Capitolo e7

Virtualizzazione e cloud

Alcune organizzazioni possiedono un multicomputer pur senza volerlo. Un esempio evidente è un'azienda che possiede un server email, un server Web, un server FTP, alcuni server per l'e-commerce e altri ancora. Tutti questi server sono in esecuzione su computer diversi nello stesso rack, interconnessi meditante una rete ad alta velocità; in altre parole, l'insieme forma un multicomputer. Uno dei motivi per cui questi server sono eseguiti su macchine separate è che una sola macchina potrebbe non essere in grado di gestire tutto il carico; l'altro è l'affidabilità: il management semplicemente non si fida del fatto che un sistema operativo sia in grado di funzionare 24 ore su 24, 365 giorni l'anno, senza mai un errore. Eseguendo ogni servizio su un computer separato, anche in caso di blocco di un server, gli altri non ne sarebbero coinvolti. Il principio è valido anche per questioni di sicurezza: in caso di attacco al server Web da parte di un intruso, questi non avrebbe accesso immediato anche alle email; questa caratteristica viene a volte chiamata **sandboxing**. Con questo sistema si ottengono isolamento e fault tolerance a scapito del costo e della complessità di gestione dato l'elevato numero di macchine coinvolte.

Questi sono solo un paio dei motivi per cui si preferiscono tenere macchine separate; spesso, per esempio, le organizzazioni utilizzano più di un sistema operativo per l'operatività quotidiana: un server Web sotto Linux, un server di email sotto Windows, un server di e-commerce per i clienti sotto OS X e altri servizi in esecuzione sotto diversi tipi di UNIX. Anche in questo caso, la soluzione funziona, ma decisamente non è economica.

Quali sono le possibili alternative? Una soluzione, molto diffusa, è l'uso della tecnologia delle macchine virtuali, che, sebbene sembri una cosa molto moderna, risale invece già agli anni Sessanta; ma l'uso che ne facciamo oggi è decisamente una novità. L'idea fondamentale è che ci sia un **VMM** (**Virtual Machine Monitor**) che crea l'illusione di più macchine (virtuali) in esecuzione sul medesimo hardware fisico. Il VMM viene chiamato anche **hypervisor**. Come si è visto nel Paragrafo 1.7.5, si distingue fra hypervisor di tipo 1, che vengono eseguiti direttamente su hardware, e hypervisor di tipo 2, che possono sfruttare tutti i servizi e le astrazioni offerti da un sistema operativo sottostante. In entrambi i casi, la **virtualizzazione** consente a un singolo computer di ospitare più macchine virtuali, ciascuna delle quali potenzialmente può eseguire un sistema operativo diverso.

Il vantaggio di questo metodo è che un errore in una delle macchine virtuali non blocca le altre. In un sistema virtualizzato, vi possono essere diversi server in esecuzione su diverse

macchine virtuali, mantenendo in questo modo il modello a guasto parziale proprio di un multicomputer, ma a un costo nettamente inferiore e con una maggiore facilità di manutenzione. Oggi è inoltre possibile eseguire sullo stesso hardware più sistemi operativi diversi, mantenendo i benefici dell'isolamento delle macchine virtuali in caso di attacco e molti altri vantaggi.

Il consolidamento dei server così ottenuto è un po' come mettere tutte le uova in un solo paniere: se il server che esegue tutte le macchine virtuali si blocca, il risultato è ancor più catastrofico del blocco di un singolo server dedicato. Il motivo per cui le virtualizzazioni funzionano è che la maggior parte dei problemi di interruzione dei servizi non è causata da errori hardware, ma da software progettato male, inaffidabile, pieno di bug e mal configurato; e ovviamente questo vale anche per i sistemi operativi. Con la tecnologia delle macchine virtuali, il solo software in esecuzione con alti privilegi è l'hypervisor, che ha un numero di righe di codice di due ordini di grandezza inferiore rispetto a un sistema operativo completo, e quindi un numero di bug di due ordini di grandezza minore. Un hypervisor è più semplice di un sistema operativo perché esegue un unico compito: emulare più copie dell'architettura hardware (solitamente Intel x86).

L'esecuzione di software nelle macchine virtuali, oltre a garantire un forte isolamento, ha anche altri vantaggi, uno dei quali è che la presenza di meno macchine fisiche fa risparmiare denaro in termini di acquisto di hardware e di consumo energetico. Per aziende come Amazon o Microsoft, che possono avere centinaia di migliaia di server che svolgono enormi quantità di compiti diversi in ogni data center, ridurre le esigenze in termini di macchine fisiche nei data center rappresenta un enorme risparmio. In effetti, spesso le aziende che forniscono servizi di server posizionano i propri data center in località remote, magari per trovarsi vicine a una centrale idroelettrica (e quindi disporre di energia elettrica a basso costo).

La virtualizzazione consente inoltre di provare nuove idee. Nelle grandi aziende spesso accade che singoli dipartimenti o gruppi abbiano un'idea interessante e che per implementarla vadano ad acquistare un nuovo server. Se poi l'idea funziona e diventano necessari centinaia di migliaia di server, il data center aziendale si allarga. Spesso è difficile portare il software sulle macchine esistenti, perché ogni applicazione potrebbe avere bisogno di una versione diversa del sistema operativo, delle proprie librerie e altro. Con le macchine virtuali, ogni applicazione può avere il proprio ambiente operativo.

Altro vantaggio delle macchine virtuali è che le operazioni di checkpoint e di migrazione di macchine virtuali (per esempio per meglio bilanciare il carico fra più server) sono molto più semplici rispetto alla migrazione di processi in esecuzione su un normale sistema operativo. In quest'ultimo caso, nelle tabelle del sistema operativo sono contenute moltissime informazioni critiche su ciascun processo, come informazioni sui file aperti, allarmi, gestori di segnali e altro. Quando si svolge la migrazione di una macchina virtuale, è sufficiente spostare memoria e immagini disco, poiché insieme con esse vengono spostate anche le tabelle del sistema operativo.

Un altro utilizzo delle macchine virtuali è il permettere l'esecuzione di applicazioni legacy su sistemi operativi (o versioni di sistemi operativi) non più supportati o che non funzionano sull'hardware in uso. In effetti, la capacità di eseguire in contemporanea applicazioni che funzionano sotto diversi sistemi operativi è uno degli argomenti più forti in favore delle macchine virtuali.

Un altro utilizzo importante delle macchine virtuali è lo sviluppo software. Un programmatore che voglia scrivere software funzionante sotto Windows 7, Windows 8, diverse versioni di Linux, FreeBSD, OpenBSD, NetBSD e OS X, e magari altri sistemi, non deve più acquistare una decina di computer e installarvi diversi sistemi operativi. Ovviamente può creare delle partizioni sul disco rigido e installare su ciascuna di esse un diverso sistema operativo, ma l'operazione è più complicata. Innanzitutto, un normale PC supporta solo quattro partizioni disco primarie, per quanto grande sia il disco. In secondo luogo, anche se è possibile installare nel blocco di avvio un programma per il boot multiplo, per lavorare su un altro sistema operativo occorre riavviare il sistema. Con le macchine virtuali, invece, tutti i sistemi operativi possono essere eseguiti in contemporanea, perché non sono altro che processi come tutti gli altri.

Ad oggi il caso d'uso forse più importante della virtualizzazione, del quale si parla più spesso, è il **cloud**. L'idea fondamentale del cloud è piuttosto semplice, ossia terzializzare le esigenze elaborative o di memorizzazione a un data center ben organizzato gestito da un'azienda specializzata e che impiega personale esperto. Poiché solitamente i data center appartengono ad altri, probabilmente sarà necessario pagare per l'uso delle risorse, ma almeno non ci si dovrà preoccupare della manutenzione, alimentazione e raffreddamento delle macchine fisiche. A causa dell'isolamento che la virtualizzazione è in grado di garantire, i fornitori di soluzioni nel cloud possono fare in modo che più clienti, anche concorrenti fra loro, condividano una singola macchina fisica: ciascun cliente avrà per sé una fetta della torta. Pur sapendo di correre il rischio di stiracchiare un po' questa metafora del cloud, è nostro dovere riferire che alcuni dei primi critici del cloud sostenevano che la torta era solamente in cielo e che un'organizzazione seria non avrebbe mai potuto affidare a risorse altrui i propri dati e le proprie elaborazioni più sensibili.

Oggi, invece, le macchine virtuali sul cloud sono impiegate da innumerevoli organizzazioni per svolgere i compiti più diversi e, anche se forse questo non vale per tutte le organizzazioni e per tutti i dati, non si può negare che il cloud computing sia stato un enorme successo.

7.1 Storia

Con tutto quello che si è detto negli ultimi anni sulla virtualizzazione, a volte ci dimentichiamo che gli standard per le macchine virtuali su Internet sono piuttosto datati, risalendo almeno agli anni Sessanta. IBM aveva già fatto sperimentazioni non con uno, ma con ben due hypervisor sviluppati in modo del tutto indipendente: **SIMMON** e **CP-40**. CP-40 era un progetto di ricerca, che fu quindi reimplementato come CP-67 a formare il programma di controllo **CP/CMS**, un sistema operativo per macchine virtuali sviluppato per IBM System/360 Model 67 e che, nel 1972, fu nuovamente reimplementato come **VM/370** per la serie System/370. La linea System/370 fu sostituita da IBM negli anni Novanta con il System/390; si trattò più che altro di un cambio di nome, dal momento che l'architettura su cui era basato rimase la stessa, per compatibilità verso il basso. Ovviamente, la tecnologia hardware era stata migliorata e le nuove macchine erano più grandi e veloci di quelle vecchie, ma per quanto riguarda la virtualizzazione niente era cambiato. Nel 2000, IBM ha rilasciato la serie zSeries, che supporta spazi di indirizzi virtuali a 64 bit, ma per il resto è del tutto

compatibile verso il basso con il System/360. Tutti questi sistemi supportavano la virtualizzazione decenni prima che diventasse comune sulle macchine x86.

Nel 1974, due informatici dell'UCLA, Gerald Popek e Robert Goldberg, pubblicarono un articolo fondamentale ("Formal Requirements for Virtualizable Third Generation Architectures", requisiti formali per architetture virtualizzabili di terza generazione), che elencava esattamente quali fossero le condizioni che un'architettura di computer dovrebbe soddisfare per supportare in modo efficiente la virtualizzazione (Popek e Goldberg, 1974). È impossibile pensare di scrivere un capitolo sulla virtualizzazione senza fare riferimento al loro lavoro e alla loro terminologia. È anche noto che la conosciuta architettura dell'x86, anch'essa nata negli anni Settanta, per decine di anni non ha soddisfatto quei requisiti; e non era la sola. Praticamente qualsiasi architettura dai tempi dei primi mainframe non riusciva a superare il test. Gli anni Settanta furono molto produttivi e videro la nascita di UNIX, Ethernet, del Cray-1, di Microsoft e di Apple.

Negli anni Novanta i ricercatori della Stanford University svilupparono un nuovo hypervisor chiamato **Disco** e fondarono quindi **VMware**, un gigante della virtualizzazione che offre hypervisor di tipo 1 e di tipo 2 e che oggi è valutato in miliardi di dollari (Bugnion et al., 1997, e Bugnion et al., 2012). Fra parentesi, la distinzione fra hypervisor "di tipo 1" e "di tipo 2" è anch'essa nata negli anni Settanta (Goldberg, 1972). VMware presentò la sua prima soluzione di virtualizzazione per gli x86 nel 1999; a essa seguirono quindi altri prodotti, come **Xen**, **KVM**, **VirtualBox**, **HyperV**, **Parallels** e molti altri. Sembra che i tempi per la virtualizzazione siano maturati solo allora, anche se la teoria era stata fissata nel 1974 e per decenni IBM aveva venduto computer che supportavano la virtualizzazione e la utilizzavano pesantemente. Nel 1999 divenne un fenomeno di massa, ma non era affatto nuovo, nonostante l'attenzione che attirò.

7.2 Requisiti della virtualizzazione

È importante che le macchine virtuali funzionino esattamente come i sistemi emulati; in particolare, dev'essere possibile avviarle esattamente come le macchine reali e installarvi i sistemi operativi desiderati, proprio come avviene su un hardware reale. Il compito di garantire questa illusione, e per di più in maniera efficiente, è affidato all'hypervisor, che dev'essere in grado di garantire tre elementi fondamentali.

1. **Sicurezza**. L'hypervisor deve avere il controllo completo delle risorse virtualizzate.
2. **Equivalenza**. Il comportamento di un programma in esecuzione all'interno di una macchina virtuale dev'essere identico a quello che lo stesso programma avrebbe in esecuzione sulla macchina reale.
3. **Efficienza**. La maggior parte del codice in esecuzione sulla macchina virtuale dovrebbe funzionare senza interventi da parte dell'hypervisor.

Un sistema certamente sicuro di eseguire le istruzioni è elaborarle tutte, una dopo l'altra, mediante un **interprete** (come Bochs) ed eseguire esattamente ciò che occorre a quella istruzione. Alcune istruzioni possono essere eseguite direttamente, ma non molte; per esempio, l'interprete può essere in grado di eseguire un'istruzione `INC` (incremento) così com'è, ma le istruzioni che non possono essere eseguite direttamente in maniera sicura devono essere simulate attraverso

l'interprete. Per esempio, non si può consentire al sistema operativo guest di disabilitare gli interrupt della macchina host o di modificare le mappature della tabella delle pagine. Il trucco consiste nel far solo credere al sistema operativo che gira all'interno dell'hypervisor di aver disabilitato gli interrupt, oppure di aver cambiato le mappature delle pagine della macchina. Vedremo più avanti come si fa; per ora, l'importante è sapere che l'interprete può essere sicuro e, se implementato con accortezza, anche equivalente alla macchina reale, ma in compenso le prestazioni sono pessime. Per soddisfare anche il criterio delle prestazioni, vedremo che i VMM cercano di eseguire direttamente la maggior parte del codice.

Passiamo ora alla equivalenza. La virtualizzazione è stata a lungo un problema sull'architettura x86, a causa dei difetti di architettura dell'Intel 386, trascinatisi per vent'anni di nuovo modello in nuovo modello di CPU in ragione della compatibilità verso il basso. In parole povere, ogni CPU con una modalità kernel e una modalità utente ha un insieme di istruzioni che si comporta in modo diverso quando viene eseguito in modalità kernel rispetto a quando viene eseguito in modalità utente; fra queste vi sono le istruzioni di I/O, le modifiche alle impostazioni della MMU e così via. Popek e Goldberg le chiamano **istruzioni sensibili**. Esiste poi un insieme di istruzioni che, quando vengono eseguite in modalità utente, causano un trap; Popek e Goldberg le chiamano **istruzioni privilegiate**. Nell'articolo succitato affermano per la prima volta che una macchina è virtualizzabile solo se le istruzioni sensibili sono un sottoinsieme di quelle privilegiate; in parole più semplici, se si cerca di eseguire in modalità utente un compito che non dovrebbe essere eseguito in questa modalità, l'hardware dovrebbe causare un trap. A differenza di IBM/370, che aveva questa caratteristica, il 386 di Intel non ce l'aveva; per esempio, l'istruzione `POPF` sostituisce il registro dei flag, che modifica il bit che abilita e disabilita gli interrupt. In modalità utente, il bit semplicemente non viene cambiato; la conseguenza è che il 386 e i suoi successori non potevano essere virtualizzati e quindi non potevano supportare direttamente un hypervisor.

La situazione è, in verità, ancora peggiore rispetto a quanto abbiamo accennato. Oltre ai problemi con le istruzioni che non causano trap in modalità utente, vi sono istruzioni che, in modalità utente, possono leggere stati sensibili senza causare trap. Per esempio, sui processori x86 prodotti prima del 2005, un programma può determinare se è in esecuzione in modalità utente o in modalità kernel leggendo il proprio selettore del segmento di codice. Un sistema operativo che si comporti in questo modo e scopra di trovarsi in modalità utente potrebbe prendere decisioni sbagliate basandosi su questa informazione.

Questo problema fu finalmente risolto quando Intel e AMD, a partire dal 2005, introdussero la virtualizzazione nelle proprie CPU. Sulle CPU Intel si chiama **VT** (**Virtualization Technology**); sulle CPU AMD prende il nome di **SVM** (**Secure Virtual Machine**). Più avanti nel testo useremo genericamente il termine VT. Entrambe queste soluzioni sono ispirate al lavoro svolto per IBM VM/370, ma ci sono alcune piccole differenze. L'idea fondamentale è creare dei contenitori all'interno dei quali eseguire le macchine virtuali. Quando in un computer si avvia un sistema operativo guest, questo continua a funzionare fino a che non solleva un'eccezione e passa un trap all'hypervisor, per esempio eseguendo un'istruzione di I/O. L'insieme di operazioni che causano un trap è controllato da una mappa di bit hardware impostata dall'hypervisor; con queste estensioni diventa possibile il classico approccio alla macchina virtuale chiamato **trap-and-emulate**.

Il lettore avrà probabilmente notato un'apparente contraddizione nella descrizione: da un lato si è detto che x86 non era virtualizzabile fino a che non sono state introdotte le

estensioni all'architettura nel 2005; dall'altro, si è visto che VMware ha lanciato il suo primo hypervisor x86 nel 1999. Come è possibile che queste due affermazioni siano contemporaneamente vere? La risposta è che gli hypervisor precedenti il 2005 non eseguivano realmente il sistema operativo guest originale, ma *riscrivevano* parte del codice durante l'esecuzione per sostituire le istruzioni che potevano causare problemi con sequenze di codice sicure che emulavano l'istruzione originaria. Si supponga, per esempio, che il sistema operativo guest esegua un'istruzione di I/O privilegiata, o che modifichi uno dei registri di controllo privilegiati della CPU (come il registro CR3, che contiene un puntatore alla directory delle pagine). È importante che le conseguenze di istruzioni di questo genere siano limitate alla sola macchina virtuale in cui sono eseguite e non tocchino le altre macchine virtuali né l'hypervisor. Un'istruzione di I/O non sicura, pertanto, veniva sostituita da un trap che, dopo un controllo di sicurezza, eseguiva un'istruzione equivalente, restituendo il risultato. Poiché si sta eseguendo una riscrittura, è possibile sostituire le istruzioni sensibili, ma non privilegiate; le altre istruzioni sono eseguite nativamente. La tecnica è detta **traduzione binaria** e se ne parlerà più approfonditamente nel Paragrafo 7.4.

