POLITECNICO DI MILANO

Corso di Laurea in Ingegneria Informatica

Dipartimento di Elettronica e Informazione

Analisi delle Prestazioni di Sistemi Virtualizzati basati su KVM

Relatore: Ing. Danilo Ardagna

Tesi di Laurea di:

Salvini Paolo - Mat. 703265

Trevisiol Ettore - Mat. 701651

Anno Accademico 2009-2010

I nostri più sentiti ringraziamenti vanno al Prof. Danilo Ardagna, per la fiducia dimostrataci nell'assegnarci questo lavoro di tesi e per averci seguito durante lo svolgimento del lavoro con consigli e incontri per intraprendere il giusto cammino.

Un grazie di cuore ai nostri genitori per la costante fiducia e il grande affetto che ci hanno dato in questi anni di studio, e senza i quali non avremmo raggiunto questo obiettivo.

Come non ringraziare tutti gli amici dell'Università, in modo particolare Giorgio, Emanuele, Mikhail, Marco, Stefano, Dario, Luigi, Riccardo e Federico con i quali abbiamo seguito numerosi corsi, sviluppato progetti, condiviso appunti e conoscenze.

Infine un pensiero va a tutti i processi che abbiamo ucciso in questo lavoro di tesi.

Indice

Elenco delle figure

Introduzione

La rapida crescita e diffusione dei servizi web-based ha comportato l'introduzione di infrastrutture IT sempre più ampie e potenti. Questo ampliamento corrispondeva, soprattutto in passato, all'aggiunta di nuovi apparati hardware, che però ovviamente causavano un progressivo aumento dell'incidenza dei costi di gestione ed energetici sui budget aziendali. Le aziende, soprattutto negli ultimi anni, sono diventate più sensibili rispetto a questo problema e nel taglio delle spese superflue. Molti studi sono stati condotti per raggiungere l'obiettivo di ridurre il consumo energetico dei device hardware, in particolare per la riduzione del numero di macchine impiegate all'interno di un'organizzazione. Questo è possibile ottimizzando le risorse disponibili eseguendo diverse applicazioni sullo stesso server e eliminando quei server ai quali erano demandate precedentemente l'esecuzione di queste attività. Questa tecnica è detta Server Consolidation.

L'obiettivo è ridurre i costi mantenendo nello stesso tempo un livello di servizio adeguato agli standard concordati con i clienti. Uno degli strumenti fondamentali per la riduzione dei costi operativi è la virtualizzazione, cioè l'insieme di tecniche che permettono di gestire un certo numero di macchine virtuali che utilizzano la stessa macchina fisica, riducendo così drasticamente il numero di componenti dell'architettura hardware e massimizzando l'utilizzo delle risorse. Nell'approccio della server virtualization le risorse hardware sono partizionate e condivise tra un certo numero di macchine virtuali che possono accedere alle risorse fisiche solo attraverso il Virtual Machine Monitor. Questo garantisce sicurezza e isolamento delle prestazioni: ogni virtual machine, nonostante faccia uso dell'hardware condiviso, è indipendente e quindi possibili malfunzionamenti non vanno a compromettere l'intera architettura. Inoltre la virtualizzazione permette una forte modularizzazione del sistema rendendo velocemente rimpiazzabile un componente guasto. Tale tecnica agevola notevolmente la possibilità di separare le macchine virtuali da quelle fisiche, cioè una virtual machine può migrare facilmente tra più macchine fisiche.

Nel caso particolare del nostro lavoro di tesi l'obiettivo è stato quello di è di analizzare le prestazioni di un server che sfrutta una tecnologia di virtualizzazione di nuova generazione chiamata Kernel-based Virtual Machine (KVM) utilizzando SPECweb2005 come applicazione di benchmark e confrontare i risultati con quelli relativi a Xen.

Il lavoro di tesi è organizzato come segue:

Nel Capitolo 1 verrà presentata una panoramica sull'architettura delle web application e sulla Quality of service, verranno discusse in dettaglio le tecniche di virtualizzazione, descritti gli hypervisor Xen e KVM e una breve introduzione allo scheduling di Linux. Nell'ultima parte del capitolo verrà descritto SPECweb, l'applicazione di benchmarking utilizzata per la valutazione delle performance dei web server.

Nel capitolo 2 verrà presentato il tool che è stato utilizzato per l'automatizzazione dell'esecuzione dei test di performance delle macchine virtuali. E' stata utilizzata un' architettura sperimentale di tipo client-server che permette di configurare ed eseguire diversi test in successione. Verranno quindi descritti i vari componenti e verrà presentato nel dettaglio il funzionamento del tool.

Nel capitolo 3 verranno descritte le problematiche dovute alla migrazione da un sistema virtualizzato ad un altro e verrà descritta la tecnica utilizzata per convertire immagini Xen in immagini KVM.

Nel capitolo 4 verrano descritte le modifiche che sono state apportate al tool presentato nel capitolo 2 necessarie per adattarlo alla nuova infrastruttura della macchina.

Nel Capitolo 5 verrà descritto l'ambiente in cui sono stati effettuati i test e la scelta dei parametri di test.

Nel Capitolo 6 verranno presentati e discussi i risultati ottenuti dai test sia a singola sia a doppia VM sulla piattaforma KVM e confrontati con quelli di Xen.

Capitolo 1

Stato dell'arte

Il seguente capitolo illustra una panoramica sullo stato delle tecnologie esistenti per lo sviluppo di sistemi informativi che forniscano servizi garantendo vincoli di qualità. Nella Sezione 1.1 viene discussa l'architettura multi-tier tipica degli applicativi, ossia un'architettura che suddivide l'elaborazione di una richiesta in più fasi, ognuna delle quali viene demandata in genere ad un particolare server fisico. Per poter gestire un'eventuale crescita dei carichi di lavoro si ricorre molto spesso alla replicazione dei server, andando a costituire in questo modo le cosiddette Server Farm, ossia insiemi di centinaia, in alcuni casi migliaia, di server collocati in un unico ambiente in modo da poterne centralizzare la gestione, la manutenzione e la sicurezza. Le funzioni vengono erogate dai sistemi garantendo vincoli di QoS (Quality of Service) che vengono concordati tra il Service Provider e il Customer. Tali vincoli verranno esaminati nella Sezione 1.2. Le moderne architetture presentano costi di impianto e di gestione molto elevati, pertanto si stanno facendo strada tecniche che permettano il consolidamento delle infrastrutture. Nella Sezione 1.3 vengono introdotte le architetture di virtualizzazione che permettono di ridurre in maniera sensibile i costi delle infrastrutture e di ottimizzarne l'utilizzo.

1.1 Le applicazioni Internet

Con la crescita e l'espansione di applicazioni web-based le architetture responsabili della fornitura di tali servizi sono diventate sempre più complesse. Questo al fine di garantire requisiti di availability e di sicurezza. Pertanto si fa ricorso ad architetture replicate e distribuite. Con il termine sistema informatico distribuito si indica una particolare architettura che gestisce applicazioni fra loro cooperanti. Esse risiedono su più nodi elaborativi e i dati di cui fanno uso, seppur unitari da un punto di vista logico, sono suddivisi in più archivi fisici.

L'architettura di un sistema distribuito può essere, da un punto di vista generale, molto complessa. Se ne può semplificare la descrizione riconoscendo tre livelli logici di distribuzione:

1. Layer di presentazione: si occupa di gestire le modalità di iterazione con l'utente. Questo livello è anche denominato front-end delle applicazioni.

2. Layer di logica applicativa: contiene i programmi che automatizzano le funzioni operative e strategiche.

3. Layer di accesso ai dati: si occupa della gestione delle informazioni accedendo a basi di dati o interagendo con sistemi legacy.

I livelli 2 e 3 costituiscono il back-end dell'applicazione occupandosi di gestire in modo trasparente per l'utente l'esecuzione dei programmi e il flusso dei dati.

Questi livelli logici nei sistemi distribuiti vengono mappati su livelli fisici. Da questo punto di vista nella definizione di sistemi distribuiti un'applicazione può essere configurata come:

Sigle Tiered: i tre livelli software sono assegnati ad un'unica macchina. Questa configurazione caratterizza i sistemi basati su mainframe che, nonostante sia un modello datato, è in grado di garantire ancora un livello di prestazioni, affidabilità e sicurezza che non è stato pienamente eguagliato dai moderni sistemi distribuiti.

Two Tiered: i livelli applicativi sono divisi tra la stazione di lavoro dell'utente e la macchina server che ospita i dati. La logica di presentazione è realizzata sulla macchina client, mentre l'accesso ai dati è consentito solo alla macchina server. Si possono sviluppare diverse politiche di allocazione della logica applicativa e in parte della presentazione. A seconda della complessità della logica installata sul client si parla di thin client se al lato utente è delegata solo la logica di

presentazione, di fat client se a livello utente si aggiunge parte della logica applicativa o di gestione dei dati (Figura 1.1).

