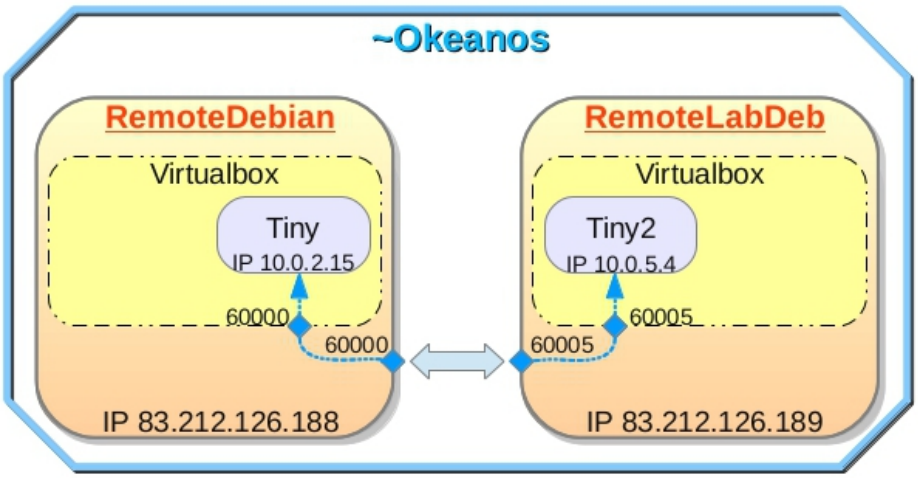

Fig. 9.3: Realizzazione della connessione nel Laboratorio Virtuale

sentare graficamente il sistema in modo semplificato. Perciò, sebbene la funzionalità del port forwarding permetta un inoltro dei pacchetti in ingresso

verso le macchine virtuali di Virtualbox, nulla impone che vengano utilizzate quelle stesse porte anche per il traffico in uscita.

9.1.3 Remote Desktop Server

Un'altra delle caratteristiche fondamentali per la creazione del nostro laboratorio virtuale è quello di poter visualizzare da remoto l'output grafico della macchina virtuale di Virtualbox. Questa risulta essere una condizione essenziale sia per permettere agli sviluppatori del software di gestione di visionare come sta procedendo il test, che per consentire al D.C.O. di interagire con l'applicazione.
Per realizzare tale funzionalità, Virtualbox mette a disposizione due strumenti:

- la *Virtualbox Remote Desktop Extension* che permette di effettuare l'accesso da remoto con alte prestazioni ad una vm attiva;
- il frontend *VBoxHeadless* che permette di avviare una vm senza avere un output visibile nell'host che la ospita.

Nel nostro caso, per rendere operativo il nostro laboratorio virtuale di testing, dobbiamo combinare entrambi questi strumenti.

Installare l'Extension Pack

```
VBoxManage extpack install <.vbox-extpack>
```

Utilizzando questo comando, si potrà installare l'Extension Pack, necessario per avere le funzionalità di Remote Display.

Avviare la vm senza output grafico

```
VBoxHeadless --startvm "Tiny"
```

Grazie al frontend VBoxHeadless, la macchina virtuale Tiny viene avviata e, contemporaneamente viene attivata l'estensione VRDE sulla porta

3389, quella di default. Infatti, l'output che appare sulla shell in risposta al precedente comando è:

```
Oracle VM VirtualBox Headless Interface 4.3.12
(C) 2008-2014 Oracle Corporation
All rights reserved.

VRDE server is listening on port 3389.
```

È importante sottolineare che sarebbe necessario modificare il numero della porta di default se si decidesse di attivare più di un VRDP server, visto che ogni porta può essere usata solo da un server per volta.