Non è necessario riscrivere tutte le istruzioni sensibili; in particolare, i processi utente del guest possono essere normalmente eseguiti senza modifiche. Se l'istruzione è non privilegiata, ma sensibile, un diverso comportamento nei processi utente rispetto a quello nel kernel non costituisce un problema perché comunque viene eseguita in modalità utente. Per le istruzioni sensibili che sono anche privilegiate, si può ricorrere al classico trap-and-emulate. Ovviamente, il VMM deve assicurarsi di ricevere i trap corrispondenti; solitamente, a questo scopo contiene un modulo che viene eseguito nel kernel e ridirige i trap ai propri gestori.

Esiste una diversa forma di virtualizzazione, chiamata **paravirtualizzazione**, sensibilmente diversa dalla **virtualizzazione completa**, perché non ha come scopo la creazione di una macchina virtuale che appaia assolutamente identica all'hardware sottostante. Presenta, invece, un'interfaccia software che espone esplicitamente il fatto che si tratti di un ambiente virtualizzato, offrendo per esempio un insieme di **hypercall** (chiamate all'hypervisor), che consentono al guest di inviare all'hypervisor delle richieste esplicite (in maniera molto simile a una chiamata di sistema, che offre alle applicazioni i servizi garantiti dal kernel). I guest usano le hypercall per svolgere operazioni privilegiate sensibili come l'aggiornamento delle tabelle delle pagine; poiché però questi compiti sono svolti esplicitamente in cooperazione con l'hypervisor, il sistema può essere complessivamente più semplice e veloce.

Non sorprenderà il fatto che anche la paravirtualizzazione non sia affatto una novità. Il sistema operativo IBM VM aveva una funzionalità simile, sebbene si chiamasse in un altro modo, fin dal 1972. L'idea fu riportata in auge dai VMM Denali (Whitaker et al., 2002) e Xen (Barham et al., 2003). Rispetto alla virtualizzazione completa, il problema della paravirtualizzazione è che il guest deve conoscere l'API della macchina virtuale; ciò significa, di solito, che è necessario personalizzarlo specificamente per un determinato hypervisor.

Prima di approfondire la trattazione degli hypervisor di tipo 1 e di tipo 2, è importante ricordare che non tutta la tecnologia di virtualizzazione cerca di imbrogliare il guest facendogli credere di avere a propria disposizione tutto il sistema; a volte, lo scopo è semplicemente consentire l'esecuzione di un processo originariamente scritto per un sistema operativo diverso o per una diversa architettura. Si opera pertanto una distinzione fra una completa virtualizzazione del sistema e una **virtualizzazione a livello di processo**. Nel prosieguo di

questo capitolo ci concentreremo sulla prima, ma anche la virtualizzazione a livello di processo viene effettivamente impiegata nella pratica. Alcuni esempi molto noti sono lo strato di compatibilità WINE, che consente l'esecuzione di applicazioni in ambienti POSIX come Linux, BSD e OS X, e la versione a livello di processo dell'emulatore QEMU, che consente ad applicazioni progettate per un'architettura di essere eseguite su di un'altra.

7.3 Hypervisor di tipo 1 e di tipo 2

Goldberg (1972) distingueva due diversi metodi di virtualizzazione. Un tipo di hypervisor, chiamato **hypervisor di tipo 1**, è illustrato nella Figura 7.1(a). Tecnicamente è simile a un sistema operativo, poiché è il solo programma in esecuzione nella modalità più privilegiata. Il suo compito è supportare più copie dell'hardware reale, chiamate **macchine virtuali**, in modo simile a quello in cui viene eseguito un normale sistema operativo.
Un **hypervisor di tipo 2**, invece, come quello illustrato nella Figura 7.1(b), è totalmente diverso. Si tratta di un programma che si basa, per esempio, su Windows o Linux per allocare e schedulare le risorse, in modo molto simile a un processo. Ovviamente, anche un hypervisor di tipo 2 finge di essere un computer completo di CPU e diverse periferiche. Entrambi i tipi di hypervisor devono eseguire le istruzioni macchina in modo sicuro. Per esempio, un sistema operativo in esecuzione sopra l'hypervisor può modificare e addirittura scompigliare le proprie tabelle delle pagine, ma non quelle degli altri.

Il sistema operativo in esecuzione al di sopra dell'hypervisor è chiamato, in entrambi i casi, **sistema operativo guest**. Nel caso di un hypervisor di tipo 2, il sistema operativo che gestisce l'hardware prende il nome di **sistema operativo guest**. Il primo hypervisor di tipo 2 sul mercato degli x86 è stato **VMware Workstation** (Bugnion et al., 2012). In questo paragrafo presenteremo l'idea fondamentale; nel Paragrafo 7.12 verrà presentato invece uno studio su VMware.

Un hypervisor di tipo 2 (anche detto **hosted**) dipende, per gran parte della propria funzionalità, da un sistema operativo host come Windows, Linux o OS X. Quando viene avviato per la prima volta, funziona come un computer appena acceso e ha bisogno quindi

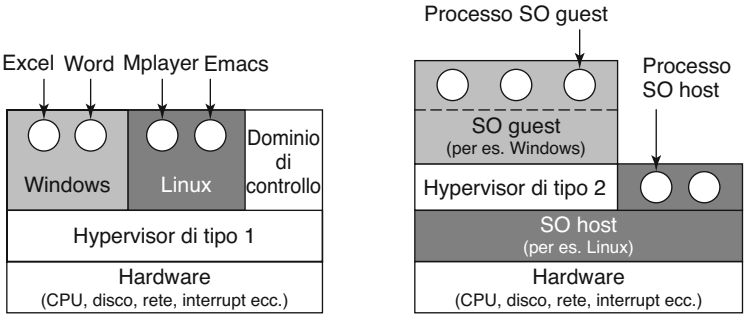

Figura 7.1 Hypervisor di tipo 1 e di tipo 2.

di un DVD, di una chiavetta USB o di un CD-ROM contenenti il sistema operativo. L'unità contenente il supporto, però, può essere virtuale: per esempio, è possibile memorizzare l'immagine come file ISO sul disco rigido dell'host e fare in modo che l'hypervisor finga di leggere da una vera unità DVD. Il sistema operativo viene quindi installato su un **disco virtuale** (che è in realtà semplicemente un file sotto Windows, Linux o OS X) eseguendo il programma di installazione che si trova sul DVD. Quando il sistema operativo è stato installato sul disco virtuale, può essere avviato ed eseguito.

Le varie categorie di virtualizzazione di cui si è parlato in precedenza sono riepilogate nella tabella riportata nella Figura 7.2, sia per gli hypervisor di tipo 1 sia per quelli di tipo 2; vengono elencati degli esempi per ogni combinazione di hypervisor e tipo di virtualizzazione.

7.4 Tecniche efficienti di virtualizzazione

La virtualizzabilità e le prestazioni sono questioni importanti, quindi esaminiamole più attentamente. Supponiamo per il momento di avere un hypervisor di tipo 1 che supporta una macchina virtuale, come illustrato nella Figura 7.3. Come tutti gli hypervisor di tipo 1, viene eseguito direttamente sull'hardware. La macchina virtuale viene eseguita come processo utente in modalità utente, pertanto non le è consentita l'esecuzione di istruzioni sensibili (nel senso di Popek-Goldberg). La macchina virtuale, tuttavia, esegue un sistema operativo, il quale ritiene di essere in modalità kernel (anche se, ovviamente, non lo è). Questo comportamento prende il nome di **modalità kernel virtuale**. La macchina virtuale esegue anche processi utente, che pensano di essere in modalità utente, come in effetti sono.

Che cosa accade quando il sistema operativo guest (che pensa di trovarsi in modalità kernel) esegue un'istruzione consentita solamente quando la CPU si trovi veramente in modalità kernel? Sulle CPU prive di VT l'istruzione normalmente fallisce e il sistema operativo si blocca. Se invece la CPU è provvista di VT, quando il sistema operativo guest esegue un'istruzione sensibile viene generato un trap sull'hypervisor, come illustrato nella Figura 7.3. L'hypervisor può quindi controllare l'istruzione per verificare se è stata richiesta

Metodo di virtualizzazione	**Hypervisor di tipo 1**	**Hypervisor di tipo 2**
Virtualizzazione senza supporto hardware	ESX Server 1.0	VMware Workstation 1
Paravirtualizzazione	Xen 1.0	
Virtualizzazione con supporto hardware	vSphere, Xen, Hyper-V	VMware Fusion, KVM, Parallels
Virtualizzazione dei processi		Wine

Figura 7.2 Esempi di hypervisor. Gli hypervisor di tipo 1 vengono eseguiti direttamente sull'hardware, mentre quelli di tipo 2 utilizzano i servizi offerti da un sistema operativo host installato.

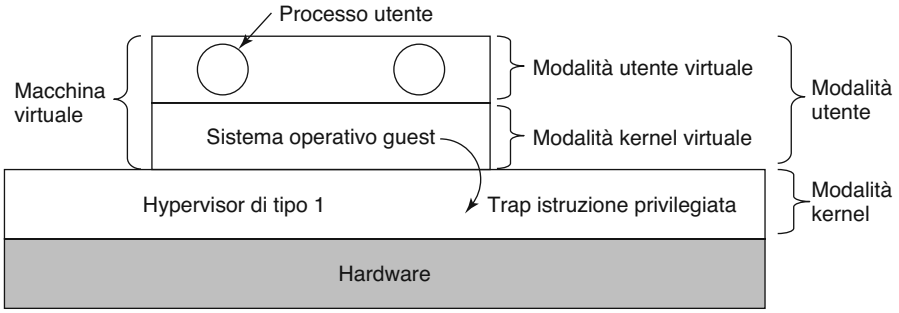

Figura 7.3 Quando il sistema operativo in una macchina virtuale esegue un'istruzione del kernel, se sul processore è presente una tecnologia di virtualizzazione invia il trap all'hypervisor.

dal sistema operativo guest all'interno della macchina virtuale oppure da un programma utente, sempre nella macchina virtuale. Nel primo caso, fa in modo che l'istruzione possa essere eseguita, mentre nel secondo emula il comportamento che avrebbe l'hardware reale qualora dovesse affrontare l'esecuzione di un'istruzione sensibile in modalità utente.

7.4.1 Virtualizzare ciò che non si può virtualizzare

Non è particolarmente difficile realizzare un sistema per l'esecuzione di una macchina virtuale, se la CPU è provvista di VT; ma come si procedeva prima dell'avvento della VT? Per esempio, VMware aveva prodotto un hypervisor molto prima dell'avvento delle estensioni di virtualizzazione sui processori x86. La risposta, anche in questo caso, è che i progettisti del software che hanno realizzato questi sistemi hanno sfruttato in maniera intelligente la **traduzione binaria** e le caratteristiche hardware già disponibili sugli x86, come gli **anelli di protezione** del processore.

Per molti anni i processori x86 hanno supportato quattro modalità di protezione, anche dette anelli; quello con i privilegi più bassi è l'anello 3 ed è in questo che vengono eseguiti i normali processi utente. In questo anello non si possono eseguire istruzioni privilegiate. L'anello con i privilegi più elevati è lo 0, che consente l'esecuzione di qualsiasi istruzione; nella normale operatività, il kernel viene eseguito nell'anello 0. Gli altri due anelli (1 e 2) non vengono utilizzati dal sistema operativo; ciò significa che possono essere tranquillamente utilizzati dagli hypervisor. Come si può vedere nella Figura 7.4, molte soluzioni di virtualizzazione mantenevano in modalità kernel (anello 0) l'hypervisor e le applicazioni in modalità utente (anello 3), facendo girare il sistema operativo guest in un livello con privilegi intermedi (anello 1). Come risultato, il kernel ha più privilegi rispetto ai processi utente e qualsiasi tentativo di accedere alla memoria del kernel da un programma utente genera una violazione di accesso; al contempo, le istruzioni privilegiate eseguite sul sistema operativo guest generano un trap sull'hypervisor, il quale effettua alcuni controlli di integrità prima di eseguire le istruzioni su richiesta del sistema operativo guest.

Per quanto riguarda le istruzioni sensibili nel codice kernel del guest, l'hypervisor fa in modo che non ne esistano più, riscrivendo il codice un blocco di base alla volta. Un **blocco**

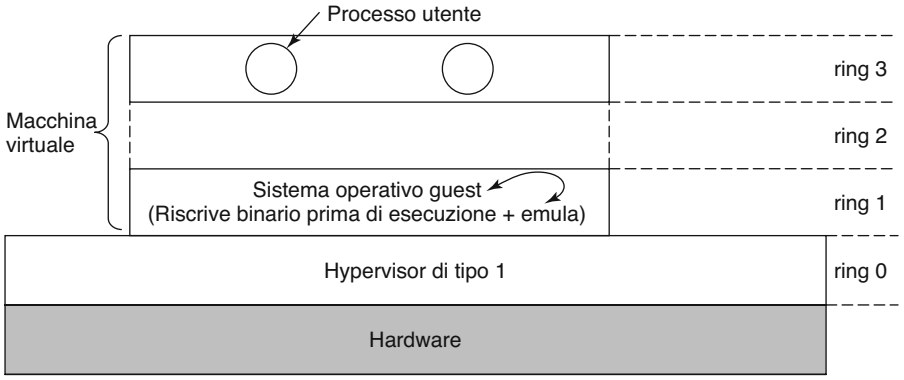

Figura 7.4 Il traduttore binario riscrive sull'anello 1 il sistema operativo guest in esecuzione, mentre l'hypervisor è in esecuzione nell'anello 0.

di base è una breve sequenza di istruzioni che termina con un'istruzione branch e che, per definizione, non contiene jump, call, trap, return o altre istruzioni che possano modificare il flusso di controllo, a parte l'ultima, che è appunto un'istruzione branch. Appena prima di eseguire un blocco di base, l'hypervisor lo esamina per verificare se contenga istruzioni sensibili (nel senso di Popek e Goldberg); in caso positivo, le sostituisce con una chiamata a una procedura dell'hypervisor che sia in grado di gestirle. L'ultima istruzione, branch, viene anch'essa sostituita da una chiamata all'interno dell'hypervisor, così da garantire che tutta la procedura possa essere ripetuta anche con il blocco di base successivo. La traduzione dinamica e l'emulazione possono sembrare eccessivamente impegnative, ma solitamente non lo sono: i blocchi tradotti vengono inseriti in una cache, che fa in modo che non occorra ripetere la traduzione in seguito. Inoltre, la maggior parte dei blocchi di codice non contiene istruzioni sensibili o privilegiate e può pertanto essere eseguita in maniera nativa. In particolare, il traduttore binario, se l'hypervisor configura l'hardware correttamente (come nel caso di VMware), può ignorare tutti i processi utenti, che sarebbero comunque eseguiti in modalità non privilegiata.

Dopo che il blocco di base è stato eseguito, il controllo torna all'hypervisor, che trova il blocco seguente; se questo è già stato tradotto, può essere eseguito immediatamente, altrimenti viene prima tradotto, quindi inserito in una cache ed eseguito. Alla fine, la maggior parte del programma si troverà nella cache e verrà eseguito quasi a velocità massima. Si usano anche diverse ottimizzazioni; per esempio, se un blocco di base termina con un salto (o una chiamata) a un altro blocco di base, si può sostituire l'istruzione finale con un salto o una chiamata diretti al blocco di base già tradotto, eliminando in questo modo l'overhead che sarebbe causato dalla ricerca del blocco successivo. Anche in questo caso non è necessario sostituire le istruzioni sensibili nel programma utente, perché l'hardware le ignorerebbe comunque.

Un'altra possibilità consiste invece nell'eseguire una traduzione binaria di tutto il codice del sistema operativo guest in esecuzione nell'anello 1, sostituendo anche le istruzioni privilegiate sensibili, che, in linea di principio, potrebbero generare dei trap. Il motivo è che i

trap sono molto esosi da gestire, mentre una traduzione binaria garantisce prestazioni migliori.

Finora è stato descritto un hypervisor di tipo 1. Anche se gli hypervisor di tipo 2 sono concettualmente diversi da quelli di tipo 1, utilizzano in linea di massima le medesime tecniche. Per esempio, VMware ESX Server (un hypervisor di tipo 1 presentato per la prima volta nel 2001) utilizzava esattamente la stessa traduzione binaria del primo VMware Workstation (un hypervisor di tipo 2 rilasciato due anni prima).

Per eseguire il codice del guest in modo nativo e utilizzare le medesime tecniche, però, è necessario che l'hypervisor di tipo 2 gestisca l'hardware a basso livello; e questo non lo si può fare dallo spazio utente. Per esempio, occorre impostare i descrittori di segmento esattamente al valore corretto per il codice guest. Per ottenere una virtualizzazione più fedele possibile, il sistema operativo guest dev'essere indotto a credere di avere il controllo completo di tutte le risorse macchina, con accesso all'intero spazio di indirizzi (4 GB sulle macchine a 32 bit). Nel momento in cui si dovesse accorgere di una presenza estranea nel proprio spazio utente, si potrebbero verificare dei problemi.

Purtroppo, è esattamente ciò che accade quando il guest esegue un processo utente su un sistema operativo normale. Per esempio, in Linux un processo utente ha accesso solamente a 3 GB dei 4 GB di spazio di indirizzi, perché 1 GB è riservato al kernel. Qualsiasi tentativo di accesso alla memoria del kernel provoca un trap. In linea di principio, sarebbe possibile prendere il trap ed emulare le azioni appropriate, ma si tratta di un'azione dispendiosa in termini di risorse e che comunque richiederebbe l'installazione, nel kernel host, del gestore di trap più adatto. Un altro (ovvio) metodo per risolvere questo problema della presenza estranea è riconfigurare il sistema in modo da rimuovere il sistema operativo host e concedere davvero al guest tutto lo spazio di indirizzi, operazione che però è ovviamente impossibile eseguire dallo spazio utente.

L'hypervisor deve essere anche in grado di gestire gli interrupt così che svolgano le azioni più appropriate, per esempio quando il disco invia un interrupt o si verifica un page fault. Anche in questo caso, se l'hypervisor desidera utilizzare il trap-and-emulate per le istruzioni privilegiate deve ricevere i trap; l'installazione di gestori di trap o di interrupt nel kernel non è possibile per i processi utente.