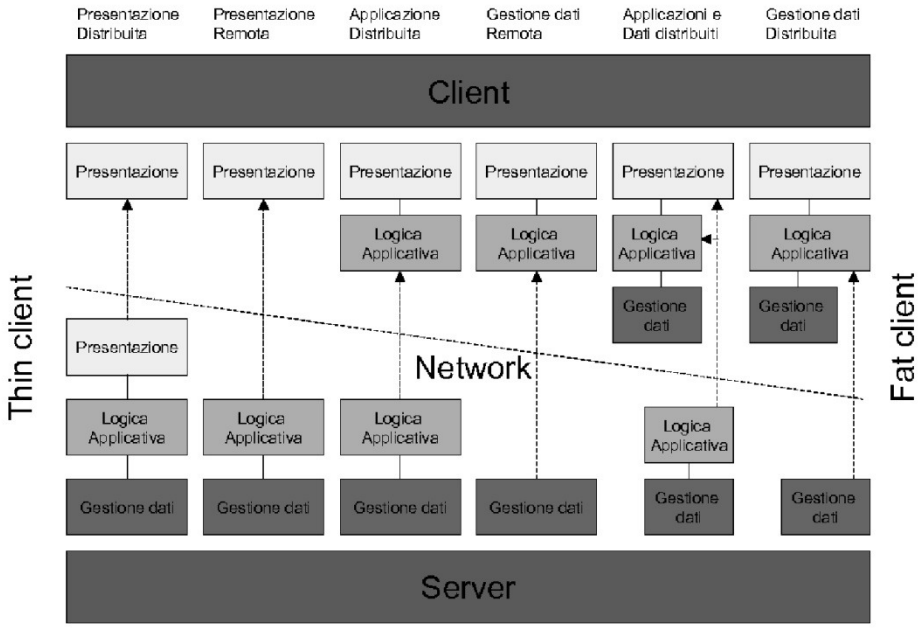


Figura 1.1 - Classificazione delle architetture 2-tier

Three Tiered: i tre livelli applicativi sono suddivisi tra altrettante macchine dedicate, ovvero una stazione di lavoro utente, un server applicativo e un server per la gestione dei dati. Tale architettura (Figura 1.2) fornisce un approccio flessibile e modulare per il design dell'applicazione internet e conferisce alle infrastrutture ICT maggiore scalabilità. L'utilizzo di un server applicativo intermedio consente la riduzione del carico al DBMS server. Il middle tier mantiene connessioni permanenti con il DBMS permettendo di ridurre notevolmente il numero di richieste che il server dati deve gestire.

L'architettura three-tier svolge un ruolo importante nella progettazione di applicazioni Web, specialmente basate su Web service (Figura 1.2).

Nelle implementazioni reali si adottano spesso soluzioni con più di tre tier. Ad esempio nei moderni sistemi di eBusiness si sviluppano soluzioni a cinque tier con il layer applicativo composto da web server, script engine e application server. Questa ulteriore suddivisione ha richiesto l'introduzione di sistemi di bilanciamento del carico (load balancer) che si occupano dello smistamento delle richieste in modo da garantire un carico uniforme su tutti i server di un dato tier.

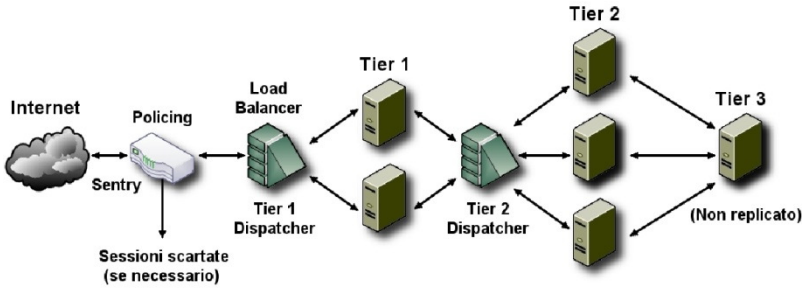


Figura 1.2 - Un'architettura 3-tier

1.2 Quality of Service

Quando è stata creata Internet, non è stata percepita la necessità di Qualita di Servizio per le applicazioni. Infatti Internet segue la filosofia del best effort, cioè il sistema non garantisce che le operazioni vadano a buon fine. Il termine Quality of Service (QoS) si riferisce alla capacità di una infrastruttura di differenziare il servizio fornito da più classi di traffico concorrentemente presenti sulla rete. Nel contesto dei Web Server i Service Provider hanno la necessità di rispettare livelli di servizio concordati con i loro utenti (Service Level Agreements) e contemporaneamente minimizzare i costi di esercizio dell'infrastruttura. Il costo più rilevante da minimizzare è il dispendio energetico. Il consumo energetico di un rack è cresciuto da 1kW nel 2000 a 8kW nel 2006 per arrivare a 20kW nel 2010. Questa voce peserà per oltre il 40% sul budget delle tecnologie enterprise nel 2012. Gli obiettivi della Quality of Service relativi alle web application sono:

- Stabilire in anticipo il livello di servizio che si otterrà dal server.
- Controllare l'accesso al server per privilegiare alcuni servizi.
- Fornire un accesso equo alle risorse del server.
- Massimizzare l'utilizzo delle risorse.
- Rendere predicibili i tempi di risposta.
- Minimizzare i costi energetici della cpu.

Gli strumenti e i metodi di base della QoS sono:

- Classificare e controllare l'accesso: le richieste vengono selezionate e smistate in una delle classi della QoS definite. La decisione può essere basata su molti parametri quali l'indirizzo IP sorgente e destinazione, il protocollo e la porta.

- Marcare: nel pacchetto della richiesta viene scritta o modificata l'informazione della QoS secondo la decisione presa nella fase precedente.
- Scartare: viene confrontato il traffico con il profilo concordato della Quality of Service e utilizzando un algoritmo di Admission Control (Figura 1.3) possono essere scartate alcune richieste a cui non si può garantire la QoS prevista. La decisione è presa sulla base dei dati acquisiti dallo schedulatore o su stime del worst case con gli attuali flussi.

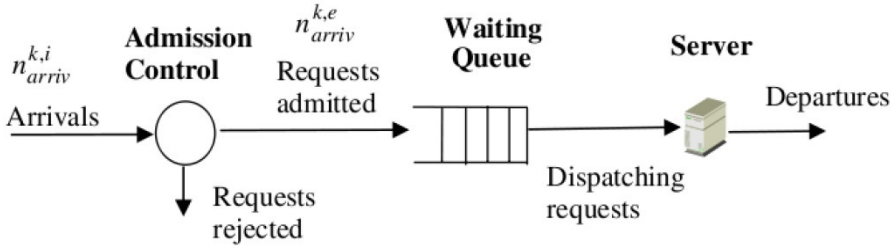


Figura 1.3 - Schema di un server a singola coda che implementa un algoritmo di admission control

Scheduling: consiste nella scelta di una tecnica per servire le richieste in funzione della loro classe. Le tecniche più note sono :

- Priorità Queueing: una classe ha priorità assoluta.
- Round Robin: è un accodamento differenziato, si esegue un ciclo di servizio tra code servendo un pacchetto per ogni coda.
- Class Queueing: riserva una percentuale fissa di capacità per ogni coda.
- Weighted Fair Queueing: divide la capacità in modo equo tra flussi mediante l'utilizzo di pesi. Se nella coda non sono presenti pacchetti la sua quota verrà divisa proporzionalmente tra le altre classi attive.

Ulteriori misure adottabili per garantire la Quality of Service consistono nell'evitare la congestione cioè evitare che il carico di una classe sia tale da causare lo scarto delle richieste; formare il segnale cioè garantire che la frequenza degli arrivi sia conforme ad una frequenza definita; raccogliere ed analizzare statistiche del servizio.

Per quanto riguarda il problema della riduzione dei costi energetici dei data center si utilizzano tecniche basate sull'accensione e lo spegnimento dei server in base al carico a cui è sottoposta la struttura. I

più recenti server si avvalgono del meccanismo del Dynamic Voltage Scaling (DVS). Tale tecnologia permette di variare la frequenza di esecuzione delle operazioni da parte delle cpu variando l'alimentazione del server. Il consumo energetico E(t) risulta proporzionale al cubo della frequenza di clock f(t)

$$E(t) = [f(t)]^3$$

mentre le performance del server variano linearmente con f(t).

I differenti approcci al problema sono organizzati in due categorie:

Tecniche di ottimizzazione basate sulla disponibilità delle risorse. Tipicamente il sistema è descritto tramite la teoria delle code. La variabile su cui è possibile agire è la disponibilità fornita ad un'applicazione di accedere ad una determinata risorsa. Si parla di politiche di admission control, resource allocation, load balancing. Tali tecniche però risultano essere efficaci in tempi medio - lunghi, non riuscendo a garantire una reattività adeguata del sistema alle variazioni improvvise di carico.

Approccio con controllo in retroazione. Queste tecniche permettono il controllo delle performance di un server in modo da renderlo capace di adattarsi dinamicamente alle condizioni di carico a cui è sottoposto.

E' possibile modellizzare in modo accurato i transitori di carico e si può intervenire sulla configurazione del sistema in tempi molto più brevi rispetto alla tradizionale teoria delle code basata su un modello statico.

1.3 La virtualizzazione

Una rapida crescita dell'infrastruttura, l'aumento dei costi energetici e degli immobili e le condizioni di mercato mutevoli sono delle vere e proprie sfide per il business d'oggi. Le aziende mirano a incrementare la propria business agility semplificando le infrastrutture IT e sfruttando al massimo le risorse di calcolo. Il management richiede soluzioni con un time to customer sempre più veloce e con una continua riduzione dei costi. Una delle odierne tecniche di consolidamento è la virtualizzazione ossia la possibilità di eseguire simultaneamente sistemi operativi e applicazioni differenti sul medesimo server, partizionando le risorse del sistema. In questa tesi si è fatto uso di sistemi virtualizzati cercando di modellarne il comportamento. Qualunque risorsa (Figura 1.6) hardware o software può essere virtualizzata: sistemi operativi, memoria, spazio disco. Ad oggi la virtualizzazione può essere effettuata a livello software e a livello hardware. Tra gli impieghi della virtualizzazione il più utilizzato è

probabilmente la virtualizzazione di sistemi operativi. Via software è necessario un sistema operativo in esecuzione (host) che esegua un software di virtualizzazione. Quest'ultimo crea ad alto livello le virtual machine (guest) che vengono eseguite come se fossero dei normali programmi e comunicano con l'hardware solo indirettamente, tramite il software di virtualizzazione che agisce a basso livello.

Ogni Virtual Machine (VM) si comporta come un singolo server stand-alone, che viene dedicato ad una particolare esecuzione ed è indipendente dall'hardware su cui è eseguita poichè la VM agisce su un'astrazione delle risorse fisiche del server. E' quindi possibile avere in esecuzione contemporaneamente sullo stesso server fisico più VM gestite dal sistema di virtualizzazione installato. Tale sistema viene chiamato VMM (Virtual Machine Monitor) detto anche hypervisor. Il VMM può decidere l'allocazione delle risorse alle Virtual Machine o addirittura trasferire una VM attiva a un altro server. Ad esempio VMWare e KVM permettono il trasferimento anche di una macchina attiva (online Migration) ad un sistema con la medesima architettura; in alternativa la migrazione può essere comunque eseguita sospendendo il servizio e trasferendo la VM (offline Migration).