Visualizzare il desktop da remoto

Per poter visualizzare da remoto il desktop della macchina virtuale di Virtualbox occorre collegarsi, utilizzando un Remote Desktop Protocol client, all'indirizzo IP della macchina host che ospita Virtualbox, nel nostro caso ∼Okeanos. Ci sono diversi client utilizzabili, soprattutto in base al sistema operativo presente nella macchina locale da cui si vuole effettuare il collegamento. Ipotizzando di avere una macchina locale con installata una distribuzione di Linux, il comando è il seguente:

```
rdesktop -a 16 -N 83.212.126.188:3389
```

9.2 I risultati ottenuti

Una volta terminata la predisposizione di tutte le entità del sistema ad ospitare il nostro laboratorio virtuale di testing, abbiamo effettuato delle prove per verificarne il funzionamento.
Prima di tutto ci siamo collegati in SSH alle due macchine di ∼Okeanos, poi abbiamo inserito le regole di iptables necessarie per stabilire un canale diretto di comunicazione tra le due macchine virtuali di Virtualbox ed infine le abbiamo avviate utilizzando il frontend VBoxHeadless. Una volta fatto

ciò, ci siamo collegati tramite `rdesktop` sia a Tiny che a Tiny2, al fine di visualizzare in locale il desktop di entrambe.
A questo punto, abbiamo esaminato l'effettivo funzionamento della connessione utilizzando netcat[3]. A turno, abbiamo impostato una delle due macchine virtuali come Server e l'altra come Client, poi abbiamo avviato la comunicazione. Il test ha dato ottimi risultati e, in entrambe le configurazioni, la connessione viene stabilita con successo.
Oltre al test della connessione, abbiamo dovuto esaminare il funzionamento anche del nostro programma java per la modifica del clock. Purtroppo, non avendo il software di gestione del traffico ferroviario a nostra disposizione, non abbiamo potuto ideare un metodo per predisporre il laboratorio di virtualizzazione a far avviare automaticamente la modifica del clock, ma abbiamo dovuto ipotizzare la presenza di un "interruttore" che faccia avviare tutto il processo di rallentamento delle macchine virtuali. Così abbiamo scritto uno script, il cui unico compito è quello di avviare l'esecuzione del programma java, in collaborazione con le librerie glue di XPCOM (vedi Sez. 8.2.2).
Per effettuare questa prova abbiamo avviato la macchina virtuale Tiny con VBoxHeadless, ne abbiamo rediretto l'output grafico con rdesktop ed infine abbiamo eseguito un semplice programmino il cui scopo era stampare a video la scritta "Tick", una volta al secondo. Contemporaneamente a tutto ciò, in un'altra shell collegata sempre in ssh alla macchina di ∼Okeanos che ospita Tiny, abbiamo attivato il nostro script, rallentando il clock della vm del 50%. Anche questo test ha soddisfatto le nostre aspettative: la macchina virtuale Tiny veniva spenta e riaccesa in automatico e, alla riaccensione, il programmino che girava al suo interno era ancora in esecuzione, ma notevolmente più lento rispetto a quanto facesse prima dell'attivazione della procedura.

[3]Netcat è un programma open source a riga di comando che permette di connettere due dispositivi al fine di realizzare una comunicazione.

9.2.1 Il problema della connessione

Un'altro test che abbiamo effettuato, consisteva nell'avviare una comunicazione tra le due macchine virtuli, Tiny e Tiny2, tramite netcat, e poi avviare la procedura di rallentamento del clock. In questo caso però l'esito del test ci ha posto di fronte ad una problematica non prevista: il mantenimento della connessione tra le vm.
Infatti abbiamo scoperto che, quando si spegne la macchina virtuale di Virtualbox, anche i processi che emulano il funzionamento della connessione smettono di funzionare (vedi Fig. 9.1). Ciò comporta una chiusura della connessione, con conseguente interruzione della comunicazione stabilita tramite netcat.
Per risolvere questo problema, abbiamo sfruttato l'idea stessa su cui si basa la tipologia di connessione Nat Network. Ci siamo cioè serviti della possibilità di creare una "rete privata" tra macchine di Virtualbox appartenenti alla stessa Nat Network, al fine di "mantenere attivi i processi". In altre parole, abbiamo creato una nuova macchina virtuale di Virtualbox, l'abbiamo configurata con la stessa Nat Network di Tiny ed infine l'abbiamo avviata e messa in pausa[4].