Oggi, pertanto, la maggior parte degli hypervisor di tipo 2 ha un modulo kernel in esecuzione nell'anello 0, mediante il quale possono manipolare l'hardware con delle istruzioni privilegiate. Ovviamente, manipolare l'hardware a basso livello e dare al guest la possibilità di accedere all'intero spazio di indirizzi è un'ottima cosa, ma a un certo punto l'hypervisor deve poter ripulire tutto e ripristinare il contesto originale del processore. Si supponga, per esempio, che il guest sia in esecuzione quando, da una periferica esterna, giunge un interrupt. Poiché un hypervisor di tipo 2 dipende dai driver di periferica dell'host per la gestione degli interrupt, dovrà riconfigurare completamente l'hardware per eseguire il codice del sistema operativo host. Quando il driver viene eseguito, dal suo punto di vista tutto sarà esattamente come ci si dovrebbe aspettare. L'hypervisor si comporta un po' come un giovanotto che organizza un festino a casa mentre i genitori non sono presenti: va benissimo risistemare i mobili completamente, purché, prima che i genitori tornino a casa, tutto venga rimesso a posto come era in origine. Il passaggio dalla configurazione hardware per il kernel host a quella per il sistema operativo guest è un'operazione nota come **commutazione di mondo** (world switch) e se ne parlerà più approfonditamente quando si parlerà di VMware, nel Paragrafo 7.12.

Ora dovrebbe essere chiaro come fanno a funzionare gli hypervisor, anche quando sono in esecuzione su hardware non virtualizzabile: le istruzioni sensibili nel kernel del guest sono sostituite da chiamate a procedure che le emulano. Nessuna istruzione sensibile richiesta dal sistema operativo guest viene mai eseguita direttamente dall'hardware fisico, ma viene trasformata in chiamate all'hypervisor, che la emula.

7.4.2 Costo della virtualizzazione

Ci si potrebbe aspettare che le CPU dotate di VT abbiano prestazioni che superano abbondantemente le tecniche software che fanno ricorso alla traduzione, mentre le misure mostrano che la situazione non è poi così chiara (Adams e Agesen, 2006). Alla fine, l'approccio trap-and-emulate utilizzato dall'hardware con VT genera molti trap, i quali, sull'hardware moderno, sono molto dispendiosi, perché sporcano la cache della CPU, i TLB e le tabelle di predizione delle diramazioni interne alla CPU. Se invece le istruzioni sensibili vengono sostituite da chiamate alle procedure dell'hypervisor all'interno del processo in esecuzione, non si verifica alcun overhead per la commutazione di contesto. Adams e Agesen dimostrano che, a volte, la soluzione software ha prestazioni migliori di quella hardware, secondo il carico di lavoro. Per questo motivo, alcuni hypervisor di tipo 1 (e di tipo 2) eseguono la traduzione binaria per questioni di prestazioni, anche se il software potrebbe essere eseguito correttamente anche senza essa.

Con la traduzione binaria, il codice in sé potrebbe essere più lento o più veloce rispetto al codice originario. Si supponga, per esempio, che il sistema operativo guest disattivi gli interrupt hardware utilizzando un'istruzione `CLI` ("clear interrupt", ripulisci interrupt). Questa istruzione, in alcune architetture, è molto lenta e richiede molte decine di cicli su determinate CPU, con pipeline profonde ed esecuzione fuori ordine. Dovrebbe ora essere chiaro che se il guest desidera disattivare gli interrupt ciò non significa che l'hypervisor li disattivi effettivamente, influenzando così il comportamento di tutta la macchina; deve invece disattivarli solo per il guest, senza disattivarli effettivamente. A questo scopo, per ogni guest deve tenere traccia di un **IF** (**interrupt flag**, flag di interrupt), all'interno della struttura dati della CPU virtuale gestita, assicurandosi che la macchina virtuale non riceva interrupt fino a che non vengono nuovamente disattivati. Ogni volta che nel guest viene eseguita un'istruzione `CLI`, sarà sostituita da un'istruzione come `"VirtualCPU.IF = 0"`, un'istruzione di spostamento poco dispendiosa che può richiedere da uno a tre cicli: il codice tradotto sarà pertanto più veloce di quello originario. Ciò detto, è comunque vero che, sulle macchine moderne con VT, solitamente le soluzioni hardware sono più veloci di quelle software.

D'altro canto, se il sistema operativo guest modifica le proprie tabelle delle pagine, questo può rappresentare un costo molto elevato. Il problema è che ogni sistema operativo guest su una macchina virtuale ritiene di "possedere" tutta la macchina e può pertanto mappare qualsiasi pagina virtuale a qualsiasi pagina fisica in memoria. Se però una macchina virtuale desidera utilizzare una pagina fisica già utilizzata da un'altra macchina virtuale (o dall'hypervisor), può succedere di tutto. Nel Paragrafo 7.6 vedremo che la soluzione consiste nell'aggiungere un altro livello di tabelle delle pagine per mappare le "pagine fisiche del guest" alle pagine fisiche effettive sull'host. È abbastanza intuibile che questa operazione di gestire più livelli di tabelle delle pagine non è economica.

7.5 Hypervisor o microkernel?

Gli hypervisor sia di tipo 1 sia di tipo 2 funzionano con sistemi operativi guest non modificati, ma per avere prestazioni decenti devono cercare di accontentare un po' tutti. Si è visto che la **paravirtualizzazione** usa un approccio diverso, modificando il codice sorgente del sistema operativo guest: invece di eseguire le istruzioni sensibili, il guest paravirtualizzato esegue delle **hypercall**. Il sistema operativo guest opera a tutti gli effetti come programma utente, che esegue chiamate di sistema al sistema operativo (l'hypervisor). Quando si sceglie questo approccio, l'hypervisor deve definire un'interfaccia formata da un insieme di chiamate a procedure che possano essere impiegate dal sistema operativo; questo insieme di chiamate viene a formare una vera e propria **API** (**application programming interface**, interfaccia di programmazione per le applicazioni), anche se si tratta di un'interfaccia che verrà utilizzata da un sistema operativo e non da un programma applicativo.

Facendo un passo avanti, quando dal sistema operativo si eliminano tutte le istruzioni sensibili, facendo in modo che per ottenere servizi come I/O si debbano eseguire delle hypercall, l'hypervisor viene di fatto trasformato in un microkernel, simile a quello della Figura 1.26. L'idea della paravirtualizzazione è che emulare determinate istruzioni hardware sia un compito sgradevole e molto impegnativo in termini di tempo: occorre una chiamata all'hypervisor, seguita dall'emulazione della esatta semantica di un'istruzione complessa. Molto meglio fare in modo che il sistema operativo guest chiami l'hypervisor (o il microkernel) perché esegua l'I/O.

Alcuni ricercatori sostengono che gli hypervisor andrebbero considerati alla stregua di "microkernel fatti come si deve" (Hand et al., 2005). Questo è un argomento molto controverso; alcuni ricercatori si sono schierati decisamente contro questo modo di vedere le cose, sostenendo che, tanto per cominciare, la differenza fra i due concetti non è poi fondamentale (Heiser et al., 2006). Altri invece suggeriscono che, rispetto ai microkernel, gli hypervisor potrebbero non essere adatti a realizzare sistemi sicuri e ritengono che sarebbe bene estenderli aggiungendovi funzionalità tipiche dei kernel, come lo scambio di messaggi e la condivisione della memoria (Hohmuth et al., 2004). Infine, alcuni ricercatori ritengono addirittura che forse gli hypervisor non siano nemmeno "ricerca sui sistemi operativi fatta come si deve" (Roscoe et al., 2007). Tutto considerato, riteniamo di fare bene a esaminare più approfonditamente le somiglianze fra hypervisor e microkernel.

Il motivo principale per cui i primi hypervisor emulavano la macchina completa era che il codice sorgente del sistema operativo guest non era disponibile (come nel caso di Windows), oppure che del sistema esistevano numerose varianti (come nel caso di Linux). Forse in futuro l'API dell'hypervisor/microkernel sarà standardizzata, e i sistemi operativi successivi saranno progettati per chiamare l'API invece di utilizzare istruzioni sensibili; se si facesse in questo modo, la tecnologia delle macchine virtuali sarebbe molto più facile da utilizzare e supportare.

La differenza fra una vera e propria virtualizzazione e la paravirtualizzazione è illustrata nella Figura 7.5, nella quale sono rappresentate due macchine virtuali supportate da un hardware dotato di VT. Sulla sinistra è presente, come sistema operativo guest, una versione di Windows non modificata; quando viene eseguita un'istruzione sensibile, l'hardware causa un trap verso l'hypervisor, che emula l'istruzione e ritorna. Sulla destra è presente una ver-

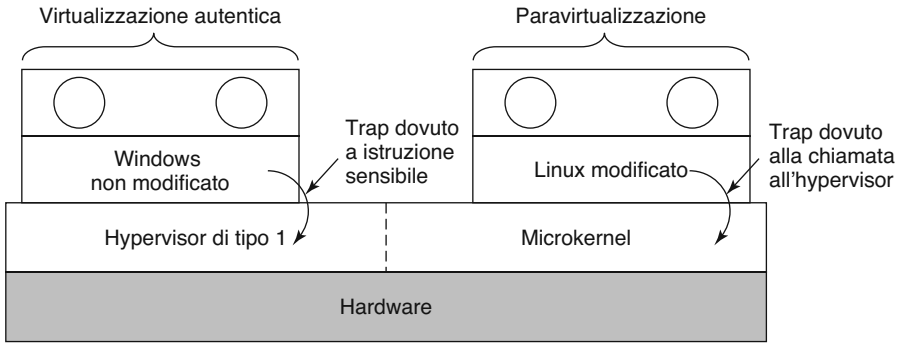

Figura 7.5 Virtualizzazione autentica e paravirtualizzazione.

sione modificata di Linux, priva di istruzioni sensibili. Quando ha bisogno di un'operazione di I/O o di modificare registri interni critici (come quello che punta alle tabelle delle pagine), per ottenere questo risultato esegue una chiamata all'hypervisor, esattamente come un programma applicativo esegue una chiamata di sistema sotto Linux.

Nella Figura 7.5 si vede come l'hypervisor sia diviso in due parti, separate da una linea tratteggiata. Nella realtà, sull'hardware viene eseguito un solo programma, una parte del quale si occupa dell'interpretazione delle istruzioni sensibili catturate da un trap, in questo caso provenienti da Windows. L'altra parte serve a gestire le hypercall; questa parte è chiamata "microkernel" nella figura. Se l'hypervisor deve eseguire solamente sistemi operativi guest paravirtualizzati, non è necessario emulare istruzioni sensibili e si può parlare allora di un vero microkernel, che fornisce solamente servizi basilari come il dispatch dei processi e la gestione della MMU. Il confine fra un hypervisor di tipo 1 e un microkernel è già vago e diventerà ancora meno chiaro a mano a mano che gli hypervisor avranno sempre più funzionalità e saranno in grado di eseguire delle hypercall, come probabilmente accadrà. Anche in questo caso l'argomento è controverso, ma sta diventando sempre più evidente che il programma in esecuzione in modalità kernel direttamente sull'hardware dovrebbe essere il più piccolo e affidabile possibile e consistere di migliaia, e non di milioni, di righe di codice.

Paravirtualizzare il sistema operativo guest solleva una serie di questioni. Innanzitutto, se le istruzioni sensibili sono sostituite da chiamate all'hypervisor, in che modo è possibile eseguire il sistema operativo su hardware nativo? L'hardware, infatti, non è in grado di comprendere queste hypercall. In secondo luogo, che cosa accade se vi sono più hypervisor presenti sul mercato, come VMware, il software open source Xen originariamente sviluppato dalla University of Cambridge e Microsoft Hyper-V, ciascuno dei quali presenta un hypervisor con API diverse da quelle degli altri? Come si fa a modificare il kernel in modo che possa essere eseguito con tutti?

Amsden et al. (2006) hanno proposto una soluzione. Nel loro modello, il kernel è modificato in modo che chiami procedure speciali quando ha bisogno di svolgere un'operazione sensibile. Queste procedure prese insieme, chiamate **VMI** (**virtual machine interface**, interfaccia verso la macchina virtuale) formano uno strato a basso livello che si inter-

faccia con l'hardware o l'hypervisor. Queste procedure sono pensate per essere generiche e non legate ad alcuna specifica piattaforma hardware e a nessun particolare hypervisor.

Un esempio di questa tecnica è illustrato nella Figura 7.6 per una versione di Linux paravirtualizzata chiamata VMI Linux (VMIL). Quando VMI Linux è in esecuzione direttamente sull'hardware, dev'essere collegato con una libreria che fornisca le istruzioni effettive (sensibili) necessarie per svolgere il lavoro, come si può vedere nella Figura 7.6(a). Quando è in esecuzione su un hypervisor, per esempio VMware o Xen, il sistema operativo guest è collegato con librerie diverse, che eseguono le chiamate adatte (anch'esse diverse) all'hypervisor sottostante. In questo modo, il nucleo del sistema operativo è portabile, può essere virtualizzato in un hypervisor e rimanere efficiente.

Sono state presentate anche altre proposte di interfaccia per macchine virtuali; una molto nota prende il nome di **paravirt ops**. L'idea è concettualmente simile a quanto descritto in precedenza, anche se sono diversi i dettagli. Sostanzialmente, un gruppo di distributori Linux che comprende aziende come IBM, VMware, Xen e Red Hat ha sostenuto la creazione, per Linux, di un'interfaccia indipendente dall'hypervisor. Questa interfaccia, inclusa nel kernel principale dalla versione 2.6.23 in poi, consente al kernel di comunicare con qualsivoglia hypervisor stia gestendo l'hardware fisico.

7.6 Virtualizzazione della memoria

Fino a questo momento si è affrontata la questione di come virtualizzare la CPU, ma un sistema di computer non è formato esclusivamente da una CPU; ha anche una memoria e dei dispositivi di I/O, che devono essere anch'essi virtualizzati. Vediamo come.

I moderni sistemi operativi supportano tutti la memoria virtuale, che fondamentalmente è una mappatura delle pagine nello spazio di indirizzamento virtuale (o logico) sulle pagine della memoria fisica. Questa mappatura è definita mediante tabelle delle pagine multilivello. Solitamente la mappatura viene eseguita facendo in modo che il sistema ope-

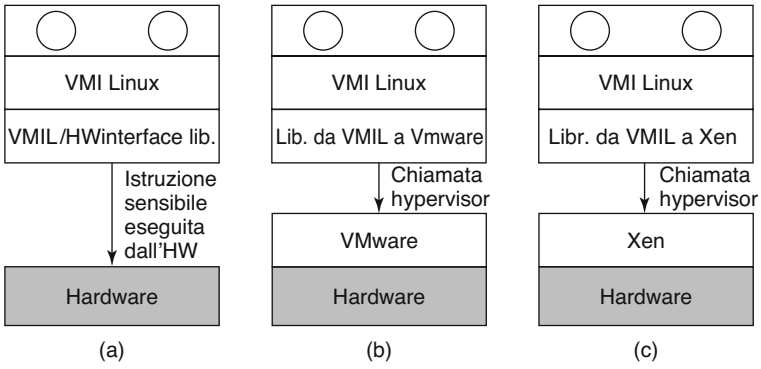

Figura 7.6. VMI Linux in esecuzione (a) direttamente sull'hardware, (b) su VMware, (c) su Xen.

rativo imposti un registro di controllo nella CPU che punti alla tabella delle pagine superiore. La virtualizzazione complica notevolmente la gestione della memoria, tanto che ci è voluto più di un tentativo per svolgerla in maniera corretta.

Si supponga, per esempio, di avere in esecuzione una macchina virtuale e che il sistema operativo guest decida di mappare le pagine virtuali 7, 4 e 3 rispettivamente sulle pagine fisiche 10, 11 e 12. Dovrà quindi costruire le tabelle delle pagine contenenti questa mappatura e caricare un registro hardware che punti alla tabella delle pagine superiore. Questa è un'istruzione sensibile: in una CPU dotata di VT, causerebbe un trap; con una traduzione dinamica causerebbe una chiamata a una procedura dell'hypervisor; in un sistema operativo paravirtualizzato, genererebbe una hypercall. Per semplicità supporremo di trovarci in un hypervisor di tipo 1 e che venga causato un trap, ma il problema è lo stesso in tutti e tre i casi.

Che cosa può fare ora l'hypervisor? Una possibile soluzione è allocare effettivamente le pagine di memoria fisica 10, 11 e 12 per questa macchina virtuale e impostare le effettive tabelle delle pagine in modo che mappino le pagine della macchina virtuale 7, 4 e 3. E fin qui tutto bene.

Ora si supponga di avviare una seconda macchina virtuale, che mappi le proprie pagine virtuali 4, 5 e 6 sulle pagine fisiche 10, 11 e 12 e carichi il registro di controllo perché punti alle proprie tabelle delle pagine. L'hypervisor cattura il trap, ma a questo punto non è chiaro come debba proseguire: non può utilizzare infatti questa mappatura, perché le pagine 10, 11 e 12 sono già in uso. Può trovare e utilizzare delle pagine libere, per esempio 20, 21 e 22, ma prima dovrà creare nuove tabelle delle pagine che mappino le pagine 4, 5 e 6 della seconda macchina virtuale sulle pagine 20, 21 e 22. Se viene avviata un'altra macchina virtuale che cerca di utilizzare le pagine 10, 11 e 12 occorre creare una mappatura apposita. In generale, per ogni macchina virtuale l'hypervisor deve creare una **tabella delle pagine shadow** che mappi le pagine virtuali utilizzate dalla macchina virtuale sulle pagine effettive assegnate dall'hypervisor.

Peggio ancora, ogni volta che il sistema operativo guest modifica le proprie tabelle delle pagine, l'hypervisor deve cambiare le tabelle delle pagine shadow. Per esempio, se il sistema operativo guest rimappa la pagina virtuale 7 su quella che vede come la pagina fisica 200 (al posto della 10), l'hypervisor deve venire a conoscere questa modifica. Il problema è che il sistema operativo guest può cambiare le proprie tabelle delle pagine semplicemente scrivendo nella memoria: non sono necessarie operazioni sensibili e pertanto l'hypervisor non viene nemmeno a conoscenza della modifica e sicuramente non può aggiornare la tabella delle pagine shadow utilizzata dall'hardware fisico.

