TECNICHE DI VIRTUALIZZAZIONE
TEORIA E PRATICA

Maurelio Boari
Simone Balboni

Negli ultimi anni l'interesse per il settore delle tecnologie delle macchine virtuali è cresciuto notevolmente. Si sono diffusi nuovi sistemi operativi per la virtualizzazione e si stanno sviluppando progetti per rendere più semplice ed efficiente il loro utilizzo. Un campo di applicazione interessante è la riorganizzazione di server farm di grandi dimensioni per rendere più efficiente l'uso delle risorse, semplificarne la gestione e aumentare la sicurezza. Questo articolo descrive le proprietà delle tecnologie di virtualizzazione e ne presenta l'utilizzo in un caso reale complesso.

3.2

1. INTRODUZIONE

Il concetto di virtualizzazione è da tempo ampiamente utilizzato in vari settori della computer science, dalla progettazione di sistemi software complessi (esempio, sistemi operativi), ai linguaggi di programmazione, alle architetture dei processori e alla trasmissione dei dati. Da un punto di vista generale le tecnologie di virtualizzazione puntano a disaccoppiare il funzionamento logico delle risorse hardware e software di un sistema di elaborazione dalla loro realizzazione fisica, con l'obiettivo di ottenere maggiore efficienza, affidabilità e sicurezza. Il disaccoppiamento è ottenuto introducendo tra le due viste della risorsa, la logica e la fisica, un livello di indirezione la cui realizzazione dipende dal tipo di virtualizzazione che si intende adottare.
Un primo esempio di virtualizzazione coincide con il concetto di astrazione. In questo caso l'obiettivo è semplificare l'uso di una risorsa nascondendo alcuni aspetti di dettaglio relativi alla sua realizzazione. Si parla in questo caso di risorsa virtuale (oggetto astratto) e il livello di indirezione introdotto è costituito dalle operazioni (interfaccia) con le quali è possibile accedere alle risorse.
Questo concetto di virtualizzazione viene normalmente applicato nella progettazione di sistemi di elaborazione complessi, che vengono organizzati come un insieme di livelli di astrazione strutturati gerarchicamente [9]. Al livello più esterno le applicazioni dispongono di una macchina virtuale il cui set di istruzioni è composto dalle istruzioni non privilegiate della macchina fisica e da un insieme di nuove istruzioni rappresentate dalle funzioni fornite dal S.O. (system call), mediante le quali è possibile accedere alle risorse del sistema in modo semplice e sicuro.
Il sistema di elaborazione è visto come un insieme di macchine virtuali, una per ogni processo attivo, che utilizzano tutte lo stesso livello di disaccoppiamento dalla macchina fisica rappresentato dall'interfaccia fornita dal S.O. Esse dipendono quindi dal S.O. di cui utilizzano le system call e dall'hardware sul quale sono eseguite.
Un caso diverso di macchina virtuale si presenta quando il livello di disaccoppiamento

dalla macchina fisica è rappresentato dal codice generato da un compilatore di un linguaggio del tipo HLL. Tale codice, chiamato codice astratto, risulta completamente indipendente dal set di istruzioni della macchina fisica e dalle system call del S.O. che in essa opera. Si parla in questo caso di macchina virtuale a livello di linguaggio. L'obiettivo di questa VM è permettere la portabilità dello stesso codice astratto su molteplici piattaforme (hardware e S.O.) diverse. Ciascuna di queste realizza una VM capace di caricare ed eseguire il codice astratto e un insieme di librerie specifiche. Nella sua forma più semplice, la VM contiene un interprete e, nei casi più sofisticati, un compilatore, che partendo dal codice astratto genera codice per la macchina fisica sulla quale la VM è in esecuzione. Un esempio ben noto di tale paradigma è la *Java Virtual Machine* (JVM) [6]. Lo sforzo di implementare per le architetture più comuni una VM capace di caricare ed eseguire il codice macchina astratto è ben ripagato dalla piena portabilità del software attraverso le piattaforme.
Un diverso utilizzo del concetto di virtualizzazione, prevede l'introduzione di un livello di indirezione, che si chiama *Virtual Machine Monitor* (VMM) o hypervisor il cui compito è quello di trasformare la singola interfaccia di macchina fisica in *N* interfacce virtuali. Ciascuna di queste interfacce (*macchine virtuali*) è una replica della macchina fisica dotata quindi di tutte le istruzioni del processore (sia privilegiate che non privilegiate) e delle risorse del sistema (memoria, dispositivi di I/O). Su ogni macchina virtuale può essere eseguito un sistema operativo. Compito del VMM è quindi consentire la condivisione da parte di più macchine virtuali di una singola piattaforma hardware. Esso si pone come mediatore unico nelle interazioni tra le macchine virtuali e l'hardware sottostante, garantendo un forte isolamento tra loro e la stabilità complessiva del sistema [5, 13].
Un primo esempio di tale tipo di architettura è quello introdotto da IBM negli anni '60 col sistema di elaborazione denominato originariamente CP/CMS e successivamente VM/370 [8]. Il CP (*Control Program*) svolge le funzioni del VMM, viene eseguito sulla macchina fisica ed ha il solo compito di creare più interfacce della stessa, senza fornire alcun servizio all'utente. Ciascuna di queste interfacce (macchine virtuali) è una replica del semplice hardware.
Il CMS (*Conversational Monitor System*) è il sistema operativo, interattivo e monoutente, che gira su ogni macchina virtuale.
Il CP/CMS è nato dal lavoro svolto presso il Centro Scientifico di IBM a Cambridge [3] a metà degli anni '60 con l'obiettivo di creare un sistema time-sharing. La sua adozione, nella versione VM/370, da parte di IBM fu una diretta conseguenza del sostanziale fallimento del sistema time-sharing TSS/360 costruito per il modello 67 del 360 che si rivelò non adeguato alle aspettative perché troppo complesso e poco efficiente.
L'idea architetturale delle macchine virtuali, propria del nuovo sistema, consentiva ad ogni utente, tramite il CP, di avere la propria macchina virtuale con la propria partizione sul disco e di supportare lo sviluppo dei suoi programmi in tale macchina virtuale tramite il CMS.
Tutta l'architettura risultava più semplice da gestire rispetto ad un tradizionale sistema time-sharing in quanto risultavano separate, e quindi sviluppabili indipendentemente, le due parti di suddivisione dell'uso delle risorse fisiche tra gli utenti e di mascheramento per l'utente delle peculiarità dell'hardware.
Inoltre, poiché ogni macchina virtuale era funzionalmente identica all'hardware della macchina fisica, era possibile mettere in esecuzione su di esse qualunque sistema operativo compatibile con l'hardware stesso e diverse macchine virtuali potevano eseguire differenti sistemi operativi. Nelle versioni successive del VM/370 furono messi in esecuzione sulle macchine virtuali diversi sistemi operativi, quali IBM OS/360 e DOS/360. Un altro aspetto di interesse era l'elevata affidabilità del sistema dal momento che la struttura del VMM consentiva l'esecuzione separata delle macchine virtuali garantendo quindi che un errore in un sistema operativo non avesse influenza sull'esecuzione degli altri S.O.
La diffusione in quegli anni di questo tipo di architettura portò anche alla progettazione di architetture hardware pensate per consentire in modo efficiente di soddisfare queste esigenze di virtualizzazione [19].
A partire dagli anni '70 si sono diffusi i mo-