Grazie a questa nuova macchina virtuale di appoggio, che abbiamo chiamato "Temp", i processi che emulano la rete rimangono comunque in esecuzione, anche negli attimi in cui Tiny non è in stato di running. Così facendo, la connessione non si interrompe e lo stesso vale per la comunicazione stabilita con netcat tra le macchine vituali Tiny e Tiny2.
Osservando perciò lo stato finale del sistema nella Fig. 9.4, possiamo vedere due macchine virtuali di Virtualbox per ogni macchina di ∼Okeanos; entrambe le vm ospitate dalla stessa macchina utilizzano la medesima configurazione di rete di Nat Network (infatti hanno un indirizzo IP della stessa sottorete), ma una di esse, più esattamente Temp da un lato e Temp2 dall'altro, è in stato di pausa.

[4]Mettendo la macchina virtuale in *pausa* la si mantiene comunque attiva, ma si notifica a Virtualbox che, mentre è in questo stato, la vm non eseguirà alcuna operazione.

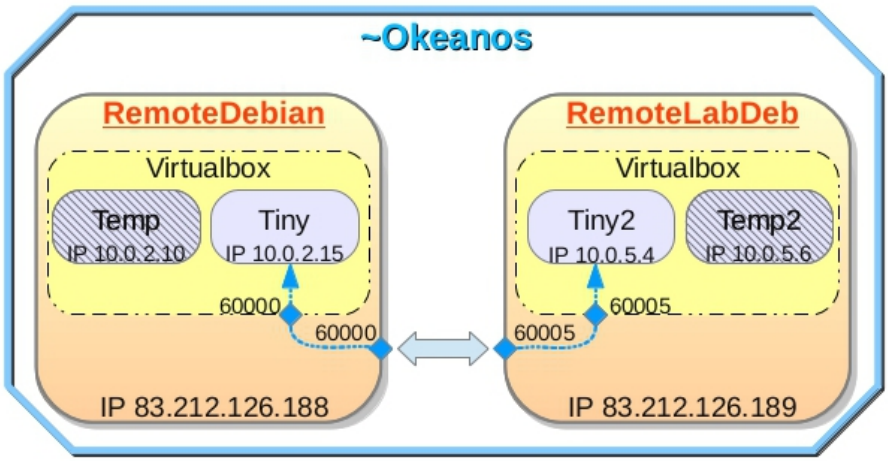

Fig. 9.4: Laboratorio Virtuale con due vm di Virtualbox

9.2.2 Le prestazioni

Il problema precedentemente descritto solleva sicuramente dei dubbi riguardanti le prestazioni del sistema. A tale scopo abbiamo effettuato delle misurazioni utilizzando il tool messo a disposizione direttamente da ~Okeanos per controllare le performance, sia in termini di cpu che in termini di rete, delle proprie macchine.
Nella Figura 9.5 viene mostrato un grafico rappresentate la percentuale di utilizzo di cpu della macchina di ~Okeanos in due casi: il primo in cui ci sono due macchine virtuali di Virtualbox in stato di running, nel secondo in cui solo una è attiva e in stato di running, mentre l'altra è in pausa. Come possiamo vedere, la percentuale di utilizzo della cpu è comunque inferiore al 18%, ma nel secondo caso non raggiunge nemmeno il 16%.