1.3.1 Vantaggi e svantaggi delle tecniche di virtualizzazione

I principali vantaggi di questo approccio possono essere classificati in cinque macro categorie:

1. Consolidamento: oltre alla già citata riduzione dei costi dell'hardware, la virtualizzazione porta anche un aumento e ad un'ottimizzazione dell'utilizzo delle risorse fisiche della macchina; consente di incapsulare e fornire supporto al software legacy e sviluppare applicazioni per hardware dedicato.

2. Affidabilità: essendo ogni singolo sistema indipendente i bug applicativi rimangono isolati e le risorse possono essere riallocate dinamicamente a seconda delle esigenze delle VM.

3. Sicurezza: L'isolamento delle macchine virtuali permette facilmente di confinare i vari attacchi. Inoltre lo sviluppo delle policy di sicurezza risulta semplificato dalla presenza dell'hypervisor. Risulta più semplice rimpiazzare sistemi infetti o compromessi e fornire ambienti di lavoro remoti. Un sistema virtualizzato inoltre fornisce la

possibilità di testare applicazioni potenzialmente dannose in un ambiente controllato.

4. Business Agility: i processi vengono disaccoppiati dalle risorse fisiche pertanto l'organizzazione può ridisegnare rapidamente i propri processi.

5. Downtime ridotto: aumenta l'availability del sistema in quanto in caso di down di una VM le altre continuano a funzionare. E' semplice e veloce spostare macchine virtuali ibernate e ripristinare VM compromesse.

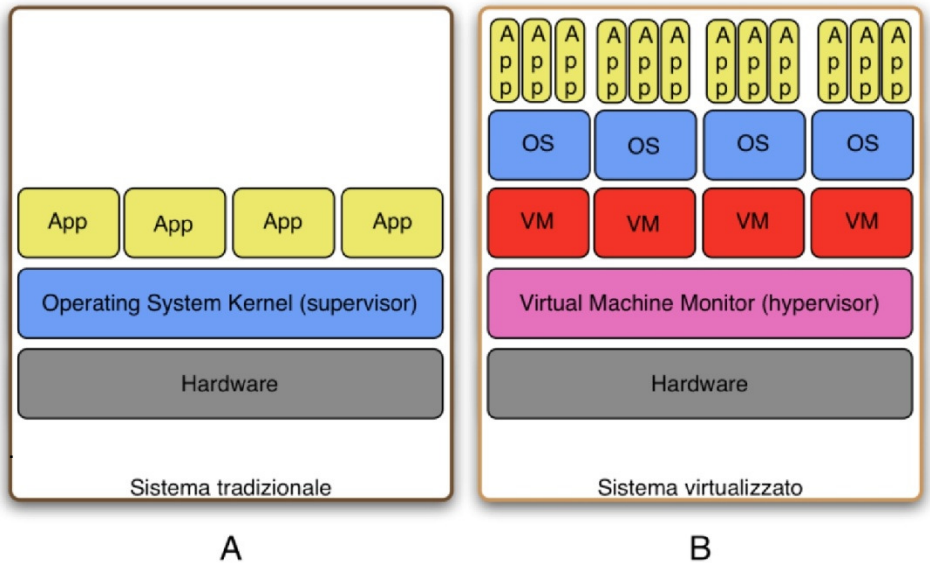


Figura 1.4 - Confronto fra un sistema tradizionale (non virtualizzato) ed uno virtualizzato

Esistono però dei punti deboli nelle tecniche di virtualizzazione, ad esempio l'ampliamento della visione della infrastruttura IT rende più difficoltosa l'allocazione delle risorse e il calcolo dei costi; la riduzione dei costi hardware viene compensata dagli alti costi di licenza se si fa uso di architetture proprietarie.

L'elevata dinamicità dei sistemi virtualizzati rende complessa la gestione globale dell'infrastruttura e il costo della migrazione a sistemi virtuali può risultare elevato.

1.3.2 Struttura di un sistema virtualizzato

La virtualizzazione di un sistema comporta l'astrazione delle sue risorse hardware e modifica il modo in cui il sistema operativo, gli applicativi e l'utente interagiscono con tali risorse. L'hypervisor crea le macchine virtuali che possono essere utilizzate da ospiti, tipicamente sistemi operativi. Il compito del VMM è quello di fornire al sistema ospite repliche delle componenti hardware che il kernel richiede.

Si possono perciò classificare i VMM in due tipologie:

VMM di tipo 1: sono eseguiti direttamente sull'hardware fisico. Questa tipologia è utilizzata tipicamente in ambito server per perseguire obiettivi di consolidamento.

VMM di tipo 2: sono eseguiti all'interno di un sistema operativo ospitante. Trovano maggior applicazione in ambito desktop/workstation.

In un sistema virtualizzato l'hypervisor è frapposto tra l'hardware e i singoli kernel. Ciascun sistema operativo pertanto è a contatto con una macchina virtuale simulata dal VMM; ognuna di queste macchine ha associato un proprio contesto di esecuzione, cioè la configurazione delle risorse del sistema il cui stato è dipendente dalla VM in esecuzione. Quando un'applicazione necessita di comunicare con l'hardware questa viene filtrata dal VMM che si preoccupa di gestire opportunamente l'evento in maniera totalmente trasparente al sistema Guest.

1.3.3 I requisiti di Popek e Goldberg

In un articolo del 1974 G.J. Popek e R.P.Goldberg, hanno stabilito le condizioni che permettono di creare architetture virtualizzate complete ed efficienti. I requisiti dell'ambiente creato dall'hypervisor devono godere delle seguenti proprietà:

- Equivalenza: un'applicazione eseguita in una Virtual Machine deve comportarsi come se fosse eseguita su una macchina fisica.
- Controllo delle risorse: il VMM deve mantenere pieno controllo delle risorse da esso virtualizzate.
- Efficienza: il Virtual Machine Monitor deve essere eseguito solo quando è strettamente necessario.

Per fare in modo che una determinata architettura generi Virtual Machine che soddisfino le proprietà sopra elencate è necessario fare ulteriori considerazioni sulle sue caratteristiche. Le istruzioni di una architettura si dividono in Privileged, cioè le istruzioni che generano una trap quando

vengono eseguite da una applicazione; istruzioni Control Sensitive, cioè quelle che possono modificare il contesto del sistema; ed infine istruzioni Behaviour sensitive, cioè quelle il cui comportamento dipende dal contesto di esecuzione. E' possibile ottenere macchine virtuali che rispettino tutte le proprietà sopra elencate solo se le istruzioni Control e Behaviour sensitive rappresentano un sottoinsieme di quelle Privileged. Pertanto se tutte le istruzioni che espongono in qualche modo il contesto di esecuzione o che ne dipendano generano una trap, l'architettura rispetta le tre proprietà.

I requisiti di Pope e Goldberg sono sufficienti ma non del tutto necessari poiché è possibile sviluppare Virtual Machine Monitor funzionanti anche in mancanza del totale soddisfacimento di questi. Ad esempio nell'architettura x86 viene spesso sacrificata l'efficienza con un numero di interventi del VMM molto più grande del necessario, al fine di rendere il sistema più flessibile.

1.3.4 Tipologie di virtualizzazione

Per poter sviluppare dei Virtual Machine Monitor funzionanti è necessario introdurre delle modifiche nel software o nell'hardware in quanto le architetture esistenti hanno dei difetti congeniti che le rendono difficilmente virtualizzabili in maniera nativa. Le soluzioni a questo problema sono molteplici e nel seguito verrà proposta una panoramica dei vari approcci esistenti.

1.3.4.1 Virtualizzazione completa

Le modifiche da apportare a un sistema operativo per renderlo virtualizzabile oltre ad essere complesse e costose, si scontrano anche con il problema della disponibilità del codice sorgente che in alcuni casi non viene rilasciato per ragione di marketing o di protezione del know-how. La virtualizzazione completa (Figura1.5) è una tecnica che aggira questo problema consentendo di ottenere un ambiente simile ad uno nativamente virtualizzabile senza bisogno di modificare il codice sorgente. Lo svantaggio di questa metodologia è la complessità del VMM il quale deve farsi carico di tutte le procedure relative all'emulazione di una architettura nativamente virtualizzabile. La tecnica maggiormente utilizzata è la binary patching, cioè la modifica di alcune istruzioni contenute nel software durante la sua esecuzione. L'hypervisor si occupa di monitorare il flusso di esecuzione sostituendo con delle trap le istruzioni che potrebbero creare dei problemi prima che esse vengano eseguite (Ring deprivileging). Il sistema operativo ospite viene comunque modificato, non a livello di codice sorgente, ma a runtime. Questo

meccanismo provoca un evidente degrado delle prestazioni e rallenta l'esecuzione delle Virtual Machine. Pertanto la proprietà di efficienza e Popek e Goldberg non viene rispettata. Un esempio di VMM che utilizza questa tecnica è VMWare.