derni sistemi operativi multitasking e contemporaneamente si è assistito ad un crollo nel costo dell'hardware. I mainframe hanno progressivamente lasciato il posto ai minicomputer e ad un paradigma del tipo "un server per ogni applicazione", e così lo sviluppo dei VMM si è interrotto al punto che le architetture non hanno più fornito l'hardware necessario per implementarli in modo efficiente. Questa tendenza ha però portato nei primi anni del 2000 ad una proliferazione dell'hardware all'interno delle sale server, con tanti minicomputer piuttosto sottoutilizzati, considerato il costante aumento della potenza di calcolo, con grossi problemi legati agli spazi, al condizionamento degli ambienti, elevati costi di alimentazione e di gestione. Per questi motivi nella seconda metà degli anni '90 sono ricomparsi sul mercato nuovi *Virtual Machine Monitor* compatibili con l'architettura più diffusa entro le sale server, ovvero IA-32, in un'ottica di consolidamento di tanti server sottoutilizzati su di un singolo calcolatore fisico. Questi sistemi saranno l'oggetto del prossimo paragrafo.
Per concludere, si può ricordare che un caso particolare di virtualizzazione si presenta quando si richiede l'esecuzione di programmi compilati per un certo insieme di istruzioni su un sistema di elaborazione dotato di un diverso insieme di istruzioni. Si parla in questo caso di emulazione.

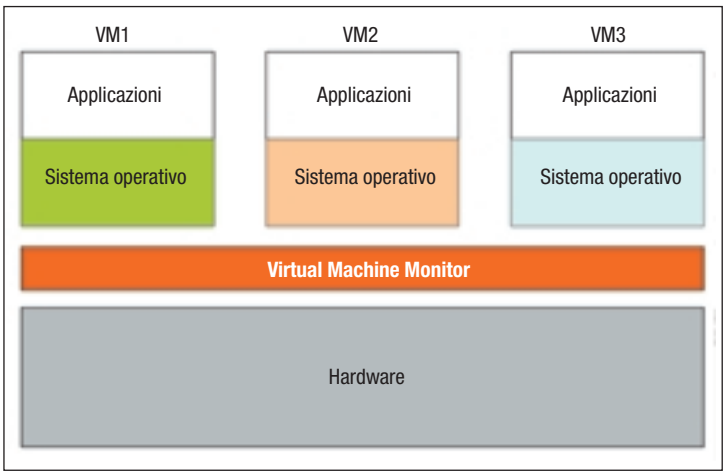

FIGURA 1

Schema logico di un ambiente virtualizzato: in questo esempio il VMM è posto direttamente sopra l'hardware, ed espone interfacce hardware virtuali funzionalmente identiche ad ognuna delle macchine virtuali in esecuzione sopra di esso

Il modo più diretto per emulare è interpretare: un programma interprete legge, decodifica ed esegue ogni singola istruzione utilizzando il nuovo set di istruzioni. È un metodo molto generale e potente, ma produce un sovraccarico mediamente molto elevato poiché possono essere necessarie decine di istruzioni dell'host per interpretare una singola istruzione sorgente.
Una famosa implementazione, molto cara agli appassionati di giochi, è MAME [15], una VM per host IA-32 in grado di caricare ed eseguire il codice binario originale delle ROM dei videogiochi da bar degli anni '80, emulando l'hardware tipico di quelle architetture. Facendo leva sull'enorme potenza di calcolo degli attuali PC rispetto ai primordiali processori per videogiochi dell'epoca, la VM può tranquillamente operare secondo il paradigma dell'interpretazione senza compromettere in modo significativo l'esperienza di gioco, almeno per i videogiochi più vecchi che non facevano ancora un uso massiccio della grafica. Sono già oltre 1900 le immagini di ROM che si possono eseguire con MAME su di un comune PC.
Un ulteriore esempio è costituito dalla classe dei microprocessori che, al fine di ridurre i consumi energetici, implementano internamente un'architettura a parole molto lunghe (VLIW); all'esterno invece, grazie ad un emulatore integrato in hardware, espongono una classica architettura IA-32, potendo così eseguire i più noti sistemi operativi e relative applicazioni. L'esempio più noto è il processore Transmeta Crusoe [21], tra i cui progettisti vi è anche Linus Torvalds.

2. REALIZZAZIONE DEL VMM

Come si è visto nel paragrafo precedente, compito del VMM è consentire la condivisione, da parte di più macchine virtuali, di una singola piattaforma hardware. Il VMM espone ad ogni macchina virtuale un insieme di interfacce hardware funzionalmente equivalenti a quelle di una macchina fisica e si pone come mediatore unico nelle interazioni tra le macchine virtuali e l'hardware sottostante, garantendo un forte isolamento tra esse e la stabilità complessiva del sistema (Figura 1).