Invece, nella Figura 9.6 abbiamo preso in esame l'evento di modifica del clock. In entrambi i casi vi è un periodo iniziale di normale esecuzione della macchina virtuale, seguito poi da un secondo periodo in cui la vm è stata rallentata del 50%. La differenza tra le due misurazioni successive è che nella prima si sta esaminando il caso di una sola vm attiva, mentre nel secondo ci sono due vm attive, di cui una in pausa. Osservando la figura possiamo fare

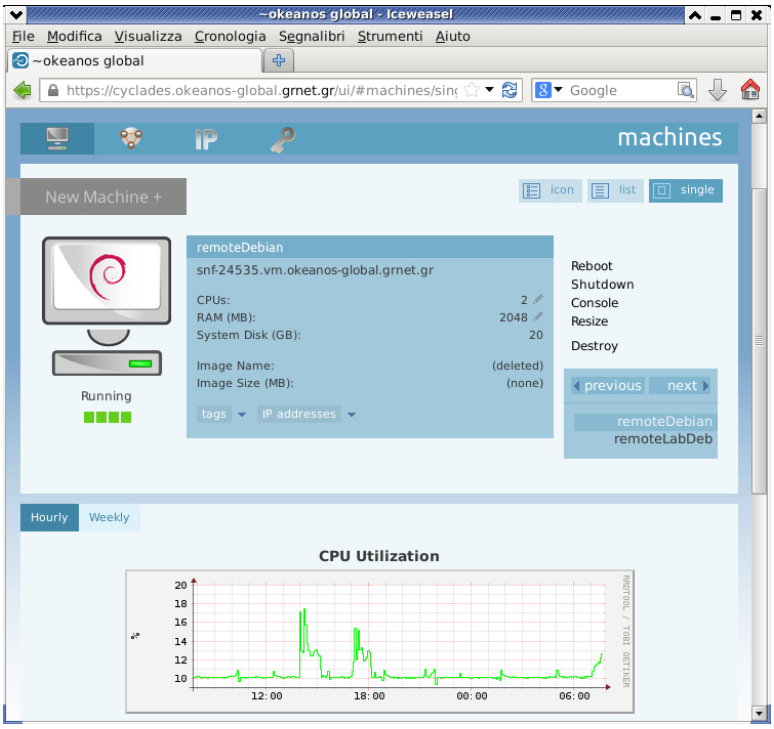

Fig. 9.5: Prestazioni di una macchina di ~Okeanos con due vm attive

due constatazioni:

1. la percentuale di utilizzo della cpu cresce notevolmente durante la fase di "rallentamento del clock"[5];

2. questo aumento di utilizzo della cpu rende pressoché irrilevante la differenza di prestazioni tra l'utilizzo di una o di due macchine virtuali di Virtualbox.

Inoltre, possiamo anche far notare che in entrambi i casi ci troviamo sotto una percentuale di utilizzo pari al 60%.
Per concludere è bene evidenziare che, essendo il nostro laboratorio virtuale localizzato in un cloud, o più esattamente un IaaS, sarebbe possibile sopperire

[5]Ciò avviene a prescindere della percentuale utilizzata nella configurazione del rallentamento.

