Virtualizzazione

La **virtualizzazione del sistema** permette di astrarre le risorse hardware di un computer in diversi ambienti di esecuzione. Una **macchina virtuale** è un ambiente d'esecuzione che consiste di un sistema operativo e le applicazioni, più tutte le risorse virtuali che gli sono assegnate. Il **virtualization layer** partiziona, gestisce e condivide le risorse hardware tra le diverse macchine virtuali.
Componenti della virtualizzazione:

- Host: la macchina fisica che esegue le macchine virtuali
- Host OS: il sistema operativo in esecuzione sull'host
- Ipervisore: crea e esegue VM fornendo loro una piattaforma hardware virtuale equivalente a quella presente sull'host
- Macchina Virtuale (o guest): l'entità controllata dall'ipervisore
- Guest OS: il SO in esecuzione sul guest

Le macchine virtuali sono composte da file, di solito il file di configurazione e i file del disco virtuale. Su un sistema di virtualizzazione è possibile sospendere una VM in esecuzione e crearne template. Inoltre, la maggior parte dei sistemi di virtualizzazione offre live migration delle VM, ovvero lo spostamento di un guest in esecuzione da un host ad un altro, senza interromperne l'opeazione.

1 Vantaggi

La virtualizzazione offre molti vantaggi:

- Rapid provisioning: clonando una VM è possibile ottenere una copia di un server in pochi minuti
- Isolamento e sicurezza: la condivisione delle risorse è gestita dall'ipervisore
- Sviluppo e testing rapido
- Scalabilità verticale delle risorse: se una VM richiede ulteriori risorse, le si possono fornire allocando ulteriori risorse fisiche
- Consolidazione dei server: aumenta l'utilizzazione dei server

2 Ipervisori

Un ipervisore deve soddisfare i seguenti requisiti: **fedeltà** (una applicazione deve essere eseguita allo stesso modo sia su HW che su VM), **efficienza** e **sicurezza** (controllo completo delle risorse virtuali).
Gli ipervisori sono di due tipi. L'**ipervisore di tipo 1** è uno strato software che viene eseguito direttamente sull'hardware del computer. Gestisce le risorse fisiche del computer e supporta l'esecuzione della VM implementando i servizi di base che sono normalmente forniti dal SO

(scheduling CPU, gestione memoria...). Esistono due architetture per implementare le suddette funzionalità. Nella prima, l'ipervisore implementa e gestisce tutte le funzionalità per eseguire le VM e di solito sono eseguibili solo su piattaforme ben conosciute, per le quali sono diponibili tutti i driver. Nella seconda, l'ipervisore si appoggia al SO esistente per fornire i servizi base; una speciale VM di servizio fornisce servizi del SO e gestisce l'ipervisore.
L'**ipervisore di tipo 2** è un programma gestito dal SO, che fornisce la piattaforma virtuale per interfacciarsi con le sue VM. Le risorse HW sono gestite dal SO.
Un ipervisore di tipo 1 è molto più efficiente di un ipervisore di tipo 2, dato che viene eseguito direttamente su HW ed inoltre fornisce maggiore isolamento, ma gli ipervisori di tipo 2 sono più flessibili in quanto non richiedono di dedicare un server alla virtualizzazione.

3 Virtualizzazione della CPU

Ad ogni VM sono assegnate una o più **CPU virtuali**; esse non eseguono istruzioni, ma rappresentano lo stato della CPU in cui il SO ospite crede che sia. Ad ogni vCPU sono associate delle CPU fisiche (pCPU) che eseguono istruzioni. L'ipervisore suddivide la CPU fisica tra diverse VM tramite il **time-sharing**. L'ipervisore schedula il tempo della CPU fisica per ogni richiesta della VM: quando il SO ospite passa istruzioni alla vCPU, l'ipervisore intercetta la richiesta, schedula del tempo sulla pCPU e ne ritorna il risultato al SO ospite. Se il SO ospite occupa la pCPU per il tempo massimo consentito, l'ipervisore ne requisisce il controllo (**machine switch**). Quando viene esguita una machine switch, l'iperviore deve salvare lo stato della VM in esecuzione, caricare lo stato della macchina da eseguire e riprenderne l'esecuzione.
Per eseguire le istruzioni dell'ospite sono usate diverse tecniche. Con la tecnica dell'**emulazione** l'ipervisore interpreta in SW ogni istruzione e mantiene lo stato di ogni VM puramente in SW. Questo metodo è facile da implementare ed è portabile, ma è molto lento.
Non è possibile eseguire ogni istruzione direttamente su HW perchè il SO ospite è eseguito in modalità utente e non modalità kernel. La seconda tecnica è quella del **trap-and-emulate**. L'ipervisore lascia che le istruzioni della VM siano eseguite direttamente sulla CPU: quando il SO ospite prova ad eseguire un'istruzione privilegiata, causa una trap che è intercettata dall'ipervisore, il quale esegue (emula) l'istruzione. Ogni istruzione che non dovrebbe essere consentita in modalità utente deve risultare in una trao al kernel.
L'architettura x86 fornisce 4 livelli di privilegio, chiamati **anelli di protezione**, che gestiscono l'accesso all'hardware. Solo l'anello più privilegiato (anello 0) può eseguire istruzioni privilegiate (gli altri causano trap): senza virtualizzazione, il SO è eseguito nell'anello 0, le applicazioni utente nell'anello 3. L'ipervisore è eseguito nell'anello 0 e il SO ospite è eseguito in un anello meno privilegiato, di solito l'anello 1.
La terza tecnica è quella della **traduzione binaria dinamica**: se il SO ospite esegue un'istruzione sensibile (istruzione che tenta di cambiare configurazioni di sistema o che si comporta in modo diverso a seconda della configurazione delle risorse HW) l'ipervisore rimpiazza l'istruzione con codice che evita di chiamare la suddetta istruzione. Per far ciò, l'ipervisore legge dinamicamente le istruzioni del SO ospite, guardando in particolare i **basic blocks** (sequenza di istruzioni senza branching).
La quarta tecnica è quella della **paravirtualizzazione** (o virtualizzazione assistita dal SO). Il SO ospite coopera con l'ipervisore per rendere l'architettura virtualizzabile: il SO ospite è modificato in modo da rimpiazzare le istruzioni non virtualizzabili con delle **hypercall** che comunicano direttamente con l'ipervisore. Problemi riguardanti la portabilità e la gesione sono stati parzialmente