Una possibile soluzione, sebbene un po' goffa, è che l'hypervisor tenga traccia di quale pagina della memoria virtuale del guest contenga la tabella delle pagine superiore. Questa informazione può essere ottenuta quando il guest cerca di caricare per la prima volta il registro hardware che punta alla tabella, perché l'istruzione è sensibile e, quindi, genera un trap. A questo punto l'hypervisor può creare una tabella delle pagine shadow e mappare come "a sola lettura" la tabella delle pagine superiore e le tabelle delle pagine a cui punta. I seguenti tentativi da parte del sistema operativo guest di modificare una tabella causano un page fault, passando così il controllo all'hypervisor che può analizzare il flusso di istruzioni, comprendere che cosa sta cercando di fare il guest e aggiornare di conseguenza le tabelle delle pagine shadow. Non è una soluzione elegante, ma funziona, in linea di principio.

Un'altra soluzione, altrettanto poco elegante, è fare esattamente il contrario. In questo caso, l'hypervisor consente semplicemente al guest di aggiungere a piacere qualsiasi mappatura alle sue tabelle delle pagine. Quando questo accade, nella tabella delle pagine shadow non cambia niente; in effetti, l'hypervisor nemmeno se ne accorge. Non appena però il guest cerca di accedere a una qualsiasi nuova pagina, si verifica un errore e il controllo torna all'hypervisor, il quale verifica il contenuto delle tabelle delle pagine del guest per vedere se vi deve aggiungere una mappatura; in caso positivo, la aggiunge e riesegue l'istruzione che ha causato l'errore. Se il guest rimuove una mappatura dalla tabella delle pagine, l'hypervisor non può aspettare che si verifichi un page fault, che non si verificherebbe. La rimozione di una mappatura da una tabella delle pagine avviene mediante l'istruzione `INVLPG` (che in realtà serve a invalidare una voce del TLB). L'hypervisor intercetta pertanto questa istruzione e rimuove la mappatura anche dalla tabella delle pagine shadow. Anche in questo caso, la soluzione non è elegante, però funziona.

Entrambe queste tecniche generano molti page fault, che sono dispendiosi. Solitamente si distingue fra "normali" page fault, causati dai programmi guest che accedono a una pagina al di fuori della RAM, e page fault correlati con il tentativo di mantenere la sincronizzazione fra le tabelle delle pagine shadow e quelle del guest. I primi sono **page fault generati dal guest** e, anche se vengono intercettati dall'hypervisor, devono essere restituiti al guest, il che non è affatto economico. I secondi sono **page fault generati dall'hypervisor** e vengono gestiti aggiornando le tabelle delle pagine shadow.

I page fault sono sempre dispendiosi, in modo particolare negli ambienti virtualizzati, perché tendono a generare le cosiddette uscite dalla VM. Un'**uscita dalla VM** è una situazione nella quale l'hypervisor riprende il controllo. Per eseguire un'uscita dalla VM, una CPU deve per prima cosa registrare la causa dell'uscita, in modo che l'hypervisor sappia come comportarsi, e registra inoltre l'indirizzo dell'istruzione del guest che ha causato l'uscita. Carica quindi lo stato del processore dell'hypervisor e solamente a questo punto l'hypervisor può gestire il page fault, che comunque ha un costo elevato. Al termine di tutto, bisogna ripetere al contrario tutta la procedura. L'intero processo può richiedere decine di migliaia di cicli, o più; non è strano pertanto che si cerchi in ogni modo di ridurre il numero di uscite.

In un sistema operativo paravirtualizzato la situazione è diversa; il sistema guest sa che, quando ha terminato di modificare la tabella delle pagine di alcuni processi, deve informare l'hypervisor e, di conseguenza, per prima cosa modifica completamente la tabella delle pagine, eseguendo quindi una chiamata all'hypervisor per avvertirlo della presenza della nuova tabella delle pagine. Invece di un errore di protezione a ogni aggiornamento della tabella delle pagine, pertanto, si ha un'unica hypercall quando la tabella è stata aggiornata, il che è evidentemente un modo molto più economico di gestire la situazione.

Supporto hardware alle tabelle delle pagine nidificate

Il costo della gestione delle tabelle delle pagine shadow ha portato i produttori di chip ad aggiungere un supporto hardware alle **tabelle delle pagine nidificate**, un termine utilizzato da AMD, mentre Intel preferisce usare la sigla **EPT** (**extended page tables**, tabelle delle pagine estese). Sono concetti simili, che mirano a eliminare la maggior parte dell'overhead gestendo via hardware le tabelle delle pagine, senza causare trap. È interessante notare che le prime estensioni di virtualizzazione nell'hardware dell'x86 non comprendevano alcun

supporto per la virtualizzazione della memoria. Anche se i processori con le estensioni VT hanno eliminato molti colli di bottiglia riguardanti la virtualizzazione delle CPU, toccare le tabelle delle pagine è oneroso quanto prima. Ci sono voluti alcuni anni prima che AMD e Intel riuscissero a produrre un hardware capace di virtualizzare con efficienza la memoria.

Si ricordi che, anche senza virtualizzazione, il sistema operativo mantiene una mappatura fra pagine virtuali e pagine fisiche. L'hardware esamina queste tabelle delle pagine per trovare l'indirizzo fisico corrispondente a un indirizzo virtuale; l'aggiunta di macchine virtuali non fa che aggiungere un'ulteriore mappatura. Si supponga, per esempio, di dover tradurre un indirizzo virtuale di un processo Linux in esecuzione su un hypervisor di tipo 1 come Xen o VMware ESX Server in un indirizzo fisico. Oltre all'**indirizzo virtuale del guest**, ora è presente anche un **indirizzo fisico del guest**, e di conseguenza un **indirizzo fisico dell'host** (che a volte prende il nome di **indirizzo fisico della macchina**). Senza EPT, è l'hypervisor il responsabile della manutenzione esplicita delle tabelle delle pagine; con l'EPT, l'hypervisor ha comunque una serie di tabelle delle pagine aggiuntive, ma la maggior parte del lavoro intermedio della gestione viene svolto dalla CPU direttamente nell'hardware. Nell'esempio visto poc'anzi, l'hardware verifica le tabelle delle pagine "regolari" per tradurre l'indirizzo virtuale del guest in un indirizzo fisico del guest, esattamente come farebbe senza virtualizzazione; la differenza è che esamina anche le tabelle delle pagine estese (o nidificate) senza alcun intervento da parte del software, per trovare l'indirizzo fisico dell'host, ogni volta che si accede all'indirizzo fisico del guest. La traduzione è illustrata nella Figura 7.7.

L'hardware, purtroppo, può aver bisogno di esaminare le tabelle delle pagine nidificate più spesso di quanto si possa pensare. Si supponga che l'indirizzo virtuale del guest non sia presente nella cache e che sia pertanto necessario un completo esame della tabella delle pagine. Ogni livello della gerarchia della paginazione richiede una ricerca all'interno delle tabelle delle pagine nidificate; in altre parole, il numero di riferimenti alla memoria aumenta con il quadrato della profondità della gerarchia. Ciononostante, la presenza di un EPT riduce drasticamente il numero di uscite dalla VM: gli hypervisor non hanno più bisogno di mappare la tabella delle pagine del guest come in sola lettura e possono tralasciare la gestione delle tabelle delle pagine shadow. In più, quando si passa alle macchine virtuali, l'EPT cambia semplicemente questa mappatura, allo stesso modo in cui un sistema operativo cambia la mappatura quando passa da un processo a un altro.

Reclamare la memoria

Avere una serie di macchine virtuali tutte in esecuzione sullo stesso hardware fisico, tutte con le proprie pagine di memoria e tutte che ritengono di avere il controllo assoluto è un'ottima cosa, almeno finché non occorre riappropriarsi della memoria, operazione che diventa particolarmente importante nel caso di un **overcommitment** (o sovraccarico) della memoria, situazione in cui l'hypervisor finge che la quantità totale della memoria a disposizione di tutte le macchine virtuali combinate insieme è superiore alla quantità totale di memoria fisica presente sul sistema. In generale, è un'ottima tecnica, perché consente all'hypervisor di gestire contemporaneamente più macchine virtuali ben equipaggiate. Per esempio, in una macchina con 32 GB di memoria, possono essere in esecuzione tre macchine virtuali ciascuna delle quali pensa di avere a disposizione 16 GB di RAM; ed evidentemente i conti non tornano. È abbastanza comune, però, che le tre macchine non abbiano bisogno contemporaneamente di tutta la memoria fisica, o magari che condividano pagine con lo stesso con-

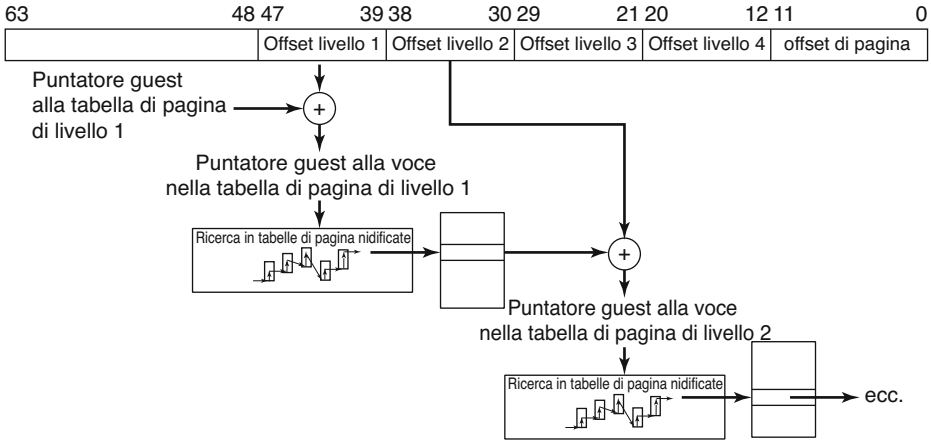

Figura 7.7. Le tabelle delle pagine estese/nidificate vengono esaminate ogniqualvolta si verifica un accesso alla memoria fisica del guest, compresi gli accessi per ciascun livello delle tabelle delle pagine del guest.

tenuto (come il kernel Linux) in macchine virtuali differenti, con una tecnica di ottimizzazione nota come **deduplicazione**. In questo caso, le tre macchine virtuali usano una quantità di memoria totale minore del triplo di 16 GB. Si parlerà più avanti di deduplicazione; per il momento è importante stabilire che ciò che in un determinato momento può sembrare un ottimo sistema per la distribuzione delle risorse potrebbe non esserlo altrettanto quando cambia il carico di lavoro. Magari la macchina virtuale 1 ha bisogno di più memoria, mentre la macchina virtuale 2 può farcela con meno pagine; in questo caso, sarebbe opportuno che l'hypervisor fosse in grado di trasferire risorse da una macchina virtuale all'altra, facendo in modo che tutto il sistema ne tragga beneficio. La domanda è: in che modo è possibile spostare delle pagine di memoria, se queste sono già state assegnate a una macchina virtuale?

In linea di principio, si potrebbe utilizzare un altro livello di paginazione. Quando occorre più memoria, l'hypervisor effettua lo swapping su disco delle pagine di una macchina virtuale, esattamente come un sistema operativo può fare lo swapping delle pagine di un'applicazione. Il problema di questo metodo è che il compito spetta all'hypervisor, che non ha idea di quali siano le pagine più importanti per il guest; è molto facile, pertanto, che venga fatto lo swapping delle pagine sbagliate. Anche se riuscisse a scegliere le pagine giuste di cui fare lo swapping (ossia le stesse pagine che avrebbe scelto il sistema operativo guest), si presentano comunque altri problemi.

Si supponga, per esempio, che l'hypervisor faccia lo swapping su disco di una pagina *P*. Poco più tardi, il sistema operativo guest decide di fare lo swapping della stessa pagina; purtroppo, lo spazio di swap dell'hypervisor e del guest non sono la stessa cosa. In altre parole, l'hypervisor deve per prima cosa riportare in memoria il contenuto delle pagine, che poi sarà scritto immediatamente su disco da parte del guest; il meccanismo non è molto efficiente.

Una soluzione molto diffusa è utilizzare un trucco noto come **ballooning**, in cui in ogni macchina virtuale viene caricato un piccolo modulo detto balloon, che funge da pseudo

driver che comunica con l'hypervisor. Il modulo balloon si può gonfiare in presenza di una richiesta da parte dell'hypervisor, allocando una serie di pagine contrassegnate e sgonfiandosi quando le dealloca. A mano a mano che il balloon si gonfia, aumenta la scarsità di memoria del guest; il guest risponde pertanto facendo lo swapping delle pagine meno importanti, il che è esattamente il comportamento desiderato. Quando invece il balloon si sgonfia, il guest può allocare più memoria che nel frattempo si è liberata. In altri termini, l'hypervisor inganna il sistema operativo, affidandogli il compito di prendere da sé le decisioni difficili.

7.7 Virtualizzazione dell'I/O

Dopo aver parlato di virtualizzazione della CPU e della memoria, esamineremo la virtualizzazione dell'I/O. Il sistema operativo guest si avvia solitamente esaminando l'hardware per sapere quali sono i dispositivi di I/O collegati, causando dei trap verso l'hypervisor. Che cosa deve fare l'hypervisor? Un possibile approccio richiede di informare che dischi, stampanti e così via siano quelli effettivamente collegati all'hardware. Il guest dovrà quindi caricare i driver di queste periferiche per utilizzarle. Quando i driver tentano di eseguire un I/O, leggono e scrivono sui registri hardware della periferica. Sono istruzioni sensibili, che causano un trap sull'hypervisor, che a sua volta può copiare i valori necessari da e verso i registri hardware, secondo necessità.

Anche in questo caso, però, si presenta un problema: ogni sistema operativo guest ritiene di avere il controllo di un'intera partizione su disco e vi possono essere molte più macchine virtuali (anche centinaia) di partizioni su disco. La soluzione solitamente adottata è che l'hypervisor crei un file o una regione del disco per ciascun disco fisico di ogni macchina virtuale. Poiché il sistema operativo guest cerca di controllare un disco posseduto dall'hardware reale (e che l'hypervisor comprende), può convertire il numero di blocco a cui ha accesso in un offset nel file o nella regione del disco utilizzati per la memorizzazione ed eseguire quindi l'I/O.

È anche possibile che il guest utilizzi un disco diverso da quello reale. Per esempio, se il disco reale è un nuovissimo disco ad alte prestazioni (o un sistema RAID) con una nuova interfaccia, l'hypervisor può comunicare al sistema operativo guest che è in presenza di un normale, vecchio disco IDE, lasciando che il sistema installi un driver disco IDE. Quando il driver invia dei comandi IDE al disco, l'hypervisor li converte in comandi per il nuovo disco. È possibile utilizzare questa strategia per aggiornare l'hardware senza dover modificare il software; e in effetti la capacità delle macchine virtuali di rimappare i dispositivi hardware è stato uno dei motivi per cui si è tanto diffuso il sistema VM/370: le aziende volevano avere la possibilità di acquistare hardware nuovo e più veloce, senza però cambiare il software. E questo divenne possibile con la tecnologia delle macchine virtuali.

Un'altra interessante tendenza relativa all'I/O è che l'hypervisor può assumere il ruolo di switch virtuale, nel qual caso ogni macchina virtuale è dotata del proprio indirizzo MAC e l'hypervisor può passare frame da una macchina all'altra, esattamente come farebbe uno switch Ethernet. Gli switch virtuali hanno diversi vantaggi; per esempio, sono molto facili da riconfigurare ed è possibile potenziarli aggiungendo funzionalità, per esempio per migliorarne la sicurezza.

I/O MMU

Un altro problema di I/O che bisogna in qualche modo risolvere è l'uso del DMA, che utilizza indirizzi di memoria assoluti. Anche in questo caso, è intuibile che l'hypervisor debba intervenire rimappando gli indirizzi prima dell'avvio del DMA. Esiste già, tuttavia, hardware con una **I/O MMU**, che virtualizza l'I/O allo stesso modo in cui la MMU virtualizza la memoria. Esistono diversi tipi di I/O MMU, per diverse architetture di processori. Anche se in questo capitolo parleremo solo della famiglia x86, Intel e AMD hanno tecnologie leggermente diverse, anche se l'idea che sta alla base è la stessa. Questo hardware elimina il problema del DMA.

Come le MMU normali, le I/O MMU utilizzano le tabelle delle pagine per mappare su un indirizzo fisico un indirizzo di memoria che un dispositivo desidera utilizzare (l'indirizzo del dispositivo). In un ambiente virtuale, l'hypervisor può impostare le tabelle delle pagine in modo che un dispositivo che esegua un DMA non si trovi a cercare di utilizzare memoria che non appartiene alla macchina virtuale in cui è in esecuzione.

Le I/O MMU hanno numerosi vantaggi quando si deve gestire un dispositivo in un ambiente virtualizzato. Il **pass through di dispositivo** consente di assegnare un dispositivo fisico direttamente a una determinata macchina virtuale. In generale, l'ideale sarebbe che lo spazio di indirizzi di un dispositivo fosse esattamente lo stesso dello spazio di indirizzi fisico del guest, ma questo è improbabile, a meno che non si disponga di una I/O MMU. L'MMU consente di rimappare gli indirizzi in maniera trasparente; sia il dispositivo sia la macchina virtuale non si rendono conto della traduzione di indirizzi che viene svolta in background.

L'**isolamento dei dispositivi** si assicura che un dispositivo assegnato a una macchina virtuale possa accedere direttamente alla macchina senza mettere a rischio l'integrità degli altri guest; in altre parole, l'I/O MMU evita la presenza di traffico DMA potenzialmente dannoso, esattamente come una normale MMU eviterebbe un accesso potenzialmente dannoso alla memoria da parte dei processi; in entrambi i casi, gli accessi a pagine non mappate danno come risultato un errore.

Purtroppo, però, DMA e indirizzi non rappresentano l'intera questione dell'I/O; per completezza, occorre anche virtualizzare gli interrupt, in modo che quelli generati da un dispositivo arrivino alla macchina virtuale giusta, con il giusto numero di interrupt. Le moderne I/O MMU supportano pertanto la **rimappatura degli interrupt**. Per esempio, un dispositivo invia un messaggio, segnalato mediante l'interrupt numero 1; il messaggio raggiunge innanzitutto la I/O MMU, che utilizza quindi la tabella di rimappatura degli interrupt per tradurlo in un nuovo messaggio destinato alla CPU che in questo momento ha la macchina virtuale in esecuzione, con il numero di vettore che la macchina stessa si attende (per esempio, 66).