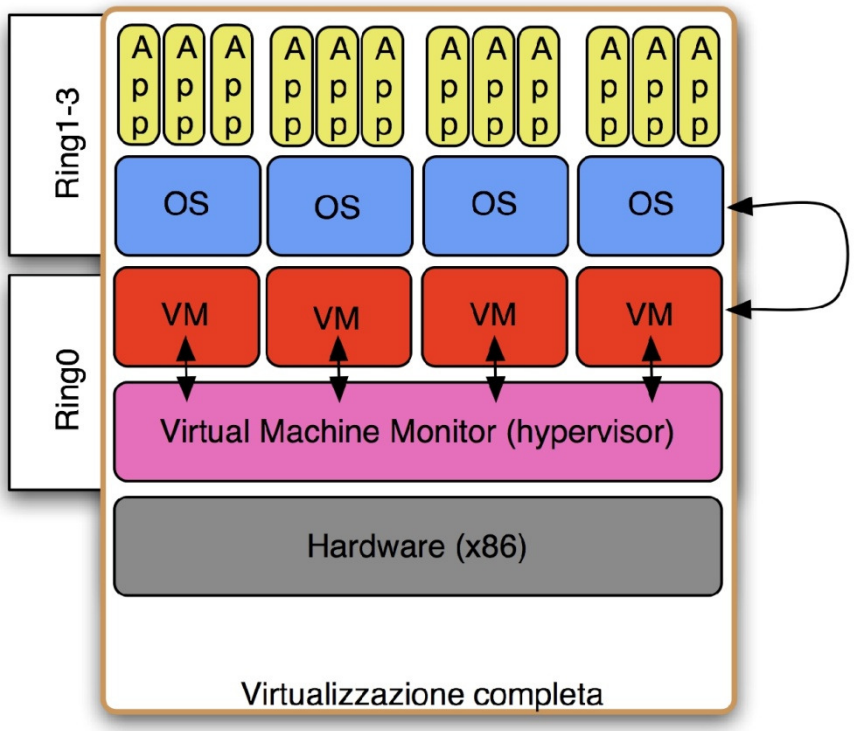


Figura 1.5 - Virtualizzazione completa

1.3.4.2 Paravirtualizzazione

L'utilizzo di Virtual Machine Monitor complessi e binary patching introducono un decadimento prestazionale notevole. Per ovviare a questo inconveniente si può optare per la modifica diretta del codice sorgente dei sistemi operativi ospitati. Ciò esclude la possibilità di utilizzare sistemi proprietari il cui codice sorgente non viene reso disponibile. In questo caso si parla di paravirtualizzazione (Figura1.6). Un hypervisor che implementa tale tecnica si limita ad esporre delle API. I sistemi operativi ospitati necessitano di una modifica tale da eliminare le interazioni dirette con l'hardware sostituendole con l'utilizzo delle funzioni fornite dalle API. I vantaggi di questo approccio sono di natura sia prestazionale che implementativa. Il VMM risulta meno intrusivo non dovendo emulare

dinamicamente il comportamento dell'hardware.

Questo rende le prestazioni delle macchine guest molto vicine a quelle di una macchina fisica anche per la velocità di accesso ai dispositivi di I/O fisici. Gli svantaggi della paravirtualizzazione sono la bassa portabilità e la difficile retrocompatibilità delle VM. Il VMM più noto che supporta la paravirtualizzazione è Xen.

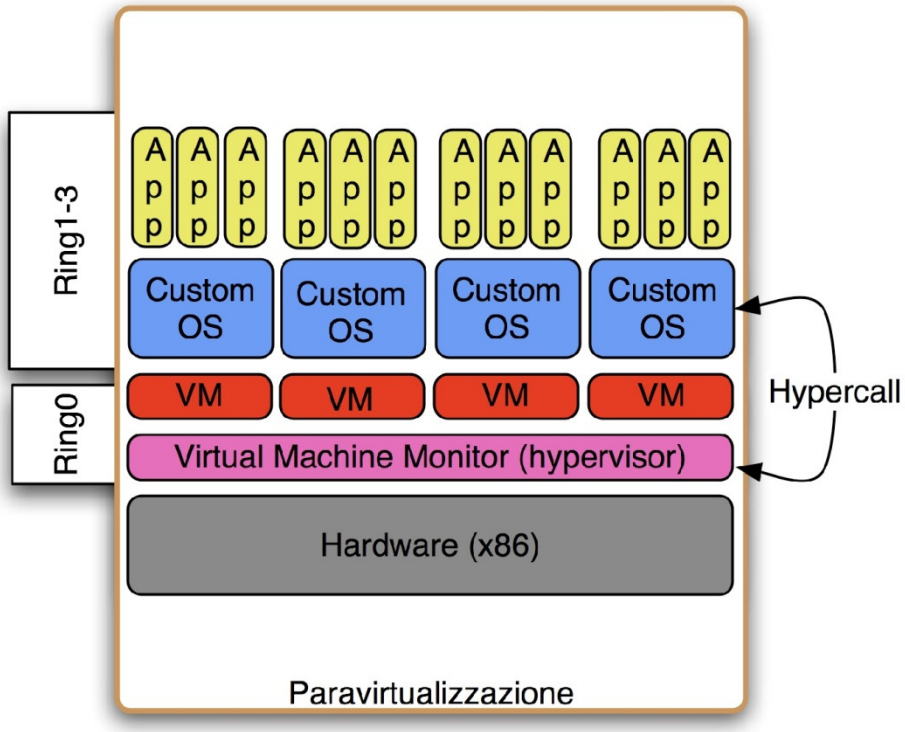


Figura 1.6 - Paravirtualizzazione

1.3.4.3 Virtualizzazione nativa

Il problema del decadimento prestazionale introdotto dalla virtualizzazione completa può essere affrontato anche intervenendo sull'hardware. I maggiori produttori di cpu x86 hanno perciò esteso il set di istruzioni delle loro architetture estendendolo con funzionalità che consentano al Virtual Machine Monitor di operare in maniera più efficiente. In particolare Intel ha sviluppato la tecnologia Intel-VT (Vanderpool Technology) mentre AMD ha proposto una soluzione concorrente rilasciata con il nome di AMD-V. Entrambe sono soluzioni proprietarie ed incompatibili tra loro, ma si basano sui medesimi concetti. I problemi causati dal ring deprivileging vengono risolti con l'introduzione

di due nuove modalità di esecuzione: root mode e non-root mode per i quali sono disponibili tutti i livelli di privilegio. Il root mode è del tutto simile alla normale modalità operativa ed è flessibile in quanto permette al VMM di utilizzare tutti i livelli di privilegio, mentre il software eseguito in non-root mode ha sempre minori privilegi. Nel non-root mode inoltre la maggior parte delle istruzioni può generare una trap permettendo all'hypervisor di gestire al meglio la situazione.

L'efficienza di questo approccio è sicuramente buona, ma in certi casi ancora inferiore a quella dei più moderni Virtual Machine Monitor paravirtualizzati che hanno raggiunto elevati livelli di ottimizzazione.

Bisogna anche tener conto che i produttori si stanno adoperando per migliorare queste tecnologie con l'obiettivo di giungere ad una virtualizzazione completamente assistita dall'hardware. Esempi di VMM che supportano la virtualizzazione nativa sono VMWare, Xen dalla versione 3.0 e KVM

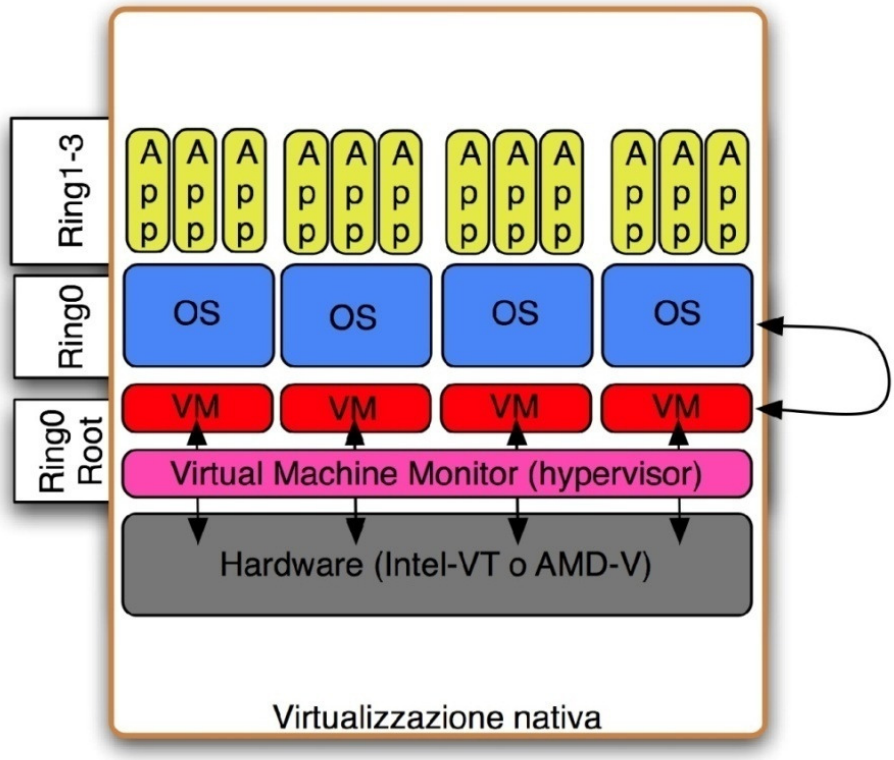


Figura 1.7 - Virtualizzazione nativa

1.3.4.4 Virtualizzazione a livello di sistema operativo

Questo tipo di virtualizzazione (Figura1.8) elimina il concetto di guestOS introducendo invece dei private server. Questa tecnica non simula repliche dell'hardware, ma consente l'esecuzione di più ambienti virtuali che si appoggiano su un unico kernel. Ciò permette di ridurre i problemi legati all'esecuzione di molteplici kernel fornendo ottime prestazioni. Questi vantaggi sono però attenuati dalla scarsa flessibilità della soluzione in quanto questa tecnica obbliga ad utilizzare un unico tipo di sistema operativo e introduce un overhead computazionale non trascurabile.

La virtualizzazione a livello di sistema operativo non è da considerarsi una vera e propria tecnica di virtualizzazione, ma è molto utilizzata negli hosting center che offrono server privati virtuali riuscendo a raggiungere fino a diverse centinaia di ambienti virtuali per macchina fisica. L'obiettivo di queste tecniche di consolidamento è la riduzione dei costi di gestione e di impianto dell'hardware.