Al di là dei modi diversi in cui si può progettare un VMM, esso deve comunque soddisfare poche condizioni essenziali: fornire un ambiente per i programmi sostanzialmente identico a quello della macchina reale; garantire una elevata efficienza nella esecuzione dei programmi; possedere caratteristiche di elevata semplicità. Il primo obiettivo richiede che qualsiasi programma eseguito all'interno di una macchina virtuale generi lo stesso risultato che si otterrebbe se il programma fosse eseguito direttamente sulla macchina reale. Le uniche differenze possono essere legate alle dipendenze temporali imposte dalla presenza delle altre macchine virtuali concorrenti e dalla disponibilità di risorse di sistema. Il secondo obiettivo, l'efficienza, richiede che l'overhead imposto dalla virtualizzazione sia comunque tale da offrire all'utente l'illusione di operare sulla macchina reale. Per ottenerlo occorre che un sottoinsieme statisticamente dominante delle istruzioni del processore virtuale siano eseguite direttamente – senza la mediazione del VMM – sul processore reale. Questo approccio, noto come *esecuzione diretta*, è centrale per potere realizzare efficacemente la virtualizzazione. In pratica esso prevede che le istruzioni *non* privilegiate, che sono la frazione più consistente di un *instruction set*, quelle da cui non può derivare l'eventuale blocco del sistema, siano eseguite direttamente in hardware senza coinvolgere in alcun modo il VMM. Quest'ultimo interviene invece solo nell'esecuzione delle istruzioni privilegiate, minimizzando così il sovraccarico. Infine, il requisito della semplicità nella realizzazione è essenziale per garantire la stabilità e la sicurezza dell'intero sistema, minimizzando l'occorrenza di malfunzionamenti che comprometterebbero l'esistenza delle macchine virtuali. In altre parole è necessario che il VMM, nonostante la disponibilità delle istruzioni privilegiate per le macchine virtuali, resti sempre nel pieno controllo delle risorse hardware e che non sia mai possibile ai programmi in esecuzione negli ambienti virtuali accedere all'hardware in modo privilegiato scavalcando il controllo del VMM.
Vi sono diversi modi per realizzare un VMM con queste proprietà; le differenze fondamentali tra le implementazioni più diffuse si possono ricondurre a due fattori discriminanti: i principi che governano il dialogo tra la macchina virtuale ed il VMM, ed il livello dove si intende collocare il VMM rispetto all'architettura del sistema di elaborazione. Rispetto alla prima scelta, che è la più importante in termini di metodo, si distingue tra i paradigmi di *virtualizzazione completa e paravirtualizzazione*. Rispetto alla seconda scelta, si distingue tra VMM posti direttamente sopra l'hardware dell'elaboratore (*VMM di sistema*) che integrano un sistema operativo "leggero" con le funzionalità di virtualizzazione e VMM che si istallano invece come applicazioni dentro un sistema operativo preesistente (*VMM ospitati*). Normalmente viene indicato con il termine *host* la piattaforma di base sulla quale si realizzano macchine virtuali, che comprende la macchina fisica, l'eventuale sistema operativo e il VMM; si indica invece con il termine *guest* tutto ciò (applicazioni e sistema operativo) che ha a che fare con la macchina virtuale.
Analizziamo ora i dettagli e le differenze delle diverse tecniche di realizzazione (Figura 2).
a. *Virtualizzazione completa*. Il paradigma della virtualizzazione completa prevede che l'hardware virtuale esposto dal VMM sia funzionalmente identico a quello della sottostante macchina fisica. In questo modo è possibile installare dentro le macchine virtuali sistemi operativi standard, senza che abbiano subito alcuna modifica specifica per eseguire in ambiente virtuale. Questo approccio semplifica notevolmente la creazione e gestione dell'ambiente guest, ma rende un po' più complesso il disegno del VMM; inoltre vedremo che per una efficace implementazione è richiesta la collaborazione dell'architettura della CPU. Negli approfondimenti che seguono supponiamo, per semplicità, che il VMM sia installato direttamente sull'hardware del calcolatore.
Un'architettura CPU in generale opera secondo livelli (ring) di protezione: per semplicità consideriamo due soli livelli, supervisore ed utente, anche se molte architetture, tra cui IA-32, implementano livelli intermedi usati ad esempio per l'I/O. Il livello supervisore è riservato al software che deve accedere alle risor-

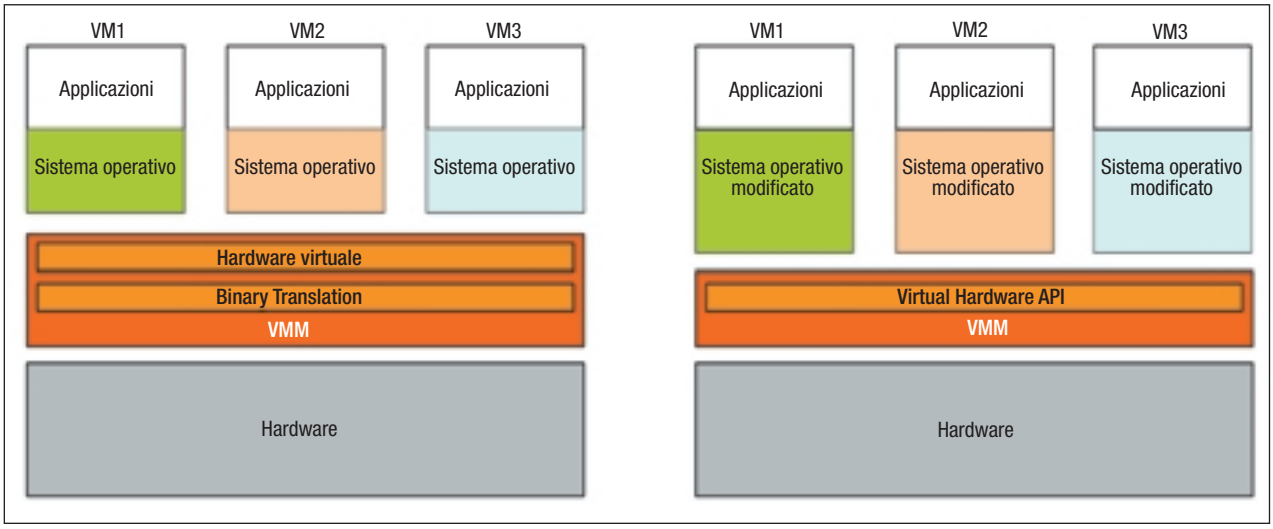

FIGURA 2

Confronto tra virtualizzazione completa (sinistra) e paravirtualizzazione (destra), su architetture IA-32. Un VMM per la virtualizzazione completa replica per ogni macchina virtuale le medesime interfacce hardware, funzionalmente identiche a quelle della sottostante macchina fisica; per questo i sistemi operativi guest non devono essere modificati. Un VMM in paravirtualizzazione espone invece una API cui i sistemi operativi guest devono interfacciarsi per accedere alle risorse

se del sistema con il massimo privilegio (sistema operativo, driver), e in tale stato si possono eseguire tutte le istruzioni proprie dell'architettura, tra cui anche le istruzioni privilegiate che danno pieno accesso alle risorse hardware e ai registri del sistema. Il livello utente è riservato al software meno privilegiato (applicazioni), e in tale stato non è possibile accedere alle istruzioni privilegiate della CPU. Se dallo stato utente si invocano istruzioni privilegiate si attiva il meccanismo di protezione della CPU che non esegue l'istruzione in questione ma notifica allo stato supervisore, mediante una eccezione (trap), la richiesta ricevuta e gli passa il controllo. Normalmente, alcune delle componenti di un sistema operativo (il kernel ed i driver) si aspettano di essere eseguite nello stato supervisore, poiché devono controllare l'hardware. In un contesto di virtualizzazione tuttavia, è solamente il VMM ad occupare lo stato supervisore, mentre tutti i software guest che vi girano sopra (applicazioni, ma anche sistemi operativi) sono spinti più in alto, nel livello utente, con i privilegi di semplici applicazioni. Vi sono dunque due ordini di problemi nella gestione di un sistema operativo guest che non si intende modificare ai fini della virtualizzazione: esso è chiamato ad operare in uno stato che non gli è proprio, poichè le chiamate di accesso alle risorse gli sono inibite (problema di ring deprivileging), inoltre esso è schiacciato nello stato utente insieme alle semplici applicazioni, con il problema di doversi proteggere da queste (problema di ring compression).
Il VMM deve dunque mascherare ai sistemi operativi guest la natura dello stato in cui sono eseguiti, facendosi carico di intercettare ogni richiesta di accesso privilegiato all'hardware ed *emulandone* il comportamento. Si tratta infatti di richieste che non possono essere eseguite direttamente, ma non possono nemmeno essere ignorate pena il malfunzionamento o il blocco del sistema guest, di cui si interromperebbe il normale flusso operazionale. Per intercettare tali chiamate il VMM è aiutato in modo determinante dalle funzionalità di protezione dell'architettura CPU: quando l'ambiente guest tenta di eseguire un'istruzione privilegiata, la CPU notifica un'eccezione al VMM (posto nello stato supervisore) e gli trasferisce il controllo. Il VMM verifica la correttezza dell'operazione richiesta e provvede ad emularne il comportamento atteso. Se per esempio, un guest tenta di eseguire l'istruzione privilegiata che disabilita gli interrupt, il VMM riceve la notifica di tale richiesta e ne emula il comportamento atteso, cioè sospende la consegna degli interrupt solamente a quella macchina