Infine, la presenza di una I/O MMU aiuta i dispositivi a 32 bit ad accedere alla memoria al di sopra dei 4 GB. Questi dispositivi normalmente non possono accedere direttamente, con un DMA, agli indirizzi al di sopra dei 4 GB, mentre la presenza di una I/O MMU consente di rimappare facilmente un indirizzo di memoria basso su qualsiasi indirizzo dello spazio di indirizzi fisico più grande.

Domini di dispositivo

Un approccio differente alla gestione dell'I/O è dedicare una delle macchine virtuali all'esecuzione di un sistema operativo standard e a riflettere in questo le chiamate di I/O eseguite

dagli altri. Questo sistema è ancora migliore se si utilizza la paravirtualizzazione, in modo che il comando che viene inviato all'hypervisor comunichi esattamente ciò che occorre al sistema operativo guest (per esempio, leggi il blocco 1403 dal disco 1), invece di avere una serie di comandi che scrivono nei registri del dispositivo, nel qual caso l'hypervisor dovrebbe cercare di capire che cosa fare. Xen utilizza questo approccio per l'I/O; la macchina virtuale che esegue l'I/O si chiama **dominio 0**.

Gli hypervisor di tipo 2 hanno un vantaggio pratico, nella virtualizzazione dell'I/O, rispetto a quelli di tipo 1: il sistema operativo host contiene già i driver per tutti i dispositivi collegati al computer. Quando un programma applicativo tenta di accedere a un dispositivo di I/O, il codice tradotto può chiamare il driver di dispositivo esistente e fargli eseguire il lavoro, mentre con un hypervisor di tipo 1 è l'hypervisor che deve contenere il driver, oppure fare una chiamata a un driver del dominio 0, in qualche modo simile a un sistema operativo host. A mano a mano che la tecnologia delle macchine virtuali va maturando, è probabile che gli applicativi potranno accedere direttamente ai dispositivi in modo sicuro; ciò significa che sarà possibile collegare i driver direttamente al codice dell'applicazione in server in modalità utente separati (come in MINIX3), eliminando in questo modo il problema.

Virtualizzazione dell'I/O a radice singola

Assegnare un dispositivo direttamente a una macchina virtuale non è molto scalabile; con quattro reti fisiche è possibile supportare non più di quattro macchine virtuali. Per otto macchine virtuali diventano necessarie otto schede di rete, mentre per eseguire 128 macchine virtuali, diciamo solo che potrebbe essere difficile ritrovare il computer sotto un mucchio di schede.

È sicuramente possibile la condivisione via software di dispositivi su più hypervisor, ma spesso non è ottimale, perché fra l'hardware e i driver e il sistema operativo guest si interpone un livello di emulazione (o un dominio di dispositivi). Il dispositivo emulato spesso non dispone di tutte le funzioni avanzate di cui dispone l'hardware; l'ideale sarebbe che la tecnologia di virtualizzazione offrisse l'equivalente di un pass through di un singolo dispositivo a tutti gli hypervisor, senza overhead. Virtualizzare un singolo dispositivo per far credere a tutte le macchine virtuali di averne l'accesso esclusivo è molto più facile se della virtualizzazione si occupa l'hardware. Sui PCIe, questa soluzione si chiama virtualizzazione I/O single root.

La **virtualizzazione dell'I/O a radice singola** (Single Root I/O Virtualization o **SR-IOV**) consente di bypassare l'intervento dell'hypervisor nella comunicazione fra driver e dispositivo. I dispositivi che supportano SR-IOV hanno uno spazio di memoria indipendente, interrupt e flussi DMA per ciascuna macchina virtuale che li usano (Intel, 2011). Il dispositivo appare come più dispositivi separati, ciascuno dei quali può essere configurato da una diversa macchina virtuale. Per esempio, ognuno avrà il proprio registro base e il proprio spazio di indirizzamento. Una macchina virtuale può mappare una di queste aree di memoria (utilizzate, per esempio, per configurare il dispositivo) nel proprio spazio di indirizzamento.

SR-IOV ha due modi per consentire l'accesso a un dispositivo: **PF** (**physical functions**, funzioni fisiche) e **VF** (**virtual functions**, funzioni virtuali). Le funzioni fisiche sono funzioni PCIe complete che consentono di configurare il dispositivo in qualsiasi modo desiderato dall'amministratore, ma non sono accessibili ai sistemi operativi guest. Le funzioni virtuali

sono funzioni PCIe leggere che non offrono le medesime opzioni di configurazione e sono ideali per le macchine virtuali. SR-IOV consente di virtualizzare i dispositivi in centinaia di funzioni virtuali che fanno credere alle macchine virtuali di essere le uniche proprietarie di un dispositivo. Per esempio, data un'interfaccia di rete SR-IOV, una macchina virtuale può gestirne la scheda di rete virtuale come fosse fisica. Per di più, molte schede di rete moderne hanno buffer separati (e circolari) per l'invio e la ricezione di dati interamente dedicati a queste macchine virtuali. Per esempio, la serie di schede di rete Intel I350 è dotata di otto code di invio e otto code di ricezione.

7.8 Virtual appliance

Le macchine virtuali offrono un'interessante soluzione a un problema che per molto tempo ha afflitto gli utenti, soprattutto quelli di software open source: come installare nuovi programmi applicativi. Il problema è che molte applicazioni dipendono da numerose altre applicazioni e librerie, a loro volta dipendenti da numerosi altri pacchetti software, e così via. Inoltre, possono esserci molte dipendenze da particolari versioni dei compilatori, linguaggi di scripting e sistema operativo.

Ora, con la tecnologia delle macchine virtuali, uno sviluppatore software può realizzare una macchina virtuale accuratamente progettata, caricarla con il sistema operativo, i compilatori, le librerie e il codice dell'applicazione e creare un blocco unico, pronto per l'esecuzione. Questa immagine di macchina virtuale può quindi essere registrata su CD-ROM oppure caricata su un sito web in modo che i clienti la possano installare o scaricare. Utilizzando questo sistema, solo lo sviluppatore deve comprendere tutte le dipendenze dell'applicazione: i clienti hanno invece un pacchetto completo che funziona, indipendentemente dal sistema operativo che utilizzano e da tutto il software, i pacchetti e le librerie che avevano installato. Spesso queste macchine virtuali preconfezionate vengono chiamate **virtual appliance**. Per fare un esempio, la soluzione cloud EC2 di Amazon ha numerose virtual appliance preimpostate per i propri clienti, offerte come comodi servizi software ("Software as a Service").

7.9 Macchine virtuali su CPU multicore

La combinazione fra macchine virtuali e CPU multicore crea un ambiente del tutto nuovo in cui il numero di CPU a disposizione può essere impostato via software. Se, per esempio, ci sono quattro core e ciascuno di essi può eseguire fino a otto macchine virtuali, una singola CPU può essere configurata come multicomputer a 32 nodi, se necessario; ma può anche impegnare meno CPU, a seconda del software utilizzato. In precedenza non era mai stato possibile, per uno sviluppatore di applicazioni, scegliere dapprima quante CPU utilizzare, per poi scrivere il codice di conseguenza. Questa è veramente una nuova epoca per l'informatica.

Le macchine virtuali, per di più, possono condividere la memoria. Un esempio tipico in cui questo approccio può tornare utile è quello di un singolo server che ospita più istanze dello stesso sistema operativo: è sufficiente mappare le pagine fisiche nello spazio di indirizzi

di più macchine virtuali. La condivisione della memoria è già disponibile nelle soluzioni di deduplicazione. La **deduplicazione** fa esattamente ciò che ci si potrebbe aspettare, evitando di memorizzare due volte (duplicare) gli stessi dati. È una tecnica molto comune nei sistemi di memorizzazione, che viene sempre più spesso utilizzata anche nella virtualizzazione. In Disco, veniva chiamata **condivisione trasparente delle pagine** (e richiede una modifica nel guest), mentre in VMware si chiama **condivisione delle pagine basata sui contenuti** (e non richiede alcuna modifica). In generale, la tecnica richiede di esaminare la memoria di ciascuna delle macchine virtuali su un host e di calcolare l'hash delle pagine di memoria. Se alcune pagine producono lo stesso hash, il sistema deve per prima cosa verificare se si tratta veramente della stessa pagina; nel caso, le deduplica, creando una pagina con il contenuto effettivo della memoria e due riferimenti a detta pagina. Poiché l'hypervisor controlla le tabelle delle pagine nidificate (o shadow), questa mappatura è molto semplice. Ovviamente, quando uno dei guest modifica la pagina condivisa, questa modifica non dev'essere visibile alle altre macchine virtuali; il trucco è utilizzare una strategia **copy-on-write** (copia in caso di scrittura) così che la pagina modificata sia privata per chi ha eseguito l'operazione di scrittura.

Se le macchine virtuali possono condividere la memoria, un singolo computer diventa un multiprocessore virtuale. Dal momento che tutti i core di un chip multicore condividono la medesima RAM, un singolo chip quad-core può essere facilmente configurato come multiprocessore a 32 nodi o multicomputer a 32 nodi, secondo necessità.

La combinazione di multicore, macchine virtuali, hypervisor e microkernel modificherà radicalmente il modo in cui si pensa ai computer. Attualmente il software non riesce a far fronte all'idea di un programmatore che scelga di quante CPU ha bisogno, che si tratti di un multicomputer o di un multiprocessore, e al modo in cui i microkernel di diverso tipo possono far parte dello scenario; il software che verrà sviluppato in futuro dovrà invece tener conto anche di queste questioni. Se siete studenti o professionisti di informatica o ingegneria informatica potreste essere proprio voi a trovare in futuro la soluzione adatta.

7.10 Problemi di licenze

Alcuni software, soprattutto aziendali, hanno licenze basate sul numero di CPU; in altre parole, quando si acquista un programma, si ha il diritto di eseguirlo su una sola CPU. Ma in pratica che cos'è una CPU? Il contratto dà diritto a eseguire il software su più macchine virtuali tutte in esecuzione sulla stessa macchina fisica? Molti produttori di software non sanno bene come comportarsi.

Il problema peggiora ancora per le aziende che hanno licenze che consentono loro di avere n macchine che eseguono il software contemporaneamente, soprattutto quando le macchine virtuali vengono attivate o disattivate a richiesta.

In alcuni casi, i produttori di software hanno inserito nella licenza una clausola esplicita che vieta di eseguire il software su una macchina virtuale o su una macchina virtuale non autorizzata. Per le aziende che eseguono tutto il software esclusivamente su macchine virtuali questo potrebbe rappresentare un problema. Che queste clausole possano reggere in tribunale, e in che modo gli utenti possano rispondere, nel caso, resta tutto da vedere.

7.11 Cloud

La tecnologia di virtualizzazione ha giocato un ruolo cruciale nella crescita del cloud computing. Esistono molti cloud. Alcuni sono pubblici, e tutti coloro che sono disposti a pagare per l'uso delle risorse vi possono accedere; altri sono privati ad uso esclusivo da parte di qualche organizzazione. I diversi cloud offrono poi diversi servizi. Alcuni concedono accesso all'hardware fisico da parte degli utenti, ma la maggior parte si limita a offrire ambienti virtualizzati. Alcuni offrono semplici architetture hardware, virtuali o no, e nient'altro, mentre altri offrono software pronto all'uso e che può essere combinato in modi interessanti, oppure piattaforme che semplificano lo sviluppo di nuovi servizi da parte degli utenti. I provider di cloud solitamente offrono diverse categorie di risorse, come macchine più o meno grandi. Nonostante tutte le chiacchiere che si fanno sul cloud, sembra che siano pochi quelli che capiscono veramente di che cosa si tratti. Il National Institute of Standards and Technology, che è sempre un'ottima risorsa a cui fare riferimento, elenca cinque caratteristiche essenziali.

1. **Self service a richiesta**. Gli utenti devono essere in grado di accedere automaticamente alle risorse, senza intervento umano.
2. **Ampio accesso alla rete**. Tutte le risorse devono essere disponibili sulla rete attraverso meccanismi standard, così che possano essere utilizzate da dispositivi di diverso tipo.
3. **Condivisione delle risorse**. La risorsa elaborativa posseduta dal provider deve servire più utenti e deve avere la capacità di assegnare e riassegnare dinamicamente le risorse. Solitamente gli utenti non conoscono nemmeno la locazione esatta delle "loro" risorse e in che paese si trovano.
4. **Elasticità rapida**. Dev'essere possibile acquisire e rilasciare elasticamente le risorse, magari anche in maniera automatica, per scalare immediatamente secondo le richieste degli utenti.
5. **Servizio misurato**. Il provider del cloud misura le risorse utilizzate in modo che corrisponda al tipo di servizio su cui ci si è accordati.

7.11.1 Cloud come servizio

In questo paragrafo affronteremo il cloud concentrandoci sulla virtualizzazione e sui sistemi operativi. In particolare, considereremo i cloud che offrono accesso diretto a una macchina virtuale e che l'utente può utilizzare nel modo che preferisce. Lo stesso cloud può eseguire diversi sistemi operativi, a volte sullo stesso hardware. Dal punto di vista del cloud, questa modalità operativa è nota come **IAAS** (**infrastructure as a service**, infrastruttura come servizio), in opposizione a **PAAS** (**platform as a service**, piattaforma come servizio, che offre un ambiente comprendente elementi come un sistema operativo, un database, un server Web specifici e così via), **SAAS** (**software as a service**, software come servizio, che offre l'accesso a specifici software, come Microsoft Office 365 o Google Apps) e ad altri prodotti offerti come servizi. Un esempio di cloud IAAS è Amazon EC2, basato sull'hypervisor Xen e che conta svariate centinaia di migliaia di macchine fisiche; se avete il denaro necessario, potete avere tutta la potenza di elaborazione che desiderate.

Il cloud può trasformare il modo in cui le aziende elaborano i propri dati. Per cominciare, consolidando tutte le risorse di elaborazione in pochi luoghi fisici (molto spesso nei pressi di una fonte energetica e di un comodo sistema di raffreddamento) si beneficia dell'economia di scala. Terzializzare le necessità elaborative significa che non bisogna più preoccuparsi di gestire la propria struttura IT, i backup, la manutenzione, il deprezzamento, la scalabilità, l'affidabilità, le prestazioni e perfino la sicurezza: tutto viene eseguito in un unico posto e, se il provider del cloud è competente, a regola d'arte. Si potrebbe pensare che i manager IT siano oggi più soddisfatti di quanto non lo fossero una decina di anni fa; eppure, anche se queste preoccupazioni stanno sparendo, ne sorgono di nuove. Ci si può veramente fidare che il proprio provider di cloud mantenga al sicuro i dati sensibili? Un concorrente che utilizza le medesime infrastrutture potrebbe avere accesso a informazioni che si desiderano mantenere private? Quali leggi si applicano ai propri dati (per esempio, se il provider di cloud è negli Stati Uniti, i dati sono sottoposti al PATRIOT Act, anche se l'azienda è in Europa)? Quando si memorizzano tutti i dati nel cloud X, si sarà in grado di riappropriarsene, oppure si rimarrà per sempre legati a quel particolare cloud e al suo provider, ossia ci si troverà in una situazione nota come **vendor lock-in** (dipendenza dal fornitore)?

7.11.2 Migrazione delle macchine virtuali

La tecnologia della virtualizzazione non soltanto consente ai cloud IAAS di eseguire più sistemi operativi diversi contemporaneamente sul medesimo hardware, ma consente altresì una gestione intelligente. Si è già parlato della possibilità di sovraccaricare le risorse, soprattutto in combinazione con la deduplicazione. Ora affronteremo un altro problema gestionale: che cosa accade se una macchina ha bisogno di manutenzione (o di sostituire un pezzo) mentre sta eseguendo una quantità di macchine virtuali importanti? I clienti non saranno molto contenti di veder crollare i propri sistemi perché il provider del cloud vuole sostituire un disco rigido.

Gli hypervisor separano la macchina virtuale dall'hardware fisico. In altre parole, non è molto importante, per la macchina virtuale, essere eseguita su un computer fisico o su un altro. L'amministratore, pertanto, può semplicemente spegnere tutte le macchine virtuali e riavviarle su un computer completamente nuovo. Questa operazione, purtroppo, causa un downtime piuttosto importante; bisogna pertanto riuscire a spostare la macchina virtuale da un hardware che ha bisogno di manutenzione ad una nuova macchina senza fermarla neanche un momento.

Un approccio lievemente migliore potrebbe mettere in pausa la macchina virtuale, invece di spegnerla; durante la pausa, si copiano più velocemente possibile sul nuovo hardware le pagine di memoria utilizzate dalla macchina virtuale, si configura il tutto correttamente nel nuovo hypervisor, dopodiché si riprende l'esecuzione. Oltre alla memoria, è anche necessario trasferire i supporti di memorizzazione e la connettività di rete, ma se le macchine sono vicine l'operazione può essere svolta rapidamente. Innanzitutto si può fare in modo che il file system sia basato su rete (come NFS, network file system), in modo che non sia importante se la macchina virtuale è in esecuzione nel rack 1 o 3 del server. Ciononostante, è ancora necessario mettere in pausa la macchina per un tempo non indifferente. Forse un tempo inferiore, ma comunque non trascurabile.

Le soluzioni di virtualizzazione moderne, invece, offrono la cosiddetta **migrazione live**; in altre parole, sono in grado di spostare la macchina virtuale mentre è ancora attiva. Per

esempio, adottano tecniche come la **migrazione della memoria prima della copia**, ossia copiano le pagine di memoria mentre la macchina sta ancora elaborando le richieste. La maggior parte delle pagine di memoria non vengono scritte molto spesso, quindi copiarle è un'operazione abbastanza sicura. Si ricordi che la macchina virtuale è ancora in esecuzione, quindi una pagina può essere modificata anche dopo che è stata copiata. Quando una pagina di memoria viene modificata, bisogna fare in modo di copiare sulla destinazione l'ultima versione, quindi la pagina viene contrassegnata come sporca e sarà copiata nuovamente più tardi. Quando la maggior parte delle pagine di memoria sono state copiate, resta un certo numero di pagine sporche, relativamente piccolo; bisogna ora fermare la macchina per qualche istante per copiare le pagine rimanenti, quindi riprendere l'esecuzione della macchina sulla nuova posizione. Una pausa c'è ancora, ma è talmente breve che solitamente le applicazioni non ne vengono influenzate. Quando il downtime non è percepibile, si parla di **migrazione live senza soluzione di continuità**.