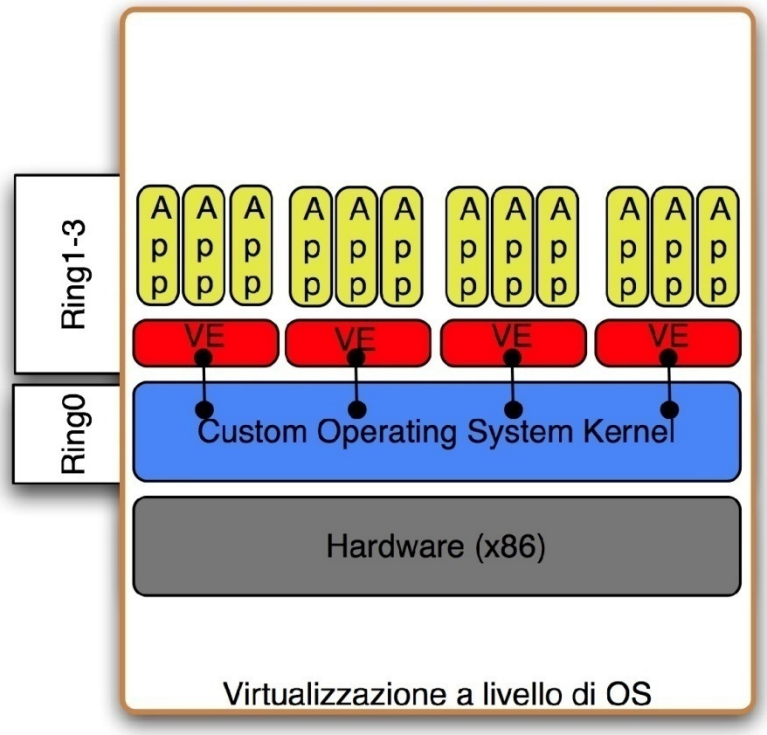


Figura 1.8 - Virtualizzazione a livello di sistema operativo

Riportiamo in Tabella 1.1 una comparazione delle tecniche di virtualizzazione discusse.

	Virt. completa	Paravirt.	Virt. Nativa	Virt. a livello di OS
Richiede la modifica degli OS	NO	SI	NO	
Supporta qualsiasi OS ospite	SI	NO	SI	NO
VM eterogenee	SI	SI	SI	NO
Alta efficienza	NO	SI	SI	SI
Richiede HW particolare	NO	NO	SI	NO

Tabella 1.1 - Tabella comparativa delle tecniche di virtualizzazione

1.3.5 Xensource XenExpress

Xen è un hypervisor nato da un progetto di ricerca dell'università di Cambridge, è open source e quindi disponibile gratuitamente. Ian Pratt, il professore a capo del progetto, ha fondato Xensource, società che vende e supporta una versione enterprise di Xen. Il solo Xen non è sufficiente a costituire una soluzione di virtualizzazione completa a livello enterprise, infatti è spesso incluso all'interno di distribuzioni GNU/Linux (come Red Hat, Suse e Debian) nelle quali è affiancato da svariati tool per consentirne una gestione efficiente, oppure viene incluso in soluzioni commerciali complete, come Xensource o ad esempio Virtual Iron. La prima versione di Xen al pubblico nel 2003 era essenzialmente un software per interfacciare macchine virtuali con l'hardware dell'host. Per questo motivo era necessario modificare i sistemi operativi Linux per poterli eseguire come guest su server Xen. Fino alla versione 2.0 del 2005 questo hypervisor non ha riscosso grande successo nel mondo delle imprese, in quanto non supportava sistemi multiprocessore e il deployment era particolarmente complesso. Con il rilascio della versione 3.0 nel 2006 sono state introdotte importanti novità che hanno portato Xen a diventare competitivo nel settore Enterprise diventando uno tra i competitor più affermati come VMWare.

Sin dalla prima versione Xen utilizza esclusivamente tecniche di paravirtualizzazione per la creazione delle VM. Consente quindi la sola esecuzione di sistemi operativi debitamente modificati, fra cui spiccano svariate versioni di GNU/Linux, NetBSD, OpenBSD, FreeBSD e OpenSolaris.

Col rilascio da parte di Intel e AMD di processori con estensioni hardware alla virtualizzazione, quest'ultime hanno dimostrato un notevole interesse nel progetto Xen, contribuendo all'integrazione nell'hypervisor del supporto alle proprie estensioni. A partire dalla versione 3 di Xen, la paravirtualizzazione viene affiancata dalla virtualizzazione nativa, consentendo quindi di eseguire qualsiasi sistema operativo senza bisogno di modifiche.

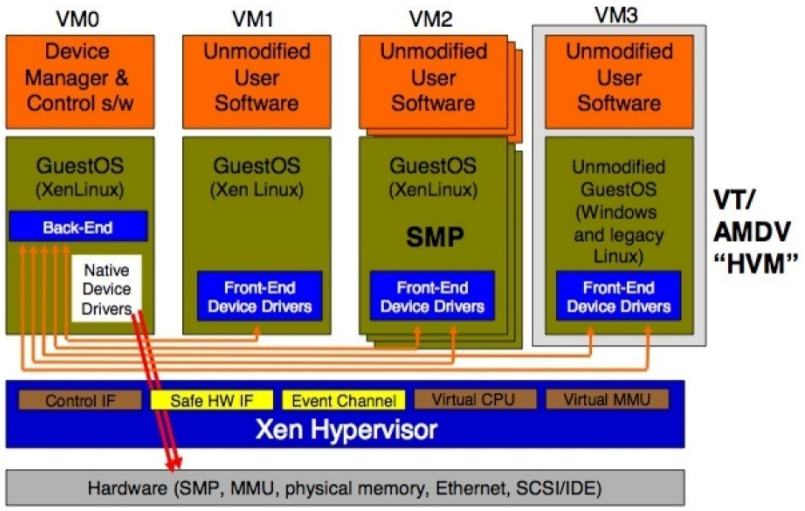


Figura 1.9 - Infrastruttura di Xen

L'infrastruttura di Xen è mostrata in Figura 1.9. L'architettura prevede diversi livelli con privilegi decrescenti. Il massimo dei privilegi è garantito all'hypervisor. Nella terminologia di Xen ogni VM è chiamata domain; ogni dominio gestisce le applicazioni dei propri utenti in modo da schedulare la loro esecuzione nel tempo di CPU garantito dall'hypervisor.

1.3.6 Kernel based Virtual Machine

Il progetto Kernel-based virtual machine (KVM) è una tecnologia di virtualizzazione opensource di ultima generazione.

L'obiettivo del progetto è di creare un nuovo e moderno hypervisor che unisca l'esperienza delle tecnologie di virtualizzazione delle generazioni precedenti (tra le quali Xen) e le potenzialità fornite dal nuovo hardware.

KVM, come mostrato in figura 1.10, è implementato da un modulo che inserito nel kernel lo integra perfettamente nel sistema operativo e lo rende di fatto un hypervisor di tipo nativo.

Due obiettivi fondamentali hanno guidato lo sviluppo di questo progetto:

- Essendo nato dopo lo sviluppo delle tecnologie hardware che assistono la virtualizzazione, quali intel-VT e AMD-V, KVM è stato progettato attorno ad esse e sfrutta al meglio le loro potenzialità.

- Per molti aspetti un hypervisor è simile a un sistema operativo, infatti oltre a virtualizzare le risorse come CPU e memoria ha bisogno di sheduler, device driver, memory manager, gestore di I/O, politiche di sicurezza ecc. Dato che il kernel Linux offre già queste funzionalità è molto più efficiente costruire KVM su queste basi piuttosto che riprogettarle da zero.

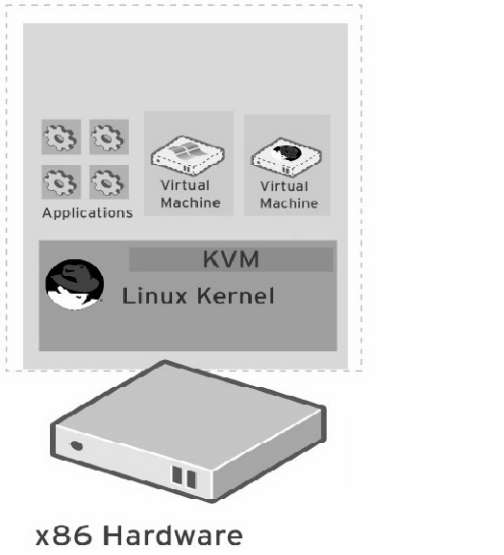


Figura 1.10 - Architettura di KVM

Nell'architettura di KVM quindi le virtual machine si comportano esattamente come un normale processo Linux, e vengono eseguite come tali.

L'emulazione delle risorse è gestita da una versione modificata di QEMU che provvede ad emulare, oltre alle periferiche standard come hard disk IDE o SCSI e schede di rete, anche BIOS e bus PCI e USB.

Come mostra la figura 1.11 KVM offre anche funzionalità di condivisione della memoria, una funzionalità del kernel, chiamata Kernel Same-page Merging (KSM), si occupa di controllare, scandendo la memoria allocata, se due o più virtual machine caricano pagine identiche. Se questo accade

viene tenuta solo una copia di queste pagine e viene messa in condivisione a tutte le virtual machine.

In questo modo diverse virtual machine possono essere consolidate in un unico host riducendo i costi hardware e ottimizzando l'utilizzo del server.

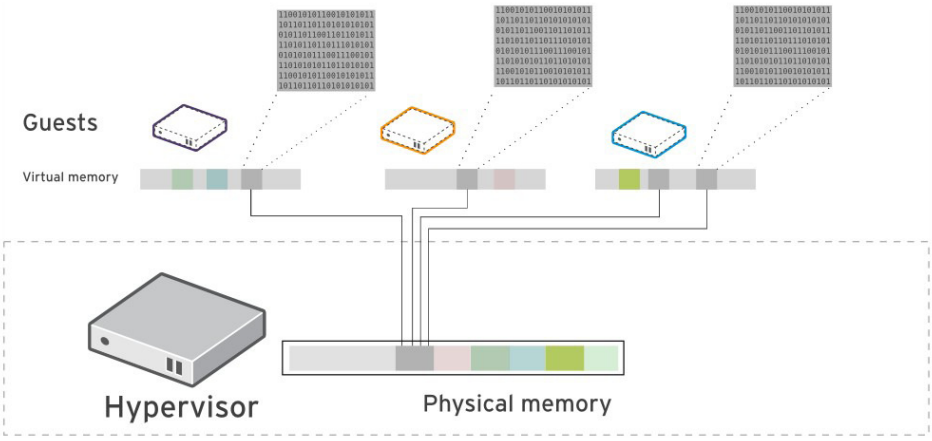


Figura 1.11 Condivisione della memoria in KVM

Oltre alla normale migrazione (offline migration) kvm supporta la live migration che consente di migrare una virtual machine da un host ad un altro senza doverne interrompere l'esecuzione, in modo totalmente trasparente all'utente.