virtuale. Così facendo, la macchina virtuale prosegue secondo il normale flusso operazionale che seguirebbe in un ambiente reale ed il sistema rimane complessivamente stabile; se invece la richiesta della macchina virtuale fosse eseguita sul processore, sarebbero disabilitati tutti gli interrupt per tutti i sistemi e questo impedirebbe al VMM di riguadagnare il controllo della CPU.

Il meccanismo di notifica della CPU aiuta a mantenere piuttosto semplice il disegno del VMM, che in modo trasparente è chiamato ad intervenire solamente per mediare l'accesso alle risorse hardware, di cui per altro mantiene sempre il controllo. La soluzione è anche efficiente in quanto consente che tutte le istruzioni non privilegiate siano eseguite direttamente dall'hardware, senza alcuna supervisione da parte del VMM che non sarebbe di alcuna utilità ed introdurrebbe solo latenza.

Un'architettura CPU si dice "*naturalmente virtualizzabile*" se supporta l'invio di notifica allo stato supervisore *per ogni* istruzione privilegiata eseguita dallo stato utente. Un'architettura di questo tipo favorisce un disegno semplice del VMM e supporta nativamente la tecnica dell'esecuzione diretta, fondamentale per garantire prestazioni.

È necessario osservare tuttavia che non tutte le architetture sono naturalmente virtualizzabili e nel novero di queste vi è anche l'architettura IA-32, che pure oggi è al centro della rinascita della virtualizzazione: realizzata nell'epoca del boom del personal computer, non è stata affatto progettata tenendo presente le condizioni per una semplice virtualizzazione. Alcune istruzioni privilegiate di questa architettura se eseguite nello spazio utente *non* provocano un'interruzione da parte del meccanismo di protezione della CPU ma vengono semplicemente ignorate e non consentono quindi un trasparente intervento del VMM. Inoltre, vi sono istruzioni non privilegiate, dunque consentite liberamente anche nello spazio utente, che permettono di accedere in lettura ad alcuni registri di sistema le cui informazioni andrebbero però mascherate ad una macchina virtuale e la cui gestione dovrebbe essere riservata solo al VMM (problema di *ring aliasing*). Vi è per esempio un registro (code segment register) che segnala il livello di privilegio corrente e la cui lettura da parte di un sistema operativo guest – che come tale è eseguito nello spazio utente, che non gli sarebbe proprio – segnalerebbe al medesimo un'anomalia di collocazione.

Questa assenza di supporto da parte dell'hardware impone al VMM di implementare stratagemmi di varia natura per garantire il funzionamento della virtualizzazione completa; i problemi possono essere risolti, ma al prezzo di un aumento di complessità ed una riduzione delle prestazioni. Una soluzione tipica prevede che il VMM scansioni il codice prima di passarlo in esecuzione per impedire che blocchi contenenti queste istruzioni siano eseguiti direttamente. VMware [8], per esempio, implementa la tecnica della *fast binary translation* [16] per sostituire i blocchi contenenti simili istruzioni problematiche in blocchi equivalenti dal punto di vista funzionale e contenenti istruzioni per la notifica di eccezioni che favoriscono l'intervento del VMM; tali blocchi equivalenti sono passati poi in esecuzione diretta ed inoltre sono conservati in una *cache* apposita per riusi futuri, risparmiando il costo di ulteriori traslazioni. Questo processo di traslazione va applicato almeno all'intero kernel del guest OS, per essere certi che tutte le istruzioni privilegiate che non notificano eccezioni vengano intercettate e gestite.

b. *Paravirtualizzazione*. Il paradigma della paravirtualizzazione prevede che l'hardware virtuale esposto dal VMM sia funzional-

Virtualizzazione completa e paravirtualizzazione

La virtualizzazione completa prevede che il VMM esponga ad ogni macchina virtuale interfacce hardware simulate funzionalmente identiche alle corrispondenti interfacce fisiche: in questo modo è possibile installare dentro le macchine virtuali sistemi operativi standard, senza che abbiano subito alcuna modifica specifica per eseguire in ambiente virtuale. All'interno della macchina virtuale, tutte le istruzioni non privilegiate sono eseguite direttamente sul microprocessore del calcolatore (esecuzione diretta), ed il VMM si fa carico solamente di intercettare le richieste di accesso privilegiato all'hardware e ne *emula* il comportamento atteso.

La paravirtualizzazione prevede che il VMM esponga ad ogni macchina virtuale interfacce hardware simulate funzionalmente simili, ma non identiche, alle corrispondenti interfacce fisiche: piuttosto che emulare le periferiche hardware esistenti, il VMM espone una libreria di chiamate (Virtual Hardware API) che implementa una semplice astrazione delle periferiche. Occorre dunque modificare il kernel ed i driver dei sistemi operativi guest per renderli compatibili con la virtual hardware API del VMM utilizzato. La complicazione di dovere modificare i sistemi guest è però ripagata da una maggiore semplicità del VMM, che non deve preoccuparsi di intercettare in modo complicato accessi alle risorse hardware, ma si avvale della loro diretta collaborazione.

mente simile, ma non identico, a quello della sottostante macchina fisica. Piuttosto che emulare le periferiche hardware esistenti, il VMM espone una semplice astrazione delle periferiche. In particolare, l'interfaccia hardware virtuale che il VMM espone ai sistemi guest è ridisegnata nella forma di una *Applications Programming Interface* (*Virtual Hardware* API), che i sistemi guest devono sapere richiamare per guadagnare l'accesso alle risorse. Si richiede dunque che i sistemi operativi guest non siano più tenuti all'oscuro, ma siano consapevoli di operare in un ambiente virtuale. Evidentemente questo impone di modificarne il kernel ed i driver per renderli compatibili con le proprietà del VMM utilizzato; ma – ed è la cosa più importante – non occorre assolutamente modificare le applicazioni che girano sui sistemi operativi guest, perchè l'interfaccia tra le applicazioni ed il sistema operativo non viene toccata in alcun modo da questo approccio. Va dunque realizzato un porting dei sistemi operativi esistenti, poiché gli attuali non sono scritti per dialogare con una API di paravirtualizzazione ma solamente per gestire le interfacce hardware standard. Questo è certamente un problema, soprattutto per vecchi sistemi operativi di tipo legacy. Di riflesso però risulta enormemente semplificata la struttura del VMM, poiché esso non deve più preoccuparsi di individuare e catturare le operazioni privilegiate dei guest OS per poi emularle, ma si avvale invece della loro diretta e consapevole collaborazione.