risolti dall'introduzione di un'interfaccia di virtualizzazione standard nel kernel.
La quinta tecnica è quella della **virtualizzazione assistita dall'hardware**: è virtualizzazione totale con trap and emulate, dove l'ipervisore può sfruttare le estensioni di virtualizzazione dell'HW. Nell'architettura x86, queste estensioni sono chiamate Intel Virtualization Technology (VT). VT duplica lo steto del processore e introduce due nuove modalità d'esecuzione: host mode, dove l'esecuzione nell'anello 0 garantisce accesso diretto all'HW (ipervisore e SO host sono eseguiti in questa modalità) e guest mode, dove le VM sono eseguite. La transizione tra modalità avviene in HW ed è gestita da una struttura dati chiamata Virtual-Machine Control Structure, che salva/carica automaticamente lo stato del processore durante la transazione. Quando il SO ospite esegue un'istruzione che causa il passaggio a host mode, il controllo è trasferito all'ipervisore, che esegue l'istruzione o su HW o tramite emulazione. Completata l'operazione, si passa a guest mode e il controllo è ritornato al SO ospite. Questa tecnica rende la virtualizzazione più semplice, ma può causare molto overhead a causa della transizione costante tra modalità.
Gli ipervisori di tipo 2 di solito hanno dei moduli kernel che sono eseguiti nell'anello 0 che consentono all'ipervisore di eseguire istruzioni privilegiate.

4 Virtualizzazione della memoria

L'ipervisore partiziona la RAM fisica tra le varie VM, dandone l'illusione di averne totale controllo. Il SO ospite, inoltre, utilizza memoria virtuale per astrarre la memoria ai suoi processi, mappando indirizzi virtuali su indirizzi fisici tramite page tables. Come mappare un page frame guest sul page frame dell'host?
Ci sono tre soluzioni. La prima, usata nella virtualizzazione totale, si chiama **shadow paging**: per ogni processo su ogni VM, l'ipervisore crea e mantiene una **shadow page table** (sPT) che mappa le pagine virtuali del guest sui page frame dell'host. Quando il page table base registry (PTBR, un registro della CPU che punta alla page table di livello superiore) del guest, a cui l'ipervisore ha accesso, viene scritto, l'ipervisore lo intercetta e aggiorna il PTBR in modo che punti alla sPT corrispondente. Per tener aggiornata la sPT, l'ipervisore deve tenere traccia dei cambiamenti che il guest fa alle sue page table. Il primo approccio è quello di marcare la guest page table come write-protected: in questo modo ogni modifica risulterà in un page fault, che farà trap all'ipervisore. Il secondo approccio è quello di usare un TLB virtuale: l'ipervisore lascia che il guest aggiunga mappature alla sua page table, ma non appena esso cerca di accedervi, si ha un page fault (la pagina non è nella sPT). Entrambi gli approcci funzionano, ma causano molti page fault (costosi in termini di tempo), sia a livello guest sia a livello ipervisore.
Con paravirtualizzazione, il guest sa di trovarsi in ambiente virtuale e quindi, per ogni cambiamento effettuato alla sua page table, invoca una hypercall. Per ottimizzare, l'hypercall è invocata solo dopo un determinato numero di cambiamenti.
L'HW offre supporto alla virtualizzazione della memoria nella forma di **nested paging**, che elimina il bisogno di introdurre lo shadow paging. La nested page table (nPT) mappa indirizzi fisici del guest agli indirizzi fisici dell'host. Quando un guest cerca di referenziare memoria tramite un indirizzo guest virtuale, viene fatta una page walk bidimensionale usando la guest page table e la nPT per tradurre l'indirizzo virtuale del guest nell'indirizzo fisico dell' host. Questa traduzione è molto costosa: con m livelli di guest page table e n livelli di nPT vengono effettuati (n+1)*m+n accessi alla memoria.
La quantità totale di RAM allocata alle VM può eccedere la quantità di RAM fisica presente sul sistema host. L'ipervisore può condividere pagine con lo stesso contenuto tra più macchine virtuali

(**deduplicazione**), usando **copy-on-write** per gestire i cambiamenti (l'ipervisore rimpiazza una pagina condivisa con una nuova copia che verrà assegnata alla VM in modo che i suoi cambiamenti non influenzino le altr VM). Un altra soluzione è quella di usare una tecnica chiamata **ballooning**, dove un *balloon module* è caricato su ogni VM come driver che comunica con l'ipervisore. Il modulo, a richiesta dell'ipervisore, si può gonfiare (allocando più pagine) e sgonfiare (deallocandole): questo costringe il guest ad ottimizzare le sue risorse.