7.11.3 Checkpointing

Il fatto di rendere la macchina virtuale indipendente dall'hardware fisico ha altri vantaggi. In particolare, si è detto che è possibile mettere in pausa una macchina virtuale. L'operazione è utile in sé; se lo stato della macchina messa in pausa (stato della CPU, delle pagine di memoria, dei supporti di memorizzazione) viene memorizzato su disco, si ottiene uno snapshot di una macchina in esecuzione. Se il software dovesse creare confusione con le macchine in esecuzione, è comunque possibile tornare allo stato memorizzato nello snapshot e continuare come se nulla fosse accaduto.

Il modo più semplice per creare uno snapshot è copiare tutto, compreso l'intero file system. Copiare un disco di più terabyte, però, può richiedere molto tempo, anche con un disco veloce; ed è auspicabile non mettere in pausa la macchina virtuale per troppo tempo mentre si fa la copia. La soluzione è utilizzare le tecniche **copy-on-write**, in modo che i dati vengano copiati solo quando assolutamente necessario.

L'uso degli snapshot funziona piuttosto bene, ma vi sono dei problemi: che fare se una macchina sta interagendo con un computer remoto? È possibile fare lo snapshot del sistema e riprenderlo più avanti, ma il computer remoto che era prima in comunicazione potrebbe non essere più disponibile. Questo è un problema che non può essere risolto.

7.12 Caso di studio: VMware

Fin dal 1999 VMware, Inc. è stata il più importante produttore commerciale di soluzioni di virtualizzazione con prodotti per desktop, server, il cloud e ora anche i cellulari. Non solo produce hypervisor, ma anche il software che gestisce macchine virtuali su larga scala.

Inizieremo a parlare di questo caso di studio con una breve storia di come è nata l'azienda; descriveremo quindi VMware Workstation, un hypervisor di tipo 2 nonché il primo prodotto dell'azienda, i problemi causati dalla sua struttura e gli elementi più importanti della soluzione. Descriveremo quindi l'evoluzione di VMware Workstation nel corso degli anni, concludendo con una descrizione di ESX Server, l'hypervisor di tipo 1 prodotto da VMware.

7.12.1 Gli inizi di VMware

Anche se l'idea di utilizzare macchine virtuali risale agli anni Sessanta e Settanta, sia nel settore informatico sia nella ricerca accademica, dopo gli anni Ottanta e con l'avvento dei personal computer l'interesse per la virtualizzazione era venuto meno. Solo la divisione mainframe di IBM continuò a occuparsi di virtualizzazione. In effetti, le architetture dei computer progettate all'epoca, in particolare quella di Intel x86, non supportavano la virtualizzazione (ossia, non soddisfacevano i criteri di Popek/Goldberg), il che è un vero peccato, perché le CPU 386, che rappresentavano una completa riprogettazione del vecchio 286, erano state prodotte un decennio dopo l'articolo di Popek-Goldberg e i progettisti avrebbero dovuto saperne di più.

Nel 1997, a Stanford, tre dei futuri fondatori di VMware avevano realizzato un prototipo di hypervisor chiamato Disco (Bugnion et al., 1997), con lo scopo di eseguire sistemi operativi indifferenziati (in particolare UNIX) su un multiprocessore a grandissima scala che era in fase di sviluppo a quell'epoca: la macchina FLASH. Durante il progetto, gli autori compresero che l'uso delle macchine virtuali poteva risolvere, in maniera semplice ed elegante, svariati problemi del software di sistema, anche molto complessi: invece di cercare di risolvere i problemi all'interno di un determinato sistema operativo, era possibile operare delle innovazioni a un livello **sottostante** i sistemi operativi esistenti. L'osservazione alla base della realizzazione di Disco era che, anche se l'elevata complessità dei sistemi operativi moderni rendeva difficile implementare delle innovazioni, la relativa semplicità di un monitor di macchina virtuale e la sua posizione sulla pila del software era un ottimo punto di partenza per risolvere le limitazioni dei sistemi operativi. Anche se Disco era pensato per server molto grandi, e progettato per l'architettura MIPS, gli autori compresero che si poteva adottare lo stesso approccio anche per il mercato degli x86, facendolo diventare commercialmente interessante.

Nel 1998 venne così fondata VMware, con l'obiettivo di fare in modo che anche l'architettura x86 e il settore dei personal computer potessero godere dei benefici della virtualizzazione. Il primo prodotto di VMware (VMware Workstation) fu la prima soluzione di virtualizzazione disponibile per le piattaforme x86 a 32 bit. Il prodotto fu rilasciato per la prima volta nel 1999, in due varianti: **VMware Workstation per Linux**, un hypervisor di tipo 2 che veniva eseguito su un sistema operativo host Linux, e **VMware Workstation per Windows**, che veniva eseguito in Windows NT. Entrambe le varianti avevano le stesse funzionalità: gli utenti potevano creare più macchine virtuali specificando prima le caratteristiche dell'hardware virtuale (per esempio quanta memoria assegnare alla macchina virtuale, oppure le dimensioni del disco virtuale), quindi installando il sistema operativo desiderato all'interno della macchina virtuale, solitamente da CD-ROM (anche virtuale).

VMware Workstation era pensato soprattutto per gli sviluppatori e i professionisti IT. Prima dell'introduzione della virtualizzazione, era normale che uno sviluppatore avesse sulla scrivania un paio di computer, uno stabile per lo sviluppo e un altro su cui poteva reinstallare il sistema operativo tutte le volte che gli serviva. Con la virtualizzazione, il secondo computer per i test diveniva una macchina virtuale.

Ben presto VMware iniziò la produzione di un secondo prodotto, più complesso, che sarebbe stato rilasciato con il nome di ESX Server nel 2001. ESX Server sfruttava lo stesso motore di virtualizzazione di VMware Workstation, ma era pensato come hypervisor di tipo 1; in altre parole, ESX Server veniva eseguito direttamente sull'hardware, senza la necessità di un sistema operativo host. L'hypervisor di ESX era stato progettato per consolidare un

carico di lavoro intenso e conteneva molte ottimizzazioni per assicurarsi che tutte le risorse (CPU, memoria e I/O) venissero allocate fra le macchine virtuali in maniera efficiente. Per esempio, fu il primo hypervisor a introdurre il concetto di ballooning per bilanciare la memoria fra le varie macchine virtuali (Waldspurger, 2002).

ESX Server era pensato per il mercato del consolidamento dei server. Prima dell'introduzione della virtualizzazione, per gli amministratori IT era normale acquistare, installare e configurare un nuovo server per ogni nuovo compito o applicazione che bisognava eseguire presso il data center; il risultato era che l'infrastruttura veniva utilizzata in maniera molto inefficiente: i server venivano all'epoca utilizzati solo al 10% della loro capacità, durante i picchi. Con ESX Server, gli amministratori IT potevano ammassare molte macchine virtuali indipendenti in un singolo server, risparmiando tempo, denaro, spazio nei rack ed energia elettrica.

Nel 2002 VMware presentò la prima soluzione di gestione per ESX Server, chiamata in origine Virtual Center e oggi vSphere. La soluzione forniva un unico punto di gestione per un intero cluster di server che eseguivano macchine virtuali: un amministratore IT poteva semplicemente entrare nell'applicazione Virtual Center e controllare, monitorare e creare migliaia di macchine virtuali in esecuzione in tutta l'azienda. Con Virtual Center arrivò un'altra importante innovazione: **VMotion** (Nelson et al., 2005), che consentiva la migrazione live sulla rete di una macchina virtuale in esecuzione. Per la prima volta un amministratore IT poteva spostare un computer in esecuzione da un posto all'altro senza dover riavviare il sistema operativo, rilanciare le applicazioni e nemmeno perdere le connessioni di rete.

7.12.2 VMware Workstation

VMware Workstation fu il primo prodotto di virtualizzazione per i computer con architettura x86 a 32 bit. L'adozione della virtualizzazione ebbe un profondo impatto sul settore e sull'intera comunità informatica: nel 2009, ACM assegnò ai suoi autori l'**ACM Software System Award** per VMware Workstation 1.0 per Linux. L'originale VMware Workstation è descritto in un dettagliato articolo tecnico (Bugnion et al., 2012), del quale riportiamo un riassunto.

L'idea considerava che un livello di virtualizzazione potesse tornare utile su piattaforme costruite sulle CPU x86 e che eseguivano soprattutto il sistema operativo Microsoft Windows (ossia la piattaforma **WinTel**). I benefici della virtualizzazione potevano aiutare a risolvere alcune delle limitazioni note della piattaforma WinTel, come l'interoperabilità delle applicazioni, la migrazione del sistema operativo, l'affidabilità e la sicurezza; inoltre, la virtualizzazione poteva facilmente consentire la coesistenza di sistemi operativi alternativi, in particolare Linux.

Nonostante ci fossero stati decenni di ricerche e sviluppo commerciale della tecnologia di virtualizzazione sui mainframe, l'ambiente x86 era così diverso che occorrevano nuovi approcci. Per esempio, i mainframe erano **integrati verticalmente**, ossia hardware, hypervisor, sistemi operativi e la maggior parte delle applicazioni erano realizzati da un singolo produttore.

Il mondo x86 era invece (ed è ancora) suddiviso in almeno quattro diverse categorie: (a) Intel e AMD producono i processori; (b) Microsoft offre Windows, mentre il mondo open source offre Linux; (c) un terzo gruppo di aziende realizza dispositivi e periferiche di I/O, con i relativi driver; (d) un quarto gruppo di integratori come HP e Dell assembla i

sistemi per la vendita. Per la piattaforma x86, bisogna per prima cosa creare una virtualizzazione che non necessiti del supporto da parte di questi produttori.

Poiché questa suddivisione era comunque presente nella vita reale, VMware Workstation era diversa dai classici monitor per macchine virtuali, pensati per gestire architetture realizzate da un solo produttore e che disponevano di un supporto esplicito alla virtualizzazione. VMware Workstation era invece realizzato per l'architettura x86 e tutto l'indotto che si portava dietro. A questi problemi VMware Workstation rispondeva combinando in una singola soluzione tecniche di virtualizzazione ben note, tecniche prese da altri domini e tecniche completamente nuove.

Ora vedremo gli specifici problemi tecnici che sono stati affrontati nella realizzazione di VMware Workstation.

7.12.3 Problemi di virtualizzazione in ambiente x86

Riprendiamo la definizione di hypervisor e macchina virtuale: gli hypervisor applicano il principio ben noto di **aggiungere un livello di indirezione** nel campo dell'hardware dei computer e forniscono l'astrazione per le **macchine virtuali**: più copie dell'hardware sottostante, ciascuna delle quali esegue un'istanza indipendente di un sistema operativo. Le macchine virtuali sono isolate le une dalle altre, appaiono ciascuna come un duplicato dell'hardware sottostante e idealmente possono essere eseguite alla stessa velocità della macchina reale. VMware ha adattato le caratteristiche fondamentali di una macchina virtuale a una piattaforma x86 nel modo seguente.

1. **Compatibilità**. Il concetto di un "ambiente sostanzialmente identico" significava che qualsiasi sistema operativo x86 e tutte le relative applicazioni dovevano poter essere eseguiti senza modifiche come macchine virtuali. Un hypervisor doveva fornire una compatibilità a livello hardware sufficiente perché gli utenti potessero eseguire qualsiasi sistema operativo (fino a livello di aggiornamento e patch) che desideravano installare in una determinata macchina virtuale, senza restrizioni.
2. **Prestazioni**. Il sovraccarico dell'hypervisor doveva essere sufficientemente basso da consentire agli utenti l'utilizzo di una macchina virtuale come ambiente di lavoro principale. L'obiettivo dei progettisti di VMware era di poter eseguire carichi di lavoro rilevanti a velocità prossime a quella nativa e, nel peggiore dei casi, di eseguirle sui processori attuali con le stesse prestazioni che avrebbero avuto se fossero state eseguite nativamente sui processori della generazione appena precedente. Questo approccio era basato sul fatto che la maggior parte del software per x86 non era stato progettato per essere eseguito solamente sulle ultimissime generazioni di CPU.
3. **Isolamento**. Un hypervisor doveva garantire l'isolamento della macchina virtuale senza fare alcuna illazione sul software che sarebbe stato eseguito al suo interno. Ossia, l'hypervisor doveva mantenere il controllo completo delle risorse; al software in esecuzione all'interno delle macchine virtuali doveva assolutamente essere impedito qualsiasi accesso che potesse far perdere il controllo all'hypervisor. Allo stesso modo, l'hypervisor doveva assicurare la privacy di tutti i dati non appartenenti alla macchina virtuale. Un hypervisor doveva tenere in considerazione che un sistema operativo guest poteva essere infettato da codice malevolo sconosciuto (preoccupazione molto più viva oggi che durante l'era dei mainframe).

Chiaramente, fra queste tre necessità si creava un'inevitabile tensione. Per esempio, la totale compatibilità in certe aree potrebbe portare a un impatto proibitivo sulle prestazioni, nel qual caso i progettisti di VMware dovevano scendere a compromessi, stando bene attenti, però, a evitare qualsiasi compromesso che potesse mettere a rischio l'isolamento o esporre l'hypervisor ad attacchi provenienti da un guest malevolo. Divennero quindi evidenti quattro problemi principali.

1. **L'architettura x86 non era virtualizzabile**. Conteneva istruzioni sensibili alla virtualizzazione, non privilegiate, che violavano i criteri di Popek e Goldberg per la virtualizzazione rigorosa. Per esempio, l'istruzione `POPF` ha una semantica diversa (anche se non causa trap) a seconda che il software in esecuzione possa o non possa disabilitare gli interrupt, il che impediva di adottare il classico approccio alla virtualizzazione di tipo trap-and-emulate. Anche gli ingegneri di Intel erano convinti che i loro processori non potevano, all'atto pratico, essere virtualizzati.
2. **L'architettura x86 era di una complessità spaventosa**. L'architettura x86 era un'architettura CISC notoriamente molto complicata, che garantiva il supporto a decenni di compatibilità verso il basso. Nel corso degli anni, erano state introdotte quattro modalità operative (reale, protetta, v8086 e gestione di sistema), ciascuna delle quali abilitava diversamente il modello di segmentazione dell'hardware, i meccanismi di paginazione e le funzionalità di sicurezza (come i call gate).
3. **Le macchine x86 avevano periferiche di ogni tipo**. Anche se c'erano solo due produttori principali di x86, i personali computer dell'epoca potevano avere collegata un'enorme varietà di schede di espansione e di dispositivi, ciascuno dei quali con il proprio driver fornito dal produttore. Virtualizzare tutte queste periferiche era semplicemente impossibile, il che portava a due implicazioni: occorreva affrontare questo problema sia sul front end (l'hardware virtuale esposto nelle macchine virtuali) sia sul back end (l'hardware reale che l'hypervisor doveva essere in grado di controllare) delle periferiche.
4. **Occorreva un'esperienza utente semplice**. I classici hypervisor erano installati in fabbrica, un po' come il firmware che si trova nei computer odierni. Poiché VMware era una startup, gli utenti avrebbero dovuto installare gli hypervisor su macchine già esistenti. VMware aveva pertanto bisogno di una procedura di installazione semplice, per incoraggiare gli utenti a farne uso.

7.12.4 VMware Workstation: le soluzioni

In questo paragrafo descriveremo ad alto livello in che modo VMware Workstation ha affrontato e risolve i problemi riportati nel paragrafo precedente.

VMware Workstation è un hypervisor di tipo 2 che consiste di moduli separati. Un importante modulo è il VMM, responsabile dell'esecuzione delle istruzioni della macchina virtuale; un secondo modulo importante è il VMX, che interagisce con il sistema operativo host. Nel corso del paragrafo vedremo innanzitutto in che modo il VMM risolve il problema della non virtualizzabilità dell'architettura x86, quindi descriveremo la strategia centrata sul sistema operativo utilizzata dai progettisti durante tutta la fase di sviluppo. Descriveremo poi come è stata progettata la piattaforma hardware virtuale, che affronta una metà del

problema della diversità delle periferiche, per poi terminare parlando del ruolo del sistema operativo host in VMware Workstation, in particolare l'interazione fra i componenti VMM e VMX.

Virtualizzazione dell'architettura x86

Il VMM esegue la macchina virtuale vera e propria, consentendole di procedere. Un VMM realizzato per un'architettura virtualizzabile utilizza, per eseguire direttamente sull'hardware in modo sicuro le istruzioni della macchina virtuale, una tecnica nota come trap-and-emulate. Quando questo non è possibile, un possibile approccio è definire un sottoinsieme di istruzioni virtualizzabili dell'architettura del processore e portare il sistema operativo guest nella nuova piattaforma che si viene così a definire. Questa tecnica è nota come paravirtualizzazione (Barham et al., 2003; Whitaker et al., 2002) e richiede una modifica del sistema operativo a livello di codice sorgente. Detta in parole povere, la paravirtualizzazione modifica il guest per evitare che possa fare qualsiasi cosa che non sia gestibile dall'hypervisor. La paravirtualizzazione non era possibile in VMware, a causa del requisito della compatibilità e della necessità di eseguire sistemi operativi il cui codice sorgente non era disponibile, in particolare Windows.

Un metodo alternativo avrebbe potuto adottare un'emulazione completa. In questo approccio le istruzioni delle macchine virtuali vengono emulate dal VMM sull'hardware, invece che eseguite direttamente. Il metodo può essere molto efficiente; precedenti esperienze con il simulatore di macchine SimOS (Rosenblum et al., 1997) avevano dimostrato che l'impiego di tecniche come la **traduzione dinamica delle librerie** in esecuzione su un programma a livello utente riusciva a limitare il sovraccarico di un'emulazione completa, portando a rallentamenti di un fattore cinque. Anche se è *abbastanza* efficiente, e sicuramente utile per le simulazioni, un rallentamento di un fattore cinque è evidentemente inadeguato e non soddisfa i requisiti di prestazione.

La soluzione di questo problema combina due idee fondamentali. Innanzitutto, anche se non si può sempre utilizzare l'esecuzione diretta trap-and-emulate per tutta l'architettura x86, la si può utilizzare almeno in parte; soprattutto, la si può usare durante l'esecuzione dei programmi applicativi, che, in caso di carichi di lavoro intensi, sono quelli che occupano la maggior parte del tempo. Il motivo è che queste istruzioni sensibili alla virtualizzazione non sono sempre sensibili, ma lo diventano solamente in determinate circostanze. Per esempio, l'istruzione `POPF` è sensibile alla virtualizzazione quando il software può disabilitare gli interrupt (per esempio quando esegue il sistema operativo), mentre non è sensibile quando il software non può disattivare gli interrupt (in pratica, quando si eseguono quasi tutte le applicazioni a livello utente).