Viene così meno il vincolo di operare con architetture CPU naturalmente virtualizzabili, non più essenziale per il funzionamento della paravirtualizzazione. Questo ha un impatto ancor più notevole nel contesto delle architetture IA-32 che non sono naturalmente virtualizzabili, e che – come poc'anzi visto – impongono al VMM l'implementazione di meccanismi complicati per prevenire anomalie. Il vantaggio maggiore di questo tipo di tecnica, rispetto alla precedente, consiste proprio nella maggiore semplicità ed efficienza di esecuzione del VMM.

Inoltre, vi sono contesti in cui la cooperazione tra il VMM ed il sistema guest è necessaria per raggiungere un risultato efficace: per esempio nell'ambito della gestione della memoria, si osserva che il VMM, come i tradizionali sistemi operativi, può fare uso della paginazione per spostare pagine di memoria dalla memoria primaria al disco, con il consueto vantaggio di potere allocare più memoria di quella strettamente disponibile. Ciò è particolarmente importante nel contesto della virtualizzazione, con molti sistemi e processi che insistono sulle stesse risorse sovra-allocate. Occorre dunque un meccanismo efficiente che consenta al VMM di reclamare ed ottenere in caso di necessità dalle diverse macchine virtuali le porzioni di memoria meno utilizzate. È chiaro che il sistema operativo di ogni macchina virtuale possiede informazioni relative a quali siano le pagine di memoria più adatte ad essere spostate su disco, decisamente migliori di quelle del VMM. Per esempio, un guest OS può notare che una pagina di memoria appartiene ad un processo terminato e dunque tale pagina non sarà più acceduta. Il VMM, operando al di fuori dei singoli guest OS, non può rendersi conto di questo e dunque non sarebbe altrettanto efficiente nel decidere quali pagine di memoria trasferire su disco per quella macchina. Una strategia comunemente adoperata per risolvere questo problema è di tipo "paravirtualizzante": su ogni macchina virtuale è in esecuzione un processo, a volte detto *balloon process* (processo aerostato) che comunica con il VMM. Quando vi è necessità di rastrellare memoria, il VMM chiede al balloon process di allocare più memoria, cioè di "gonfiarsi". Il guest OS, che meglio di tutti conosce l'utilizzo della memoria nell'ambito della macchina virtuale, seleziona le pagine da offrire al balloon process per soddisfarne la richiesta; il balloon process comunica tali pagine al VMM che ne rientra in possesso. La richiesta del balloon process provoca da parte del guest OS il trasferimento su disco delle pagine probabilmente meno utilizzate dalla macchina virtuale, con l'effetto netto di avere liberato memoria a favore del VMM, che provvede alla riallocazione. Anche VMM che per il resto operano secondo il paradigma della virtualizzazione completa, come lo stesso VMware, fanno largo uso di questi meccanismi tipici della paravirtualizzazio-

ne, in specifico nella gestione della memoria che – effettuata completamente dal di fuori – sarebbe altrimenti molto complessa e poco efficiente. In particolare, VMware offre l'opzione di installare dentro i sistemi guest un pacchetto di programmi (VMware Tools) in cui sono presenti questa e altre "sonde" per migliorare lo scambio di dati tra VMM e guest.
Nonostante la difficoltà di dovere realizzare il porting dei sistemi operativi esistenti, la paravirtualizzazione sta catalizzando un'attenzione sempre crescente. Il progetto attualmente più promettente di un VMM che opera secondo tale paradigma è XEN, un VMM open source sviluppato dall'Università di Cambridge [6]. Nell'ambito del progetto è stato realizzato il porting di Linux su XEN (XenoLinux), con un costo in termini di righe di codice del Kernel modificato per dialogare con la API del Virtual Hardware pari a circa 3000 righe, cioè circa 1,36% del totale. Un lavoro analogo, in collaborazione con Microsoft, è in corso per consentire il porting di Windows XP (XenoXP), ma il lavoro non è ancora terminato e risulta essere più che altro una sperimentazione. Attualmente lo sviluppo di XEN è alla versione 3; tra le funzionalità più rilevanti vi è il supporto per sistemi multiprocessore e per i più recenti kernel di Linux. Sono supportati inoltre meccanismi di "live migration" di VM da un host con XEN ad un altro: ciò permette di eseguire manutenzioni su di un singolo calcolatore senza interrompere i servizi, oppure bilanciare il carico complessivo tra i vari host con XEN di cui si dispone. Inoltre le principali distribuzioni di Linux commerciali (Suse e RedHat) offrono già i pacchetti precompilati per installare XEN, nonchè le immagini della propria distribuzione modificate per la paravirtualizzazione. XEN 3 supporta le istruzioni per la virtualizzazione che AMD ed Intel hanno di recente inserito nei propri processori basati su IA-32 e questo consente di eseguire sistemi guest Windows non modificati, in una modalità di virtualizzazione completa. In tal modo si sono superati i problemi legati al porting di sistemi guest proprietari. Sta infine crescendo l'offerta di strumenti di lavoro che permettono una gestione centralizzata di varie installazioni di XEN e delle relative macchine virtuali, da un'unica console di controllo, in analogia al prodotto VirtualCenter venduto da VMware. Il progetto più rilevante in tale ambito è Xensource, gestito dal gruppo storico che coordina lo sviluppo di XEN. Si tratta di una piattaforma commerciale per controllare più installazioni di XEN, con le relative macchine virtuali, da un'unica console centralizzata; grazie a questa piattaforma si possono compiere le operazioni fondamentali di gestione come installare un nuovo sistema guest, monitorare l'utilizzo delle risorse, modificare la configurazione di un guest, accederne la console ecc..

c. *VMM di sistema e VMM ospitati*. Soffermiamoci su un'ultima distinzione rilevante che caratterizza i sistemi di virtualizzazione: il livello dove si intende collocare il VMM rispetto all'architettura del sistema di elaborazione. Abbiamo già accennato che si distingue in questo caso tra VMM posti direttamente sopra l'hardware dell'elaboratore (*VMM di sistema*) e VMM che si istallano invece come applicazioni dentro un sistema operativo preesistente (*VMM ospitati*) (Figura 3). Nella prima opzione si integrano ad un sistema operativo "leggero" le funzionalità di virtualizzazione, in un unico sistema che esegue direttamente sopra l'hardware dell'elaboratore. Un'implementazione così a basso livello offre migliori prestazioni, anche se si rende necessario corredare il VMM di tutti i