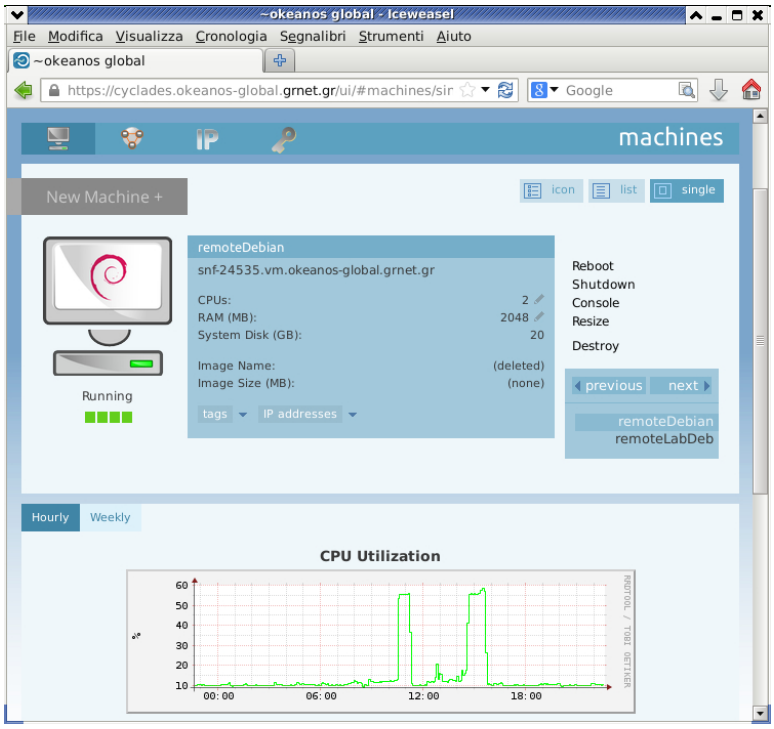

Fig. 9.6: Prestazioni di una macchina di ~Okeanos prima e dopo la modifica del clock

al bisogno di maggiore potenza computazionale semplicemente richiedendo più risorse o, nel nostro caso, più cpu[6].

[6]Ricordiamo che le nostre macchine presenti su ~Okeanos hanno solamente due cpu.

Conclusioni e Sviluppi Futuri

In questo elaborato abbiamo presentato e sviluppato la nostra idea, nata dall'osservazione dell'attuale sistema di testing del software di gestione del traffico ferroviario. Tale idea consiste nella creazione di un laboratorio virtuale di testing, delocalizzato nel cloud.
Attraverso uno studio approfondito delle entità in gioco, quali la virtualizzazione, la virtualizzazione annidata, il funzionamento del clock, il software di Virtualbox e le sue configurazioni, siamo stati in grado dare corpo al nostro progetto, scegliendo sempre la metodologia di realizzazione più efficiente ed efficace a nostra disposizione.
Nonostante i problemi riscontrati durante la predisposizione del sistema, siamo riusciti a creare ed a rendere operativo il nostro laboratorio virtuale, mantenendo fede ai requisiti prefissati durante lo studio dello scenario di applicazione.
I risultati prestazionali sono stati più che soddisfacenti e costituiscono delle solide basi per un futuro miglioramento del sistema.
I possibili sviluppi futuri rigardanti il nostro laboratorio virtuale di testing riguardano:

- uno studio più approfondito del software di gestione, in modo da poter impostare il laboratorio in base alle configurazioni reali;

- l'ideazione di un metodo che renda ancora più automatico la procedura di modifica del clock, attraverso l'attivazione automatizzata dello script;

- la definizione degli opportuni messaggi che il sistema distribuito dovrà scambiarsi per coordinare le diverse operazioni strutturali del laboratorio, come quella di modifica del clock;
- la progettazione di una configurazione di rete che permetta di evitare la presenza di due macchine virtuali di Virtualbox.

Bibliografia

[1] M. Boari e S. Balboni, "Mondo Digitale", *Tecniche di virtualizzazione: teoria e pratica*, pag.38-49, Marzo 2007.

[2] Creasy R.J., *The Origin of the VM/370 Time Sharing System*, IBM J., Research and Development, Settembre 1981.

[3] VMWare, *Virtualization Overview*, `http://www.nitro.ca/images/pdf/virtualization.pdf`, 19 Maggio 2006.

[4] VMWare, *Understanding Full Virtualization, Paravirtualization and Hardware Assist*, `http://www.vmware.com/files/pdf/virtualization.pdf`, 19 Maggio 2006.

[5] John R. Phelps, Philip Dawson *Hype Cycle for Virtualization*, Luglio 2009.

[6] Sergio Sagliocco, Diego Feruglio, Gianluca Ramunno, *La virtualizzazione e i suoi aspetti di sicurezza*.

[7] `http://it.wikipedia.org/wiki/Memoria_virtuale`.

[8] VMware vExpert, `http://vmnomad.blogspot.it/2011/05/basics-of-memory-management-part-2.html`, Maggio 2011.