La Figura 7.8 mostra i blocchi modulari del VMM originale di VMware; si può notare come consista di un sottosistema che esegue direttamente le istruzioni, di un sottosistema di traduzione binaria e di un algoritmo di decisione che stabilisce quale sottosistema utilizzare. Entrambi i sottosistemi si basano su alcuni moduli condivisi, per esempio per virtualizzare la memoria attraverso le tabelle delle pagine shadow o per emulare i dispositivi di I/O.

Il sottosistema che esegue direttamente le istruzioni è quello che si usa di preferenza, mentre quello di traduzione binaria dinamica entra in funzione quando non è possibile l'esecuzione diretta. Questo è il caso, per esempio, di quando la macchina virtuale si trova in uno stato che potrebbe richiamare un'istruzione sensibile alla virtualizzazione; ogni sistema

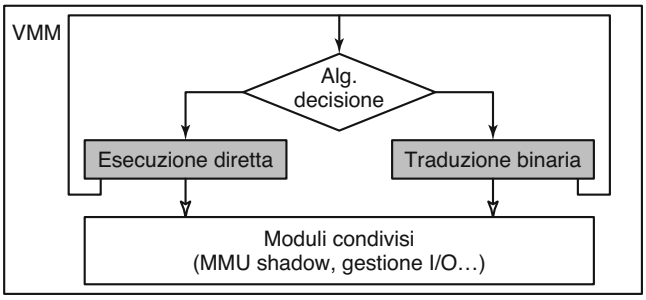

Figura 7.8. Componenti di alto livello del monitor di macchine virtuali VMware (in assenza di supporto hardware).

calcola pertanto costantemente l'algoritmo di decisione per determinare se un passaggio al sottosistema è possibile (dalla traduzione binaria all'esecuzione diretta) oppure è necessario (dall'esecuzione diretta alla traduzione binaria). Questo algoritmo ha svariati parametri in ingresso, come l'attuale anello di esecuzione della macchina virtuale, se a quel livello sia possibile oppure no attivare gli interrupt e lo stato dei segmenti. Per esempio, è necessario utilizzare la traduzione binaria se è vera una qualsiasi delle condizioni sotto riportate.

1. La macchina virtuale è attualmente in esecuzione in modalità kernel (anello 0 dell'architettura x86).
2. La macchina virtuale può disattivare gli interrupt ed emettere istruzioni di I/O (nell'architettura x86, quando il livello dei privilegi di I/O è impostato sul livello dell'anello 0).
3. La macchina virtuale è in esecuzione in modalità reale, una modalità di esecuzione legacy a 16 bit utilizzata fra le altre cose dal BIOS.

L'effettivo algoritmo decisionale contiene alcune altre condizioni, i particolari delle quali sono riportati in Bugnion et al. (2012). È interessante notare che l'algoritmo non dipende dalle istruzioni attualmente presenti in memoria e pronte all'esecuzione, ma soltanto dal valore assunto da alcuni registri virtuali e può pertanto essere calcolato in maniera estremamente efficiente con una manciata di istruzioni.

La seconda idea fondamentale è che, configurando opportunamente l'hardware, e soprattutto impiegando in maniera attenta il meccanismo della protezione dei segmenti dell'x86, anche il codice di sistema che viene sottoposto a traduzione binaria dinamica può essere eseguito quasi a velocità nativa; si tratta di un risultato molto diverso dal rallentamento di un fattore cinque che ci si potrebbe normalmente aspettare dai simulatori.

La differenza può essere spiegata confrontando il modo in cui un traduttore binario converte una semplice istruzione che accede alla memoria. Per emulare un'istruzione di questo tipo via software, un classico traduttore binario che emula l'architettura x86 con l'intero set di istruzioni dovrebbe per prima cosa verificare se l'effettivo indirizzo si trova all'interno del range dei segmenti dati, quindi converte l'indirizzo in un indirizzo fisico e finalmente copia la parola a cui si fa riferimento nel registro simulato. Ovviamente, i diversi

passaggi possono essere ottimizzati con una cache, in modo simile a quello in cui il processore utilizza una cache per le mappature delle tabelle delle pagine in un TLB (translation lookaside buffer); anche queste ottimizzazioni, però, comportano che una singola istruzione diventa una sequenza di istruzioni.

Il traduttore binario di VMware non esegue *alcuno* di questi passaggi a livello software, ma configura l'hardware in modo che una semplice istruzione possa essere sostituita da un'istruzione identica. Questo è possibile solo perché il VMM di VMware (di cui il traduttore binario è un componente) ha in precedenza configurato l'hardware in modo che corrisponda alle esatte specifiche della macchina virtuale: (a) il VMM utilizza tabelle delle pagine shadow, assicurandosi che la MMU possa essere utilizzata direttamente (invece che essere emulata) e (b) il VMM utilizza un sistema di shadowing molto simile per le tabelle contenenti i descrittori dei segmenti (che avevano un ruolo fondamentale nel software a 16 e a 32 bit in esecuzione sui vecchi sistemi operativi x86).

Ovviamente ci sono complicazioni e questioni particolari; un aspetto importante del design è assicurare l'integrità della sandbox di virtualizzazione, ossia assicurarsi che nessun software in esecuzione all'interno della macchina virtuale (compreso quello malevolo) possa andare a toccare il VMM. Il problema è generalmente noto come **software fault isolation** (isolamento dalle condizioni di malfunzionamento del software) e aggiunge overhead in fase di esecuzione a ogni accesso alla memoria se la soluzione viene implementata via software. Anche in questo caso, il VMM di VMware utilizza un approccio diverso, basato sull'hardware, suddividendo lo spazio di indirizzi in due zone distinte. Il VMM si riserva i 4 MB superiori di questo spazio di indirizzi, il che libera tutto lo spazio restante (ossia 4 GB – 4 MB, dal momento che l'architettura è a 32 bit) per la macchina virtuale. Il VMM configura quindi l'hardware di segmentazione in modo che nessuna istruzione della macchina virtuale (comprese quelle generate dal traduttore binario) possa mai accedere alla regione di 4 MB dello spazio di indirizzamento.

Una strategia centrata sul sistema operativo guest

Idealmente un VMM dovrebbe essere progettato senza tenere conto del sistema operativo guest in esecuzione nella macchina virtuale, né di come il sistema operativo guest intende configurare l'hardware. L'idea alla base della virtualizzazione è fare in modo che l'interfaccia della macchina virtuale sia identica all'interfaccia hardware, in modo che tutto il software in esecuzione sull'hardware possa essere eseguito allo stesso modo anche nella macchina virtuale. Purtroppo questo approccio è valido soltanto quando l'architettura è virtualizzabile e semplice; nel caso dell'x86, la spaventosa complessità del software rappresentava ovviamente un problema.

I progettisti di VMware semplificarono questo problema concentrandosi esclusivamente sul supporto ad alcuni sistemi operativi selezionati. Alla prima release, VMware Workstation supportava ufficialmente come sistemi operativi guest soltanto Linux, Windows 3.1, Windows 95/98 e Windows NT. Nel corso degli anni, furono poi aggiunti all'elenco altri sistemi operativi, a ogni nuova revisione del software. L'emulazione, in ogni caso, era abbastanza buona da poter eseguire perfettamente anche sistemi operativi che non erano previsti, come MINIX 3, senza ulteriori interventi.

Questa semplificazione non aveva cambiato la struttura generale del progetto (era comunque il VMM a fornire una copia fedele dell'hardware sottostante), ma aiutò a portare

avanti il processo di sviluppo. In particolare, i progettisti non si dovevano preoccupare della combinazione di caratteristiche che venivano utilizzate nella pratica dai vari sistemi operativi guest supportati.

Per esempio, l'architettura x86 contiene quattro anelli di privilegi in modalità protetta (dall'anello 0 all'anello 3), ma nessun sistema operativo utilizza in realtà l'anello 1 e l'anello 2 (a parte OS/2, un sistema operativo di IBM ormai da lunga pezza abbandonato). Invece di cercare di trovare un modo per virtualizzare correttamente l'anello 1 e l'anello 2, il VMM di VMware conteneva semplicemente un codice che verificava se un guest stesse cercando di accedere all'anello 1 o all'anello 2 e, in quel caso, bloccava l'esecuzione della macchina virtuale. Questo modo di procedere non soltanto eliminava codice inutile, ma, cosa più importante, consentiva al VMM di VMware di dare per scontato che la macchia virtuale non avrebbe mai utilizzato gli anelli 1 e 2, e pertanto poteva utilizzare questi anelli per i propri scopi. In effetti, il traduttore binario del VMM di VMware è eseguito sull'anello 1 per virtualizzare il codice dell'anello 0.

La piattaforma hardware virtuale

Fino a questo momento si è parlato principalmente del problema rappresentato dalla virtualizzazione del processore x86, ma un computer basato su x86 contiene molto più del processore: è dotato anche di chipset, firmware e un insieme di periferiche di I/O per controllare dischi, schede di rete, CD-ROM, tastiera e così via.

La diversità fra le periferiche dei personal computer x86 rese di fatto impossibile stabilire una corrispondenza fra l'hardware virtuale e l'hardware reale sottostante. Se da un lato sul mercato esistevano pochi modelli di processore x86, con piccole variazioni nelle capacità a livello di set di istruzioni, esistevano invece migliaia di dispositivi di I/O, la maggior parte dei quali priva di una documentazione accessibile che ne illustrasse interfaccia o funzionalità. L'idea di partenza era di **non** tentare di mettere l'hardware virtuale in corrispondenza con un hardware sottostante specifico, ma di fare in modo che questo fosse messo in corrispondenza con alcune configurazioni composte da una serie ben selezionata di dispositivi di I/O canonici. I sistemi operativi guest potevano quindi utilizzare i propri meccanismi esistenti già installati per rilevare e mettere in funzione questi dispositivi virtuali.

La piattaforma di virtualizzazione consisteva di una combinazione di elementi in multiplexing ed emulati. Multiplexing significa configurare l'hardware in modo che potesse essere utilizzato dalla macchina virtuale e condiviso (nello spazio o nel tempo) fra più macchine virtuali; emulazione significa esportare alla macchina virtuale una simulazione software di componenti hardware selezionati. La Figura 7.9 mostra come VMware Workstation utilizzava il multiplexing per il processore e la memoria e l'emulazione per tutto il resto.

Per l'hardware in multiplexing, ogni macchina virtuale pensava di avere una CPU dedicata e configurabile, con una certa quantità di RAM contigua a partire dall'indirizzo 0.

Dal punto di vista dell'architettura, l'emulazione di ciascun dispositivo virtuale era suddivisa in componente front-end, visibile alla macchina virtuale, e componente back-end, che interagiva con il sistema operativo host (Waldspurger e Rosenblum, 2012). Il front-end era sostanzialmente un modello software del dispositivo hardware che poteva essere controllato da driver non modificati in esecuzione all'interno della macchina virtuale; quale che fosse l'hardware fisico corrispondente sull'host, il front-end esponeva sempre lo stesso modello di dispositivo.

Per esempio, il primo front-end per un dispositivo Ethernet fu il chip AMD PCnet "Lance", una scheda da 10 Mbps all'epoca molto diffusa sui PC, e il back-end forniva la connettività di rete alla rete fisica dell'host. Ironia della sorte, VMware continuò a supportare il dispositivo PCnet anche molto tempo dopo che le schede fisiche Lance non erano più commercialmente disponibili, raggiungendo comunque velocità di I/O di ordini di grandezza più veloci dei 10 Mpbs (Sugermann et al., 2001). Per i dispositivi di memorizzazione, i front-end originali erano un controller IDE e un controller Buslogic, mentre il back-end era tipicamente o un file sul sistema operativo host, come un disco virtuale o un'immagine ISO 9660, oppure una risorsa raw, come una partizione del disco o il lettore CD-ROM fisico.

	Hardware virtuale (front end)	**Back-end**
In multiplexing	1 CPU x86 virtuale, con le stesse estensioni del set di istruzioni della CPU hardware sottostante	Schedulato dal sistema operativo host su un host monoprocessore o multiprocessore
	Fino a 512 MB di DRAM contigua	Allocata e gestita dal sistema operativo host (pagina per pagina)

	Bus PCI	**Bus PCI compatibile completamente emulato**
Emulato	4x dischi IDE 7x dischi Buslogic SCSI	Dischi virtuali (memorizzati come file) o accesso diretto a un determinato dispositivo raw
	1x CD-ROM IDE	Immagine ISO o accesso emulato a un lettore CD-ROM fisico
	2x unità floppy da 1.44 MB	Floppy fisico o immagine floppy
	1x scheda grafica VMware con supporto a VGA e SVGA	Eseguito in finestra o a tutto schermo. La modalità SVGA richiedeva il driver guest VMware per SVGA
	2x porte seriali COM1 e COM2	Connessione a una porta seriale sull'host o a un file
	1x stampante (LPT)	Può connettersi alla porta LPT dell'host
	1x tastiera (104 tasti)	Completamente emulata; gli eventi di tastiera vengono generati quando sono ricevuti dall'applicazione WMware
	1x mouse PS-2	Come la tastiera
	3x schede Ethernet AMD Lance	Modalità bridge e solo host
	1x Soundblaster	Completamente emulata

Figura 7.9 Opzioni di configurazione dell'hardware virtuale delle prime versioni di VMware Worktstation, anno 2000 circa.

Separare i front-end dai back-end aveva un altro vantaggio: una macchina virtuale VMware poteva essere copiata da un computer a un altro, anche dotato di dispositivi hardware differenti senza bisogno di installare nuovi driver, perché interagiva esclusivamente con il componente front-end. Questa caratteristica, chiamata **incapsulamento indipendente dall'hardware**, oggi ha un vantaggio enorme negli ambienti server e nel cloud computing e consente numerose innovazioni, come la possibilità di mettere in pausa e riavviare, la creazione di checkpoint e la migrazione trasparente di macchine virtuali live indipendentemente dai confini fisici (Nelson et al., 2005). Nel cloud, consente ai clienti di utilizzare le proprie macchine virtuali su qualsiasi server disponibile, senza doversi preoccupare dei dettagli dell'hardware sottostante.

Il ruolo del sistema operativo host

L'ultima decisione critica sulla struttura di VMware Workstation fu di installarlo "al di sopra" di un sistema operativo esistente, il che lo fa entrare nella categoria degli hypervisor di tipo 2. La scelta presentava due grossi vantaggi.

Innanzitutto, avrebbe risolto la seconda parte del problema rappresentato dalla diversità delle periferiche. VMware implementò l'emulazione front-end dei diversi dispositivi, basandosi però sui driver del sistema operativo host per il back-end. Per esempio, VMware Workstation avrebbe letto o scritto un file sul sistema operativo host per emulare un disco virtuale, o avrebbe disegnato in una finestra sul desktop dell'host per emulare una scheda video. Purché il sistema operativo host avesse i driver necessari, VMware Workstation poteva eseguirvi sopra delle macchine virtuali.

Secondo, il prodotto poteva essere installato e percepito dall'utente come una normale applicazione, il che ne semplificava l'adozione. Come qualsiasi altra applicazione, l'installatore di VMware Workstation scrive semplicemente i file che compongono il programma nel file system dell'host, senza toccare la configurazione hardware (nessuna necessità di riformattare il disco, di creare una partizione o di modificare le impostazioni del BIOS). In effetti, VMware Workstation poteva essere installato ed eseguire macchine virtuali senza nemmeno riavviare il sistema operativo host, quanto meno sugli host Linux.

Una normale applicazione, però, non possiede i collegamenti e le API necessari a un hypervisor per eseguire il multiplexing della CPU e delle risorse di memoria, che sono essenziali se si vuole garantire una velocità operativa vicina a quella nativa. In particolare, il nucleo della tecnologia di virtualizzazione dell'x86 descritta poc'anzi funziona solamente quando il VMM è in esecuzione in modalità utente e può controllare tutti gli aspetti del processore senza alcuna restrizione, compresa la capacità di modificare lo spazio di indirizzamento (per creare tabelle delle pagine shadow), di modificare le tabelle dei segmenti e di modificare tutti i gestori degli interrupt e delle eccezioni.

Un driver ha un accesso all'hardware più diretto, in particolare se viene eseguito in modalità utente. Anche se in teoria può dare qualsiasi istruzione privilegiata, in pratica un driver dovrebbe interagire con il sistema operativo utilizzando API ben definite e non deve (e non dovrebbe mai) riconfigurare l'hardware in maniera arbitraria; poiché gli hypervisor richiedono una riconfigurazione massiccia dell'hardware (compresi l'intero spazio di indirizzamento, le tabelle dei segmenti, i gestori di eccezioni e di interrupt), l'esecuzione di un hypervisor come driver di periferica non era un'alternativa percorribile.

Poiché nessuna di queste necessità è supportata dai sistemi operativi host, anche l'esecuzione dell'hypervisor come driver (in modalità kernel) non era un'alternativa possibile.

Questi requisiti così stretti portarono allo sviluppo di VMware Hosted Architecture, in cui, come si può vedere dalla Figura 7.10, il software è suddiviso in tre componenti separati e distinti.

Questi componenti hanno ognuno la propria funzione e operano in maniera indipendente l'uno dall'altro.