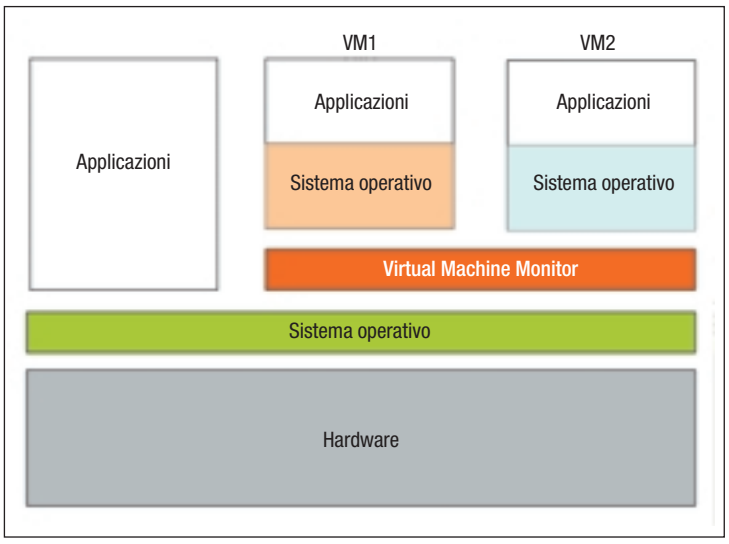

FIGURA 3
Schema puramente indicativo di un VMM ospitato su di un sistema operativo preesistente, con cui condivide l'accesso alle risorse

driver necessari per pilotare le periferiche. In generale i prodotti implementati secondo questo modello supportano un numero molto limitato di hardware certificato, rendendo meno impegnativa la gestione delle periferiche altrimenti molto difficile vista l'enorme varietà degli hardware nel mercato consumer. Un esempio di VMM di sistema è per esempio la versione ESX di VMware, non a caso quella più diffusa in ambiti professionali, o anche lo stesso XEN. Quest'ultimo adotta driver derivati dal kernel di Linux ottimizzati per offrire le migliori prestazioni sull'I/O in un contesto di virtualizzazione (varie VM in concorrenza); naturalmente tali driver sono compatibili con una lista molto ristretta di schede di rete e FibreChannel.

La seconda opzione prevede invece l'installazione del VMM come un'applicazione sopra un sistema operativo preesistente e non direttamente sull'hardware del calcolatore. Con un certo livello di approssimazione per non entrare troppo nei dettagli, si può dire che il VMM, anziché collocarsi sotto tutti gli altri livelli software, opera nello spazio utente e accede all'hardware attraverso le system call messe a disposizione dal sistema operativo su cui si installa. Le performance sono certamente inferiori, ma ci sono alcuni vantaggi importanti: il VMM è più semplice da installare poiché è come un'applicazione; inoltre potrà fare affidamento sul sistema operativo sottostante - sicuramente più fornito di driver per l'hardware più diffuso – per la gestione delle periferiche; infine, potrà servirsi di vari altri servizi dell'host OS come lo scheduling e la gestione delle risorse. È dunque una soluzione comoda, che sacrifica le performance ma semplifica il disegno del VMM ed è supportata da qualsiasi piattaforma IA-32 sulla quale sia già installabile un sistema operativo. Spesso tale opzione è più che sufficiente per un utente che abbia solo l'esigenza di avere contemporaneamente attivi sul proprio PC diversi sistemi operativi per sviluppare o testare applicazioni. In questo segmento, per le architetture x86, si trovano la versione gratuita di VMware Server (prima noto come GSX), Virtual Server di Microsoft, il software opensource User Mode Linux.

VMM di sistema e VMM ospitati

I VMM di sistema si installano direttamente sull'hardware del calcolatore, ed integrano le funzionalità di virtualizzazione ad un sistema operativo minimale. Un'implementazione a basso livello offre migliori prestazioni, anche se si rende necessario corredare il VMM di tutti i driver necessari per pilotare le periferiche. In generale i prodotti implementati secondo questo modello supportano un numero molto limitato di hardware certificato, risultando compatibili con un piccolo sottoinsieme dell'hardware presente nel mercato consumer.

I VMM ospitati si installano come un'applicazione all'interno di un sistema operativo preesistente e non direttamente sull'hardware del calcolatore. Con un certo livello di approssimazione si può dire che il VMM opera nello spazio utente e accede l'hardware attraverso le system call messe a disposizione dal sistema operativo su cui si installa. Le performance sono certamente inferiori, ma il VMM è più semplice da installare ed inoltre si affida al sistema operativo sottostante - sicuramente più fornito di driver per l'hardware più diffuso – per la gestione delle periferiche. Esso potrà inoltre utilizzare servizi dell'host OS come lo scheduling e la gestione delle risorse. È dunque una soluzione comoda, che sacrifica le performance ma semplifica il disegno del VMM.

3. VIRTUALIZZAZIONE E CONSOLIDAMENTO DEI SERVER ALL'UNIVERSITÀ DI BOLOGNA

3.1. Il contesto

L'Università di Bologna ha concentrato presso il proprio Centro Servizi Informatici d'Ateneo (CeSIA) la gestione della rete accademica ALMAnet e della Server Farm che ospita i sistemi per i servizi centralizzati: posta elettronica, DNS, file server, portale web d'Ateneo, sistemi per l'autenticazione, strumenti per il monitoraggio e la gestione della rete ALMAnet, le basi dati di personale e studenti, le applicazioni per le segreterie, servizi per docenti, studenti, e per l'amministrazione generale. All'inizio del 2005 ci si è posti il problema del rinnovamento della Server Farm, che aveva raggiunto dimensioni critiche: 200 server stand alone, Linux e Windows, ognuno con il proprio disco in locale, ognuno dedicato in modo esclusivo ad una singola applicazione o servizio, e come tale piuttosto sottoutilizzato. Un modello così rigido non era chiaramente in alcun modo scalabile e già così poneva grossi problemi sui costi di approvvigionamento e di manutenzione; vi erano inoltre problemi di ordine logistico come il condizionamento dell'ambiente e la distribuzione dell'energia elettrica. Ma, soprattutto, si osservava che la sola gestione ordinaria di un parco così ampio assorbiva completamente il personale preposto, che vedeva ridursi il tempo per la parte più qualificante del lavoro: lo sviluppo dei ser-