1.3.7 Ubuntu Linux

Il sistema operativo con cui sono stati effettuati i test di prestazioni eseguiti al capitolo 6 è basato su Ubuntu Server Edition 10.04. Ubuntu è una distribuzione GNU/Linux, un progetto Open Source sviluppato dalla community. L'obiettivo è quello di lavorare con la community di GNU/Linux per creare un sistema operativo completo, utilizzando esclusivamente software liberi e forum pubblici con processi aperti. Questo porta a un continuo aggiornamento delle distribuzioni che è diventato molto semplice, anche da una distribuzione all'altra. La distribuzione offre un buon supporto alla virtualizzazione, vengono anche fornite applicazioni di gestione delle VM guest. Virt-manager permette il controllo da remoto di più server anche attraverso l'utilizzo di VNC Server che consente di gestire le virtual machine utilizzando un'interfaccia grafica. Virt-manager fornisce inoltre il supporto grafico per la maggior parte delle operazioni che possono essere compiute sui server virtualizzati, dal dimensionamento della memoria e della quantità di CPU

utilizzata dalle VM fino alle impostazioni della rete virtuale. La rete virtuale rappresenta uno dei nodi fondamentali per la creazione di un'architettura virtualizzata, la creazione di un bridge virtuale permette di assegnare un indirizzo IP di rete locale alle Virtual Machine dando possibilità di accedervi tramite LAN in maniera totalmente trasparente per il client.

1.4 Lo scheduler di Linux

Poiché GNU/Linux è un sistema multitasking, deve gestire la presenza contemporanea di più processi eseguiti contemporaneamente sulla stessa macchina. Questo però non è fisicamente possibile poiché, generalmente, le macchine hanno una sola CPU ed una CPU può eseguire soltanto un'operazione alla volta. Viene adottata quindi la tecnica di assegnazione di intervalli di tempo (time slice) da dedicare ad ogni singolo processo. In questo modo un processo viene eseguito fintantoché il time slice ad esso riservato non è terminato, dopodiché viene sospeso. Il sistema poi determina quale sia il prossimo processo da attivare ed il time slice da assegnargli. E così via.

Lo scheduler è la parte del kernel che si preoccupa di gestire l'attivazione e la sospensione dei processi secondo una determinata logica detta anche politica di scheduling. La scelta del meccanismo in grado di distribuire l'opportuno time slice ai vari processi non è un problema semplice ed è importante notare il fatto che non esiste un sistema ottimale per la temporizzazione dei processi, ma dipende in maniera essenziale dall'utilizzo che si intende fare del sistema.

La politica di scheduling di Linux si propone il raggiungimento dei seguenti obiettivi (molti dei quali sono in contrasto):

- timesharing
- gestione di priorità dinamiche
- avvantaggiare i processi I/O-bound
- tempi di risposta bassi
- throughput elevato per i processi in secondo piano
- evitare starvation

L'algoritmo di scheduling cerca di raggiungere i suddetti obiettivi attraverso

- una suddivisione del tempo in epoche
- dipendenza della priorità dal residuo del quanto di tempo
- definizione delle condizioni che determinano l'invocazione dello scheduler

In Linux lo scheduling si basa sul concetto di timesharing, per cui ad ogni processo è assegnato un quanto di tempo massimo per l'esecuzione. La selezione del prossimo processo da eseguire è basata sul concetto di

priorità; questa può essere dinamica o statica. In particolare, la prima è introdotta per evitare il fenomeno della starvation, mentre la seconda è stata introdotta per consentire la gestione di processi real-time. Ai processi real-time è assegnata una priorità maggiore di quella assegnata ai processi "ordinari".

Di norma Linux prevede uno scheduling preemptive, per cui ad un processo viene tolta la CPU se: esaurisce il quanto di tempo a sua disposizione oppure un processo a priorità più alta è pronto per l'esecuzione.

Il valore iniziale del quanto di tempo assegnato ad un processo, memorizzato nell'attributo priority del descrittore, è tipicamente di 20 clock ticks, o 210 millisecondi. Questo è il più grande valore ammissibile senza che sia compromessa la reattività del sistema. La durata del quanto assegnata ad un processo ne identifica la priorità. Tramite le funzioni nice(int N) e renice(int N), si può variare il valore del quanto. Il valore di nice è un attributo numerico di ciascun processo dei sistemi Unix e Unix-like che è usato dallo scheduler per stabilire quanto tempo di CPU dedicare all'esecuzione del processo. A parità di priorità e di politica di schedulazione, i processi che hanno valori di nice maggiori ottengono in proporzione meno tempo di CPU rispetto a processi che hanno valori di nice minori, e quindi la loro esecuzione procede più lentamente, favorendo gli altri processi. Per diminuire il valore di nice è necessario disporre dei privilegi dell'administrator (root), mentre ciò non è necessario per aumentarlo. Tipicamente è possibile diminuire il valore di nice fino a 20 unità rispetto al valore predefinito, o aumentarlo fino a 19 unità.

1.5 SPECweb2005

SPECweb2005 è un'applicazione di benchmark di nuova generazione utilizzata per valutare le performance di web server. Come anche i suoi predecessori SPECweb99 e SPECweb99 SSL, SPECweb2005 continua la tradizione di fornire agli utenti il benchmark più oggettivo e rappresentativo di come un sistema possa agire da web server. In risposta al rapido avanzamento della tecnologia, SPECweb2005 include molte funzioni sofisticate e all'avanguardia, al fine di andare incontro alla domanda degli utenti di oggi e domani:

- Misura sessioni di utenti simultanee.
- Contenuti dinamici: l'implementazione include PHP e JSP.
- Le immagini di una pagina sono richieste usando due connessioni HTTP parallele.
- Workload diversi e standardizzati: Banking (HTTPS) che simula il funzionamento dei servizi offerti da una banca in rete, Ecommerce (HTTP e HTTPS), che simula un negozio di vendite online,

Support (HTTP), che simula l'accesso ad un catalogo online.

- Accesso ai file che rispecchia i metodi usati dai web server reali attuali supporto tramite sito ufficiale.
- Implementazione stabile.
- Java-based client per un codice più snello e portabile.

Il software, per ogni tipo di workload, è formato da diversi componenti, detti anche domain, ognuno designata a svolgere determinati compiti:

- Il componente specweb cli che svolge il compito di coordinatore delle operazioni e di iniettore di carico di lavoro vero e proprio.
- Il componente specweb web che svolge le funzioni di web server.
- Il componente specweb bes che si occupa di simulare le funzioni di back-end.

I generatori di carico di SPECweb2005 operano secondo un modello chiuso. Le sessioni degli utenti sono avviate a seconda del numero di utenti impostato e continuano ininterrottamente a inviare richieste di pagine dinamiche, attendono per un lasso di tempo in media pari a 10 secondi (THINK_TIME), quindi accedono ad una nuova pagina, oppure lasciano il sistema.

Il client di coordinamento inizializza gli altri sistemi, monitora i test in esecuzione e infine, al completamento del test, raccoglie i risultati. Il web server è il componente principale per valutare le performance del sistema, mentre il simulatore delle operazioni di back-end è usato per determinare il contenuto dinamico delle pagine, in quanto ha il compito di emulare il funzionamento di un database.

Ogni test ha un proprio file di configurazione contenente tutti i parametri necessari per il proprio avvio.

Per essere avviato un test ha bisogno di due componenti:

Specwebclient.jar: deve essere avviato per primo e svolge il compito di generatore di carico. Il processo può essere avviato semplicemente, oppure, come consigliato dagli sviluppatori di SPECweb, indicando la memoria minima e massima che esso può utilizzare quando è in funzione e i parametri di comportamento del garbage collector.

In questo lavoro di tesi il processo è stato lanciato soltanto con i limiti di memoria utilizzabili, tralasciando i parametri inerenti al GC.

Specweb.jar: è il test manager di SPECweb e viene avviato necessariamente dopo specwebclient.jar. Con l'avvio di specweb.jar ha inizio il test vero e proprio. Il processo rimane attivo fino alla conclusione del test e termina subito dopo. Come specwebclient.jar anche specweb.jar puo essere invocato semplicemente oppure con le opzioni descritte in precedenza. Anche in questo caso, nel nostro lavoro di tesi, specweb.jar è stato sempre avviato con i soli limiti di memoria utilizzabili dal processo.

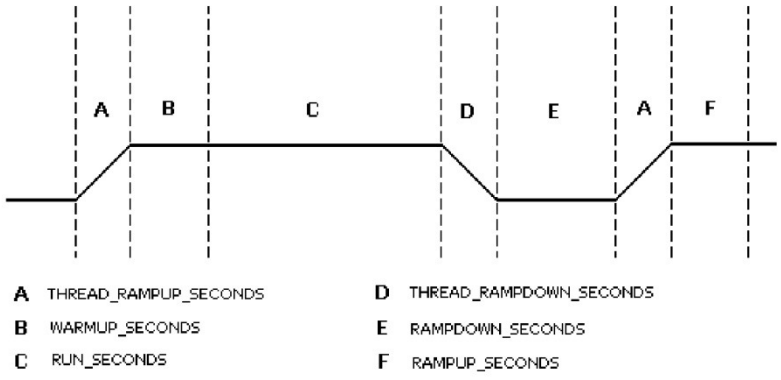


Figura 1.12 - Struttura del test di default di SPECweb

Una volta avviato, il test di default si compone di più fasi:

Thread Rampup Seconds A: periodo di tempo durante il quale il generatore di carico crea i threads, questa fase è implementata in questo modo per non iniziare il test con un picco di richieste che in un istante di tempo arrivino al massimo carico impostato.

Warmup Seconds B: come indica la parola è un periodo di riscaldamento, durante il quale il test è al massimo carico, ma non vengono prese statistiche ed eventuali errori in questa fase vengono ignorati.