1. Un programma nello spazio utente (il **VMX**) che l'utente percepisce come l'effettivo programma di VMware. Il VMX esegue tutte le funzioni dell'interfaccia utente, avvia la macchina virtuale e quindi esegue la maggior parte delle emulazioni dei dispositivi (front-end), eseguendo regolarmente chiamate di sistema al sistema operativo host per interagire con il back-end. Solitamente, per ogni singola macchina virtuale è in esecuzione un singolo processo VMX multithread.
2. Un piccolo driver in modalità kernel (il **driver VMX**), che viene installato all'interno del sistema operativo host ed è utilizzato principalmente per consentire l'esecuzione del VMM interrompendo temporaneamente tutto il sistema operativo host. Nel sistema operativo host è installato un unico driver VMX, che viene solitamente lanciato all'avvio.
3. Il VMM, che comprende tutto il software necessario per il multiplexing della CPU e della memoria, compresi i gestori delle eccezioni, quelli per il trap-and-emulate, il traduttore binario e il modulo per la paginazione shadow. Il VMM viene eseguito in modalità kernel, ma non nel contesto del sistema operativo host; in altre parole, non si può basare direttamente sui servizi offerti dal sistema operativo host, ma d'altro canto

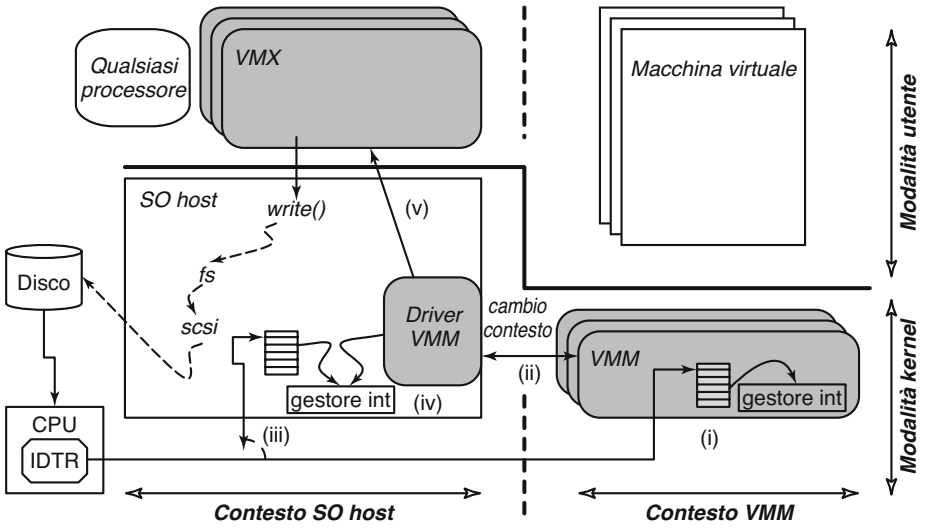

Figura 7.10. VMware Hosted Architecture e i suoi tre componenti: VMX, driver VMM e VMM.

non è nemmeno legato ad alcuna regola o convenzione propria del sistema operativo host. Esiste un'istanza del VMM per ciascuna macchina virtuale, creata all'avvio della stessa.

VMware Workstation sembra essere eseguito al di sopra di un sistema operativo esistente e, in effetti, il suo VMX è eseguito come processo del sistema operativo; tuttavia, il VMM opera a livello di sistema e ha il controllo completo dell'hardware, senza dipendere in alcun modo dal sistema operativo host. La Figura 7.10 mostra la relazione fra due entità: i due contesti (sistema operativo host e VMM) sono paritari e ciascuno ha un componente a livello utente e un componente kernel. Quando si esegue il VMM (la parte destra della figura), questo riconfigura l'hardware, gestisce tutti gli interrupt e le eccezioni di I/O e può pertanto rimuovere temporaneamente il sistema operativo host dalla memoria virtuale, in maniera sicura: per esempio, la posizione della tabella degli interrupt è impostata all'interno del VMM assegnando a un nuovo indirizzo il registro IDTR. Al contrario, quando è in esecuzione il sistema operativo host (la parte sinistra della figura), il VMM e la macchina virtuale vengono eliminati dalla sua memoria virtuale.

La transizione fra questi due contesti indipendenti a livello di sistema viene chiamata **commutazione di mondo** (world switch). Il nome sottolinea come, durante questa operazione, cambia assolutamente tutto a livello di software, a differenza di quanto accade durante una commutazione di contesto eseguita dal sistema operativo. La Figura 7.11 mostra la differenza fra le due operazioni. La normale commutazione di contesto fra i due processi "A" e "B" commuta la parte utente dello spazio di indirizzi e i registri dei due processi, senza però toccare in alcun modo alcune risorse di sistema critiche. Per esempio, la parte kernel dello spazio di indirizzi resta identica per tutti i processi, così come non vengono modificati i gestori delle eccezioni. Al contrario, una commutazione di mondo cambia tutto: tutto lo spazio di indirizzamento, tutti i gestori delle eccezioni, i registri privilegiati, eccetera. In particolare, lo spazio di indirizzamento del kernel del sistema operativo host viene mappato soltanto quando viene eseguito nel contesto del sistema operativo host; dopo la commutazione di mondo nel contesto del VMM, viene totalmente rimosso dallo spazio di indirizza-

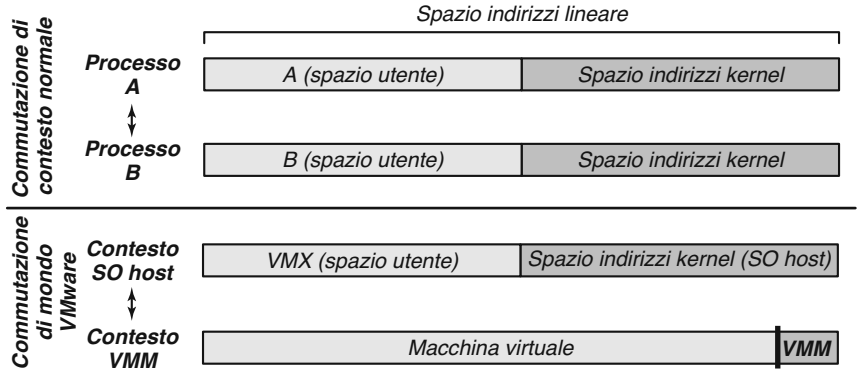

Figura 7.11. Differenza fra una normale commutazione di contesto e una commutazione di mondo.

mento, liberando spazio per eseguire sia il VMM sia la macchina virtuale. Anche se sembra complicato, implementare questa operazione è molto efficiente e, sull'architettura x86, richiede solamente 45 istruzioni in linguaggio macchina.

Il lettore attento si sarà domandato: che cosa accade dello spazio di indirizzamento del kernel di sistema? La risposta è semplicemente che fa parte dello spazio di indirizzamento della macchina virtuale ed è presente quando è eseguito nel contesto del VMM; pertanto, il sistema operativo guest può utilizzare l'intero spazio di indirizzamento del sistema operativo host, in particolare le medesime locazioni della memoria virtuale. Questo è, in particolare, ciò che accade quando i sistemi operativi host e guest sono identici (per esempio sono entrambi Linux). Ovviamente, il tutto "funziona" perché i due contesti sono indipendenti e c'è una commutazione di mondo fra i due.

Lo stesso lettore, poi, si chiederà: e l'area del VMM, al di sopra dello spazio degli indirizzi? Come si è visto in precedenza, quest'area è riservata al VMM e le porzioni dello spazio degli indirizzi non possono essere utilizzate direttamente dalla macchina virtuale. Per fortuna, quella piccola parte di 4 MB non viene utilizzata spesso dal sistema operativo guest, poiché ogni accesso dev'essere individualmente emulato e produce un significativo overhead nel software.

Tornando alla Figura 7.10, si può notare come illustri meglio i diversi passaggi che si verificano quando si esegue il VMM (passaggio i). Ovviamente, il VMM non è in grado di gestire l'interrupt, poiché non ha il driver del back-end. In (ii), il VMM esegue una commutazione di mondo, tornando al sistema operativo host. In particolare, il codice della commutazione di mondo restituisce il controllo al driver VMware, che in (iii) emula lo stesso interrupt inviato dal disco. Nel passaggio (iv), pertanto, il gestore degli interrupt del sistema operativo host esegue la propria logica, come se si fosse verificato un interrupt mentre era in esecuzione il driver VMware (ma non il VMM). Infine, al passaggio (v) il driver VMware restituisce il controllo all'applicazione VMX. A questo punto, il sistema operativo host può scegliere di schedulare un altro processo, oppure di continuare a eseguire il processo VMX di VMware. Se questo continua l'esecuzione, riprende l'esecuzione della macchina virtuale eseguendo una chiamata speciale al driver, che genera una commutazione di mondo per tornare al contesto del VMM. Il trucco funziona e nasconde il VMM e la macchina virtuale al sistema operativo host; in più, cosa ancora più importante, fornisce al VMM la completa libertà di riprogrammare l'hardware come meglio crede.

7.12.5 Evoluzione di VMware Workstation

Il panorama tecnologico è cambiato profondamente nel decennio seguito allo sviluppo del primo VMM di VMware.

È ancora in uso l'architettura hosted per gli hypervisor più avanzati come VMware Workstation, VMware Player e VMware Fusion (il prodotto per i sistemi operativi host Apple OS X), e anche nel prodotto di VMware per i telefoni cellulari (Barr et al., 2010). La commutazione di mondo e la sua capacità di separare il contesto del sistema operativo host da quello del VMM è ancor oggi il meccanismo fondamentale dei prodotti hosted di VMware. Anche se l'implementazione della commutazione di mondo si è evoluta nel corso degli anni, per esempio per supportare i sistemi a 64 bit, l'idea che ne sta alla base, di avere spazi di indirizzamento completamente separati per sistema operativo host e VMM, resta ancora valida.

L'approccio alla virtualizzazione dell'architettura x86, invece, è cambiato radicalmente con l'introduzione della virtualizzazione assistita dall'hardware. Le virtualizzazioni assistite dall'hardware, come Intel VT-x e AMD-v, sono state introdotte in due fasi. La prima fase, iniziata nel 2005, era stata pensata con lo scopo esplicito di eliminare la necessità della paravirtualizzazione o della traduzione binaria (Uhlig et al., 2005). A partire dal 2007, con la seconda fase venne garantito il supporto hardware nella MMU sotto forma di tabelle delle pagine nidificate, eliminando in questo modo la necessità di mantenere nel software tabelle delle pagine shadow. Oggi gli hypervisor di VMware utilizzano per lo più un approccio basato su hardware di tipo trap-and-emulate (come formalizzato da Popek e Goldberg una quarantina d'anni prima) se il processore supporta sia la virtualizzazione sia le tabelle delle pagine nidificate.

La nascita del supporto hardware alla virtualizzazione ha avuto un impatto molto significativo sulla strategia di VMware centrata sul sistema operativo guest. Nella prima VMware Workstation, si era impiegata questa strategia per ridurre al minimo la complessità di implementazione, a scapito della compatibilità con tutta l'architettura; oggi ci si aspetta di avere una completa compatibilità con l'architettura, proprio grazie alla presenza del supporto hardware. La strategia VMware centrata sul sistema operativo guest serve oggi per migliorare le prestazioni sul sistema guest in cui è in esecuzione.

7.12.6 ESX Server, l'hypervisor di tipo 1 di VMware

Nel 2001, VMware rilasciò un altro prodotto, chiamato ESX Server, pensato per il mercato dei server e nel quale i progettisti di VMware seguirono un approccio del tutto diverso: invece di creare una soluzione di tipo 2 in esecuzione su un sistema operativo host, decisero di realizzare una soluzione di tipo 1 che doveva essere eseguita direttamente sull'hardware.

La Figura 7.12 mostra l'architettura ad alto livello di ESX Server, che combina un componente esistente, il VMM, con un vero hypervisor in esecuzione direttamente sull'hardware della macchina. Il VMM ha la stessa funzione di VMware Workstation, ossia eseguire la macchina virtuale in un ambiente isolato che è un duplicato dell'architettura x86. Di fatto, i VMM utilizzati nei due prodotti impiegano lo stesso codice sorgente di base e sono in gran parte identici. L'hypervisor ESX sostituisce il sistema operativo host, ma invece di implementare tutta la funzionalità che ci si aspetta da un sistema operativo il suo unico scopo è eseguire le diverse istanze del VMM e di gestire con efficienza le risorse fisiche della macchina. ESX Server contiene quindi il consueto sottosistema che si trova in un sistema operativo, come lo scheduler della CPU, il gestore della memoria e il sottosistema di I/O, con ogni sottosistema ottimizzato per eseguire le macchine virtuali.

L'assenza di un sistema operativo host obbligò VMware a risolvere direttamente i problemi di diversità delle periferiche e di esperienza utente descritti in precedenza. Per la diversità delle periferiche, VMware risolse il problema facendo in modo di obbligare ESX Server a essere eseguito solamente su piattaforme server certificate per le quali era presente un driver. Per quanto invece riguarda l'esperienza utente, a differenza di VMware Workstation, ESX Server richiedeva agli utenti di installare una nuova immagine del sistema su una partizione di avvio.

Pur tenendo conto degli svantaggi, i vantaggi erano tali che valeva la pena installare una virtualizzazione dedicata nei data center, consistenti di centinaia o migliaia di server e spesso di (molte) migliaia di macchine virtuali. A volte oggi si parla di cloud privati in rife-

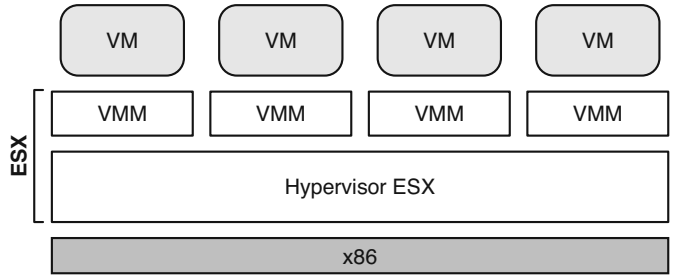

Figura 7.12. ESX Server, l'hypervisor di tipo 1 di VMware.

rimento a questo tipo di installazione. L'architettura ESX Server garantisce vantaggi evidenti in termini di prestazioni, scalabilità, possibilità di gestione e funzionalità; eccone alcuni.

1. Lo scheduler della CPU garantisce che ogni macchina virtuale ottenga una fetta consistente della CPU, per evitare la starvation; inoltre è progettato in modo tale che le diverse CPU virtuali di una determinata macchina virtuale multiprocessore siano schedulate contemporaneamente.
2. Il gestore della memoria è ottimizzato per la scalabilità, in particolare per eseguire con efficienza le macchine virtuali anche quando necessitano di più memoria di quella effettivamente disponibile sul computer. Per ottenere questo risultato, in ESX Server è stato per la prima volta introdotto il concetto di ballooning e di condivisione trasparente delle pagine per le macchine virtuale (Waldspurger, 2002).
3. Il sottosistema di I/O è ottimizzato per le prestazioni. Anche se VMware Workstation ed ESX Server spesso hanno gli stessi componenti di emulazione nel front-end, i back-end sono completamente diversi. Nel caso di VMware Workstation, tutto l'I/O scorre attraverso il sistema operativo host e le sue API, il che spesso aggiunge overhead, soprattutto nel caso di dispositivi di rete e memorizzazione. Con ESX Server, questi driver vengono eseguiti direttamente all'interno dell'hypervisor ESX, senza richiedere una commutazione di mondo.
4. I back-end tipicamente si basavano su astrazioni fornite dal sistema operativo host; per esempio, VNware Workstation memorizza le macchine virtuali come normali file (benché di grandi dimensioni) sul file system host. ESX Server, invece, è dotato di VMFS (Vaghani, 2010), un file system ottimizzato specificamente per memorizzare le immagini delle macchine virtuali e garantire un throughput elevato di I/O, arrivando a livelli di prestazioni estremamente elevati. Per esempio, VMware ha dimostrato, nel 2011, che un singolo ESX Server può eseguire un milione di operazioni su disco al secondo.
5. Con ESX Server l'introduzione di nuove funzionalità, che in precedenza avrebbero richiesto il coordinamento e la configurazione di molti componenti di un computer, è stata semplificata. Per esempio, con ESX Server è stata introdotta VMotion, la prima soluzione di virtualizzazione che poteva migrare una macchina virtuale live in esecuzione da una macchina con ESX Server in un'altra macchina con ESX Server. Questa

operazione richiedeva il coordinamento fra gestore di memoria, scheduler della CPU e stack di rete.

Nel corso degli anni, a ESX Server sono state aggiunte nuove funzioni; ESX Server si è evoluto fino a diventare ESXi, un'alternativa leggera di dimensioni sufficientemente ridotte da poter essere installata nel firmware dei server. Oggi ESXi è il prodotto di punta di VMware e serve come fondamento della suite vSphere.

7.13 Ricerche sulla virtualizzazione e sul cloud

La tecnologia della virtualizzazione e il cloud computing sono campi in cui la ricerca è estremamente attiva; le ricerche prodotte sono veramente troppe per poter essere elencate. Per entrambi gli argomenti esistono numerose conferenze di ricerca. Per esempio, la conferenza Virtual Execution Evironments (VEE) si concentra sulla virtualizzazione nel suo senso più ampio: vi si trovano articoli sulla deduplicazione nella migrazione, sulla riduzione di scala e così via. ACM Symposium on Cloud Computing (SOCC) è un altro ottimo esempio di congresso, questa volta sul cloud computing. Fra gli atti del SOCC si possono trovare lavori sulla resilienza agli errori, sullo scheduling dei carichi di lavoro nei data center, sulla gestione e sul debug nel cloud e così via.

Anche argomenti all'apparenza obsoleti sono ancora interessanti, come nel caso di Penneman et al. (2013), che esamina i problemi della virtualizzazione dei processori ARM alla luce dei criteri di Popek e Goldberg. La sicurezza è un argomento sempre molto caldo (Beham et al., 2013; Mao, 2013; e Pearce et al., 2013), così come il risparmio energetico (Botero and Hesselbach, 2013; e Yuan et al., 2013). Con così tanti data center che utilizzano oggi la tecnologia della virtualizzazione, la connessione in rete di queste macchine è un altro importante oggetto di ricerca (Theodorou et al., 2013). Anche la virtualizzazione in ambienti di rete wireless è un argomento vitale e interessante (Wang et al., 2013a).

Un campo interessante che ha visto numerose ricerche è la virtualizzazione nidificata (Ben-Yehuda et al., 2010; e Zhang et al., 2011). L'idea è che una macchina virtuale potrebbe essere a sua volta virtualizzata in diverse macchine virtuali di alto livello, ciascuna delle quali potrebbe essere a sua volta virtualizzata, e così via. Uno dei progetti è chiamato, molto appropriatamente, "Turtles" (tartarughe), perché, una volta che si è partiti, come dice il proverbio, "Ogni tartaruga poggia su un'altra tartaruga".

Uno dei grandi vantaggi dell'hardware di virtualizzazione è che il codice non sicuro può ottenere accesso diretto a caratteristiche hardware come tabelle delle pagine e TLB etichettati, ma sempre in maniera sicura. Tenendo conto di questo fatto, il progetto Dune (Belay, 2012) non cerca di ottenere un'astrazione di macchina, bensì fornisce un'astrazione *di processo*. Il processo è in grado di entrare in modalità Dune, una transizione irreversibile che dà accesso all'hardware a basso livello, pur rimanendo un processo, in grado di comunicare con il kernel e di basarsi su di esso. L'unica differenza è che, per svolgere una chiamata di sistema, usa l'istruzione `VMCALL`.