vizi a valore aggiunto che sono oggi considerati fondamentali, come la remotizzazione dei dati, le politiche di recupero da disastro e di continuità operativa (Figura 4).
Si è così stabilito di avviare un progetto di razionalizzazione della Server Farm secondo linee più moderne, con l'obiettivo primario della diminuzione del mero lavoro gestionale e dei costi di manutenzione, da attuarsi mediante una riduzione del parco macchine e concentrando più servizi sui medesimi sistemi (*consolidamento hardware ed applicativo*). Era poi richiesto che la nuova architettura supportasse in modo semplice e nativo funzionalità di recupero da disastro e di continuità operativa, da attuarsi con l'appoggio di un sito secondario a circa un chilometro di distanza dal CeSIA e connesso con esso tramite fibre ottiche, di cui l'Ateneo si era nel frattempo dotato. Si richiedeva inoltre il supporto per sistemi Windows e Linux, entrambi ampiamente utilizzati in Ateneo e la possibilità di sviluppare in modo più semplice, economico e standardizzato i servizi intorno ai server, in particolare backup/restore, monitoraggio, ridondanze.
Sono state studiate le tecnologie per il consolidamento ed acquisiti gli elementi di base per realizzare un'architettura consolidata: hardware scalabili che consentono di dimensionare la potenza di elaborazione e lo spazio su disco in modo dinamico al crescere della necessità, e sistemi operativi per la virtualizzazione, che consentono di partizionare le risorse hardware disponibili tra diversi sistemi che ne fanno un uso concorrente.
I calcolatori scelti sono quadriprocessori dualcore in architettura IA-32; essi accedono ai sottosistemi disco attraverso la rete Fibre-Channel, una tecnologia di trasporto ottimizzata per la comunicazione dei comandi SCSI.
Dei diversi VMM per architetture IA-32 è stato preferito VMware ESX per le seguenti ragioni: supporta sistemi operativi guest sia di tipo Windows che Linux; è dotato di una console di gestione centralizzata (VirtualCenter) che permette di tenere sotto controllo ed eseguire operazioni su tutte le macchine virtuali installate su qualsiasi calcolatore con VMware; dispone di meccanismi per la continuità operativa (Vmotion) che permettono la migrazione a caldo di una macchina virtuale

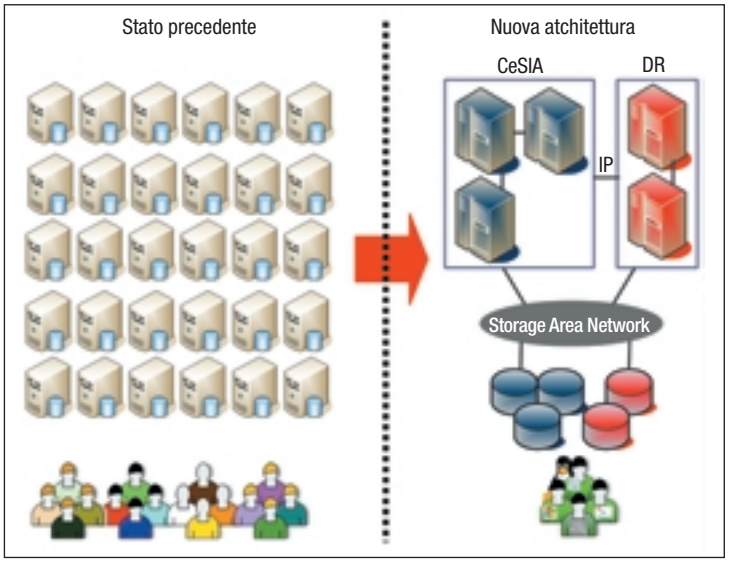

FIGURA 4

Modelli di gestione a confronto: rispetto ad un modello classico di gestione tipo "un server fisico per un'applicazione", si evidenza la modularità dell'architettura nella nuova infrastruttura, la semplicità nell'estendere su siti comunicanti la Server Farm per ottenere continuità operativa, il ridotto numero di persone necessarie al presidio

da un calcolatore con VMware ad un altro; dispone di driver per la gestione del più diffuso hardware professionale in commercio, con funzionalità avanzate per supportare la ridondanza dei percorsi di I/O; è dotato di un sistema molto raffinato di gestione della memoria (risorsa scarsa e costosa) che consente una forte sovra-allocazione di memoria alle VM rispetto alla memoria fisica presente.
Va comunque osservato che le funzionalità descritte non sono in alcun modo prerogativa del prodotto di VMware, ma con gradi di sviluppo diversi sono caratteristiche comuni di quasi tutti i sistemi di virtualizzazione commerciali ed opensource presenti sul mercato ([5, 6, 7, 12] e altri ancora). In particolare, ciò che ha permesso il cambiamento radicale nel modello di gestione della Server Farm è stata la combinazione di hardware scalabile di tipo "*commodity*" (cioè facilmente reperibile sul mercato, e indifferentemente da produttori diversi) unito al partizionamento delle risorse operato secondo principi molto simili da qualsiasi VMM per architetture IA-32. In particolare, è patrimonio comune di queste tecnologie operare il *multiplexing* delle risorse disponibili ed incapsulare un intero server in un solo file su

disco, azioni che sono la chiave di volta di tutte le funzionalità sopra descritte.

3.2. La soluzione

È nato così il nucleo della nuova Server Farm: in una tale architettura, una macchina virtuale è un processo in esecuzione su di un calcolatore, ed il suo disco è interamente incapsulato in un file posizionato sullo Storage Array. Tutti i nuovi server sono stati sistematicamente creati come VM all'interno di questa infrastruttura. I vecchi server sono stati migrati uno ad uno all'interno della nuova infrastruttura, alcuni ricostruendoli da zero in ambiente virtuale, altri trasferiti da fisico a virtuale mediante strumenti automatici che si occupano di ricostruire in una VM un server fisico ed il contenuto dei suoi dischi. Tali software sono spesso indicati come P2V, che sta per "Physical to Virtual".

Per supportare la continuità operativa, le reti di trasmissione dati sono state estese dal CeSIA verso il sito secondario mediante fibre ottiche di proprietà dell'Ateneo; nel sito secondario è stata predisposta un'uscita di backup verso Internet, e lì sono stati posti alcuni calcolatori con VMware ed uno dei due Storage Array acquistati. Attraverso la rete FibreChannel, i calcolatori in un sito vedono anche il disco che è nell'altro sito: così le macchine virtuali in esecuzione al CeSIA possono essere migrate a caldo – tramite Vmotion – anche sui calcolatori presenti nel sito secondario in pochi secondi, consentendo ad esempio di effettuare manutenzioni su un singolo calcolatore senza interrompere i servizi. Inoltre, in caso di rottura di un calcolatore nel sito primario i calcolatori del sito secondario possono intervenire riavviando i server caduti. Mediante meccanismi di replica sincrona e asincrona dei dati attuati tra i controller dei due Storage Array, è possibile mantenere sul sito remoto una copia allineata in tempo reale dei dati più critici del sito primario, per esempio le basi dati di personale e studenti. In modo analogo si può mantenere nel sito secondario una copia delle immagini disco delle macchine virtuali più importanti, per potere ripristinare i servizi nel sito secondario anche nel caso di un disastro che paralizzi totalmente il CeSIA.