Run Seconds C: intervallo di tempo durante il quale vengono presi i risultati del test.

Thread Rampdown Seconds D: questa fase è semplicemente l'esatto opposto della fase A, nonostante i threads continuino a inviare richieste al web server, le statistiche non vengono prese in quanto sono state stoppate alla fine della fase C.

Rampdown Seconds E: tempo fornito al client e al server per tornare al loro stato di riposo.

Rampup Seconds F: rimpiazza la fase B della prima iterazione, il sistema è già pronto e necessita di un tempo minore per raggiungere la fase di raccolta dei risultati.

Capitolo 2

Come funziona il tool

Nel presente capitolo verrà descritto il framework implementato per l'analisi delle prestazioni di web server virtualizzati. Questo tool è stato sviluppato a partire da lavori di tesi precedenti per analizzare le prestazioni dell'hypervisor Xen, è stato dunque necessario modificarlo, come descritto nel capitolo 3, per adattarlo alla nuova piattaforma di test basata su KVM.

Il presente capitolo è organizzato come segue: nel paragrafo 2.1 viene descritta l'architettura complessiva del framework. Nel paragrafo 2.2 vengono illustrate le operazioni di configurazione dei file di sistema. In fine, nel paragrafo 2.3, viene descritta la soluzione adottata per variare la priorità di esecuzione assegnata a ogni VM.

2.1 Struttura dell'architettura sperimentale

Il framework si basa principalmente su due componenti secondo il classico paradigma client/server: il server lavora sulla Virtual Machine e si occupa di eseguire i comandi ricevuti e di avviare i test di performance emulando un sistema web reale. Il client, invece, opera lato hypervisor e si occupa della configurazione del sistema e del coordinamento delle operazioni tra le VM attive.

La struttura generale del tool sviluppato è illustrata nella Figura 2.1.

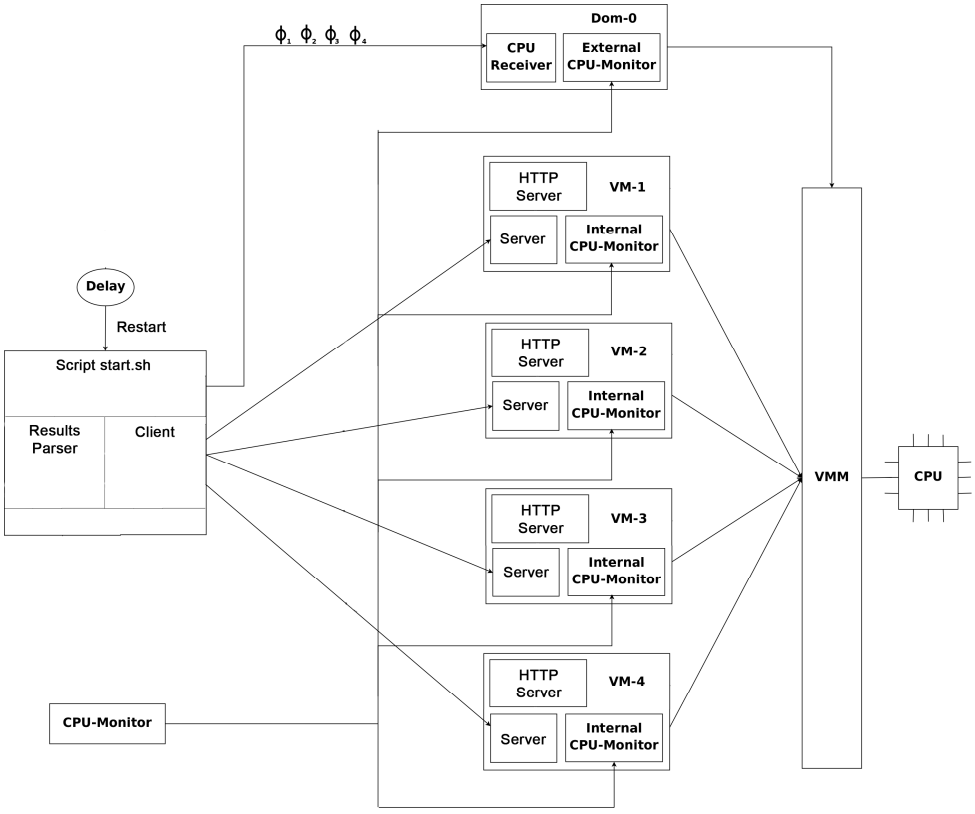


Figura 2.1 - Architettura del framework

Con test ad 1 VM e a 2 VM si intendono test effettuati con 1 e 2 workload ogni workload è costituito da:

- una Virtual Machine per il componente specweb cli
- una Virtual Machine per il componente specweb web
- una Virtual Machine per il componente specweb bes

Di conseguenza quando parleremo di 1VM o 2VM si sottointenderà l'insieme di queste tre Virtual Machine.

2.2 Configurazione del sistema

L'utente, come mostrato in figura 2.2, dopo essersi loggato sul server dove risiede il tool, si preoccupa di avviare le Virtual Machine necessarie all'esecuzione dei test e di modificare i file di configurazione dell'applicazione.

Il sistema legge i valori inseriti in questi file di testo e per ogni i-esima riga crea un nuovo file di configurazione, a partire da un file di template differente a seconda del tipo di workload (eCommerce, Banking o Support), andando a sostituire alle voci ITERATIONS, RAMPUP_SECONDS, RAMPDOWN_SECONDS, RUN_SECONDS e SIMULTANEOUS_SESSIONS il valore letto nei file di configurazione corrispondenti.

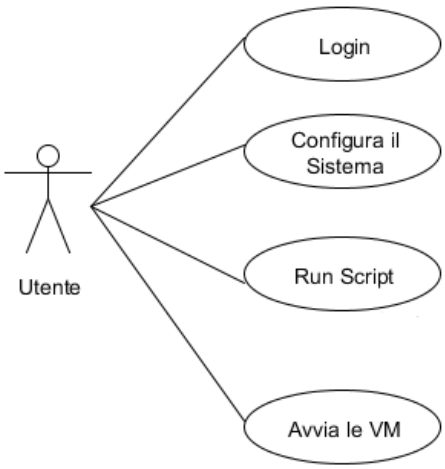


Figura 2.2 - L' utente esegue le operazioni preliminari

All'interno del file di template è possibile modificare altri parametri che, al contrario di quelli descritti sopra, rimarranno invariati per tutti i Test_i.config generati.

Una volta esaurite le operazioni preliminari, il lancio del tool è completamente automatizzato tramite uno script, che, come si vede dalla figura 2.3, si preoccupa di avviare l'HTTP Server e di lanciare il componente server all'interno di ogni VM.

In seguito lo script genera i file di configurazione dei test da eseguire, il file che verrà usato per l'assegnazione dei pesi delle CPU virtuali e infine lancia il componente client.

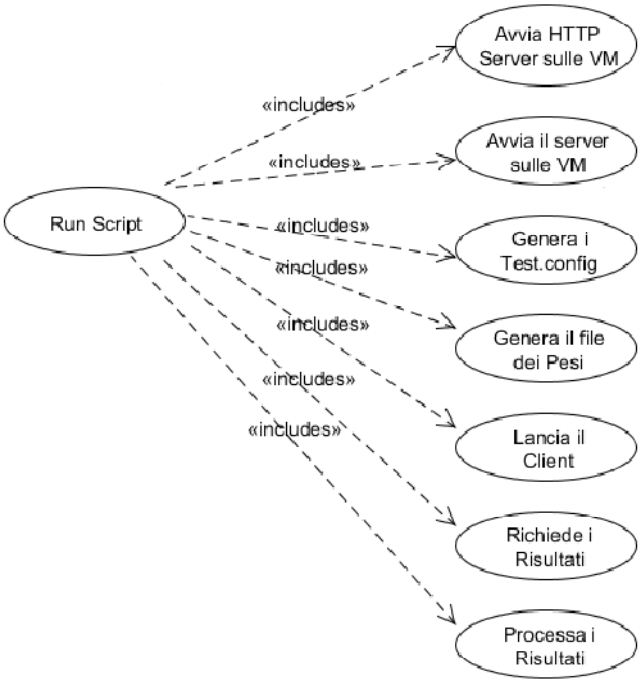


Figura 2.3 - Operazioni eseguite dallo script start.sh

La figura 2.4 mostra come non appena il client viene lanciato tutti i file Test_i.config vengono inviati alle VM attive e, dopo aver settato i pesi delle CPU virtuali, vengono eseguiti uno ad uno i test.

Per ogni test lanciato il client si occupa di effettuare il monitoring delle CPU virtuali rilevato da Domain-0 e di quello rilevato all'interno del web server virtuale.

Il client, quindi, si pone in attesa della risposta del server, che invierà per ogni test l'ack di test concluso.

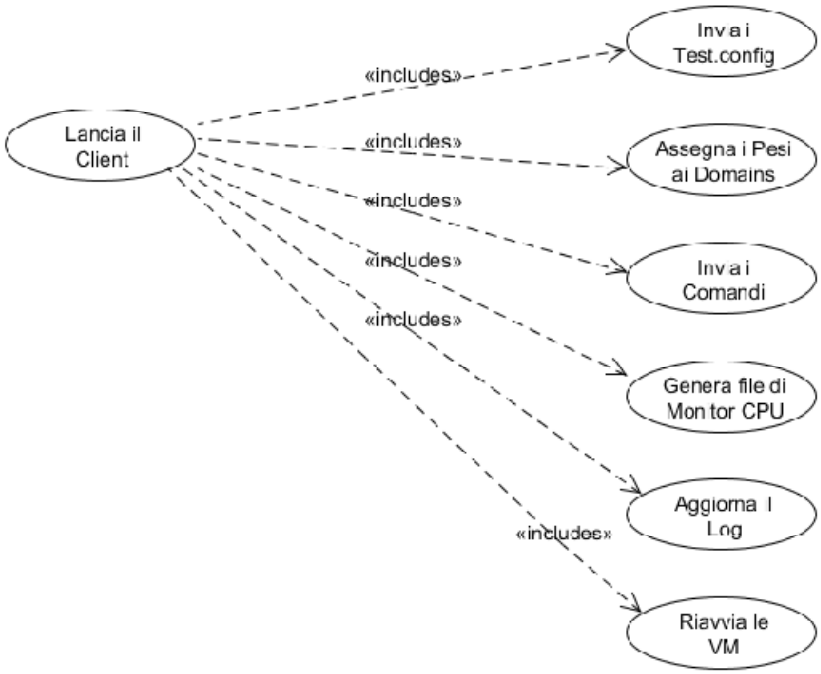


Figura 2.4 - Operazioni eseguite al lancio del client

Nel caso una VM non invii l'ack prima dello scadere del tempo di delay opportunamente impostato, significa che quel test è fallito.