[9] VMware, "Performance Evaluation of Intel EPT Hardware Assist", builds 140815 & 136362 (internal builds).

[10] Xiaolin Wang, Jiarui Zang, Zhenlin Wang, Yingwei Luo, Xiaoming Li, *Selective Hardware/Software Memory Virtualization*, 2011.

[11] Intel, "Intel®Virtualization Technology for Directed I/O", *Architecture Specification*, Settembre 2013.

[12] Claudio Chinosi, *Un confronto prestazionale tra tecnologie di virtualizzazione* 2009.

[13] Sam Verboven, Ruben Van den Bossche, Olivier Berghmans, Kurt Vanmechelen and Jan Broeckhove, "Evaluating Nested Virtualization Support", `http://www.academia.edu/1869613/Evaluating_Nested_Virtualization_Support`, Department of Computer Science and Mathematics, University of Antwerpv.

[14] `http://www-03.ibm.com/systems/it/z/`

[15] Muli Ben-Yehuda, Michael D. Day, Zvi Dubitzky, Michael Factor, Nadav HarâEl, Abel Gordon, Anthony Liguori, Orit Wasserman, Ben-Ami Yassour, "The Turtles Project" emphDesign and Implementation of Nested Virtualization, IBM Research, IBM Linux Technology Center.

[16] Hewlett-Packard Corporation, Intel Corporation, Microsoft Corporation, Phoenix Technologies Ltd., Toshiba Corporation, Revision 5.0, *Advanced Configuration and Power Interface Specification*, 6 Dicembre 2011.

[17] Numascale©, Technology Background, White Paper, `http://www.numascale.com/numa_background.html`.

[18] Toasty Tech, "Windows 3.0 with Multimedia Extension", `http://toastytech.com/guis/win3mme.html` 1991

[19] VMware, "Timekeeping in VMware Virtual Machines", *Information guide*, `http://www.vmware.com/files/pdf/Timekeeping-In-VirtualMachines.pdf`, Novembre 2011.

[20] Intel, "Architecture and Technology, *Tecnologia Intel® Turbo Boost 2.0*, http://www.intel.it/content/www/it/it/architecture-and-technology/turbo-boost/turbo-boost-technology.html.

[21] Helen Custer, Microsoft Press, *Microsoft Windows NT "Resource Guide"*, http://support.microsoft.com/kb/99588, 1993.

[22] Windows Dev-center homepage, Develop Desktop technology, "Using Time", *Acquiring high-resolution time stamps*, http://msdn.microsoft.com/en-us/library/windows/desktop/dn553408(v=vs.85).aspx .

[23] Ron Bean, "The Clock Mini-HOWTO", *How Linux Keeps Track of Time*, http://www.tldp.org/HOWTO/Clock-2.html, Novembre 2000.

[24] Oracle Corporation, "Oracle VM Virtualbox®", *User Manual* https://www.virtualbox.org/manual/UserManual.html, 2004-2014.

[25] Oracle Corporation. "Oracle VM Virtualbox®", *Programming Guide and Reference*, http://download.virtualbox.org/virtualbox/SDKRef.pdf, Version 4.3.10, 2004-2014.

[26] Wikipedia, "Gestione del traffico ferroviario", http://it.wikipedia.org/wiki/Gestione_del_traffico_ferroviario.

[27] Evangelos Koukis, Panos Louridas, "Proceedings of Science", *∼Okeanos IaaS*, https://okeanos.grnet.gr/static/medialibrary/2012/04/vkoukis-egicf2012.pdf

[28] https://www.virtualbox.org/wiki/Linux_Downloads

[29] http://distro.ibiblio.org/tinycorelinux/

[30] Giovanni Cattani, "Modifica dinamica del clock di Virtualbox", http://amslaurea.unibo.it/7239/1/Cattani_Giovanni_tesi.pdf 2014.