Dopo un'attenta fase di test, si è compreso che era possibile concentrare fino a 40 VM su di un unico quadriprocessore (8 core) con 32 GB di RAM. La vecchia sala server di 200 macchine fisiche è stata così ristretta a soli 7 quadriprocessori, con risparmi ingenti dal punto di vista dei costi di gestione. Le operazioni quotidiane di presidio e gestione ordinaria si sono radicalmente semplificate poiché l'intera Server Farm è diventata controllabile da un unica interfaccia – VirtualCenter – attraverso cui è possibile accedere alle console dei server, monitorarne l'uso delle risorse, cambiarne a caldo la connettività di rete, attivare allarmi al superamento di soglie critiche, comandarne la migrazione a caldo nel caso occorra compiere manutenzioni sull'hardware sottostante. È inoltre diventato semplice aggiornare l'hardware virtuale di una VM: si può incrementare la RAM o il numero di processori, aggiungere spazio disco o ulteriori schede di rete nello spazio di un riavvio.

Un altro vantaggio fondamentale dell'ambiente virtuale è legato alla creazione di un nuovo server: si possono infatti confezionare dei *template* di installazioni tipiche (per esempio, una immagine di Windows 2003 con a bordo antivirus, agente di backup e altre applicazioni già configurate) ed eseguire installazioni di nuovi server virtuali a partire da essi, con un notevole risparmio di tempo ed energie. Si può anche creare un nuovo server clonando una macchina virtuale già in esecuzione, ottenendo in modo molto rapido un ambiente di test fedele all'originale per provare ad esempio l'effetto di aggiornamenti di sistema, variazioni di configurazione etc. Inoltre, liberarsi di questi ambienti è facile come cancellare un file: un vantaggio enor-

Virtual Appliance

Le Virtual Applicance sono macchine virtuali confezionate e configurate con a bordo tutto l'occorrente per svolgere funzioni applicative di un certo tipo, come web server, firewall, applicazioni di fonia su IP, ecc.. Esistono in rete repositori che contengono centinaia di immagini di Virtual Appliances divise per categorie e scaricabili gratuitamente. Ad oggi tali immagini sono eseguibili esclusivamente sul Virtual Machine Monitor per cui sono state create, poichè non esistono standard condivisi per il formato del disco delle VM o per le interfacce di paravirtualizzazione. Sono comunque già molto utilizzate per testare e prototipare rapidamente ambienti operativi.

me rispetto all'approccio classico che impone investimenti in acquisto di hardware dedicato, l'installazione del sistema operativo e delle applicazioni ecc. con pochissima flessibilità e costi elevati in termini di risorse umane e materiali.
La modularità dell'architettura, ad ogni suo livello, permette di acquisire autonomamente ed in stock le parti necessarie se le risorse per nuove VM scarseggiano: ulteriori calcolatori per aumentare la capacità di elaborazione, o ulteriori dischi per lo Storage Array. Si beneficia così di una riduzione dei costi sia in virtù di acquisti in lotto, sia in virtù della gestione centralizzata dell'hardware e della elevata standardizzazione delle componenti. Dal punto di vista organizzativo poche persone, esperte sui temi della virtualizzazione dei sistemi operativi, dei sistemi di calcolo e disco, delle reti, sono in grado di governare senza affanno un'architettura come quella descritta, poiché la quota di lavoro meramente gestionale si riduce in modo notevole, e non scala in modo lineare con il numero dei server.

Bibliografia

[1] Abramson, at al.: Intel Virtualization Technology for directed I/O. *Intel Technology Journal,* Vol. 10, Issue 3, 2006.

[2] Adair R.J., Bayles R.U., Comeau L.W., Creasy R.J.: *Virtual Machine for the 360/40*. IBM Corp., Cambridge Scientific Center, Report, n. 320-2007, May 1966.

[3] Adams K., Agesen O.: *A Comparison of Software and Hardware Techniques for x86 Virtualization*. ASPLOS'06 Conference Proceedings, October 21–25, 2006, San Jose, California, USA.

[4] AMD Corporation: *AMD64 Virtualization Codenamed "Pacifica" Technology: Secure Virtual Machine*. Architecture Reference Manual, May 2005.

[5] AA.VV.: *User Mode Linux*. 2006, http://user-mode-linux.sourceforge.net/

[6] Barham P., Dragovic B., Fraser K., Hand S., Harris T., Ho A., Neugebauer R., Pratt I., Warfield A.: *Xen and the Art of Virtualization*. Proc. of the 19-th ACM SIGOPS, October 2003.

[7] Bellard F.: *QEMU opensource process emulator*, 2006. http://fabrice.bellard.free.fr/qemu/

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

[9] Dijkstra E.W.: The Structure of the Multiprogramming System. *Communication of the ACM*. Vol. 8, n. 9, 1968.

[10] Goldberg R.P.: *Survey of Virtual Machines*. Computer, June 1974.

[11] Goslin B., Steele G.: *The Java Language Specification*. Addison Wesley,1996.

[12] Microsoft Corporation: *Microsoft Virtual Server 2005 R2 Technical Overview, 2005*. http://www.microsoft.com/windowsserversystem/virtualserver/overview/vs2005tech.mspx

[13] Popek G.J, Goldberg R.P: Formal Requirements for Virtualizable Third Generation Architectures. *Communication of the ACM*, July 1974.

[14] Rosemblum M., Garfinkel T.: *Virtual Machine Monitors: Current Technology and Future Trends*. IEEE Computer, May 2005.

[15] Salmoria N.: *MAME, the Multiple Arcade Machine Emulator*. http://www.mame.net

[16] Sites R., et al.: Binary Translation. *Comm. ACM*, Febr. 1993.

[18] Sugerman J., Venkitachalam G., Beng-Hong Lim: *Virtualizing I/O devices on VMware Workstation Hosted Virtual Machine Monitor*. Proc.Usenix Annual Technical Conference., June 2001.

[17] Smith J.E, Ravi Nair: *The Architecture of Virtual Machines*. IEEE Computer, May 2005.

[19] Uhlig C., Neiger G., et altri : *Intel Virtualization Tecnology*. IEEE Computer, May 2005.

[20] Waldspunger C.: *Memory Resource management in VMware ESX Server*. ACM SIGOPS Operating Systems Rew, Winter 2002.

[21] Klaiber A.: *The technology beyind Crusoe processors: low-power x86-compatible processors implemented with code morphing software*. Tech. brief, Transmeta Corp., 2000.

Simone Balboni è responsabile dei sistemi e servizi di rete presso il Centro Servizi Informatici d'Ateneo dell'Università di Bologna (CeSIA). Ha conseguito il dottorato di ricerca in Fisica computazionale ed è autore di alcuni articoli sul calcolo scientifico, sulle trasmissione in rete di dati multimediali, sulla sicurezza informatica.
E-mail: simone.balboni@unibo.it

Maurelio Boari è professore ordinario di calcolatori elettronici presso la Facoltà di Ingegneria dell'Università di Bologna. È autore di numerosi articoli scientifici ed alcuni libri. Ha interessi di ricerca nel settore dei sistemi distribuiti, linguaggi di programmazione e sistemi operativi. È attualmente delegato del Rettore per l'informatica e le reti di Ateneo.
E-mail: maurelio.boari@unibo.it