A questo punto, come descritto in figura 2.5, il componente lato client si preoccupa di riavviare tutte le VM e di ripetere l'intera serie di test.

Questa operazione di reboot delle VM viene effettuata ogni volta che un test fallisce per un numero n di volte, dove n è configurabile dall'utente.

Se, invece, tutti i test si concludono correttamente, il componente client invia alle Virtual Machine un comando che arresta il componente server, dopo di che termina anch'esso.

Lo script si preoccupa quindi di copiare tutti i file dei risultati che risiedono sulle VM nelle corrispondenti cartelle su Domain-0.

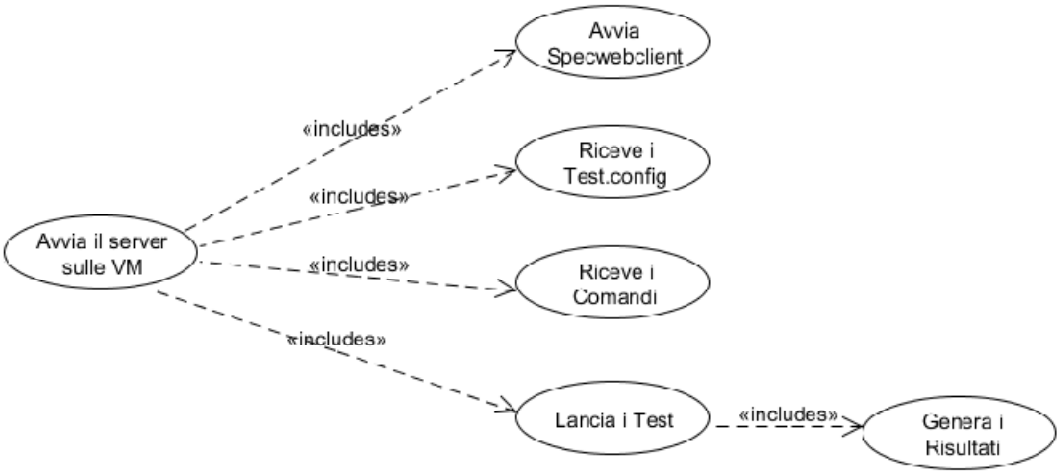


Figura 2.5 - Avvio e conclusione della fase di test

Concluse le operazioni di copia viene quindi avviato il tool ausiliario per l'elaborazione dei risultati.

Per ogni file di risultato, il tool ausiliario genera un file contenente la sola tabella delle statistiche di riepilogo del test e un file contenente l'ora di avvio del test, il numero di TOTAL_REQUESTS (le richieste totali fatte al web server virtuale) e l'AVG_RESPONSE_TIME (il tempo medio di risposta del web server). Inoltre, per avere un quadro generale di tutti i test, il tool crea un ulteriore file di testo contenente, in ordine temporale, le richieste totali e il tempo medio di risposta di tutti i test eseguiti.

Dato che la maggior parte dei test dura diverse ore, per visualizzare il loro andamento, le operazioni effettuate dal tool sono registrate in un file di log, contenente l'ora e il comando eseguito, oppure l'errore riscontrato.

In figura 2.6 è illustrato il class diagram del componente client che, eseguito dall'hypervisor, viene lanciato all'interno dello script che automatizza l'esecuzione del tool ed effettua le seguenti fasi:

1. Acquisizione dei dati di configurazione.
2. Invio dei file di configurazione dei test.
3. Comunicazione e coordinamento con le VM per il lancio dei test.

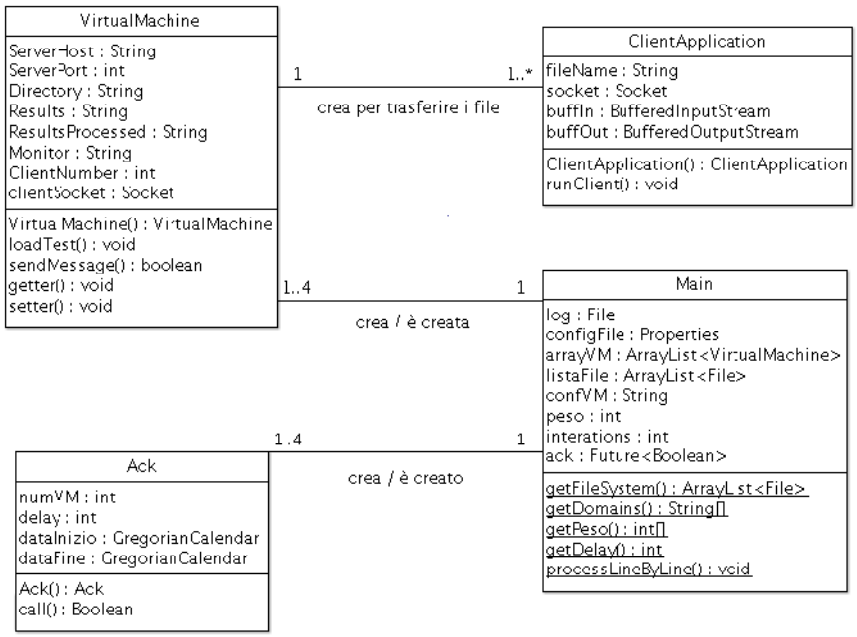
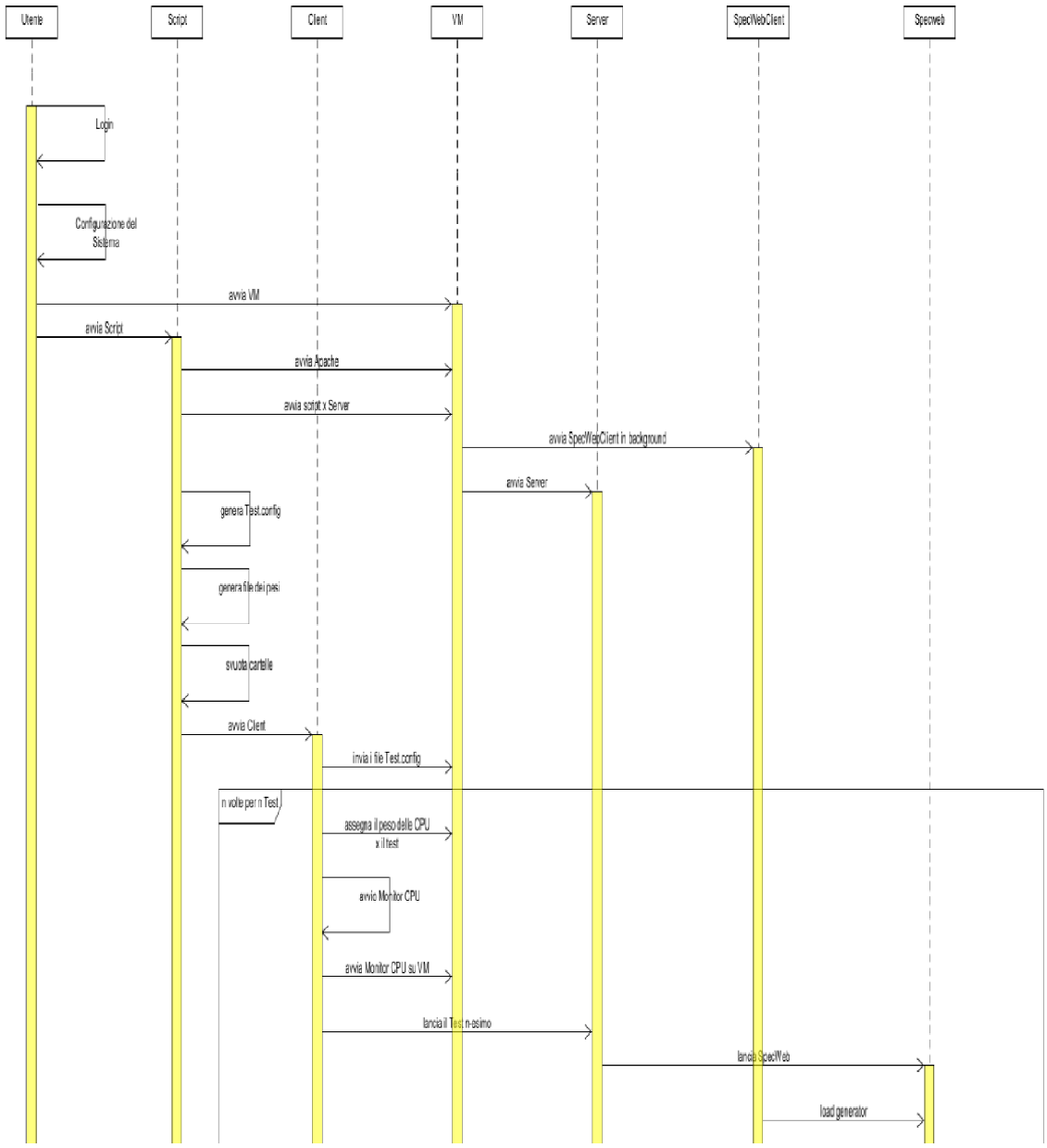


Figura 2.6 - Class Diagram del client

In figura 2.7 è mostrato il sequence diagram che descrive l'intero flusso di esecuzione de test. Inizia con fase di login e configurazione, seguita da quella di lancio effettivo del test e si chiude con la raccolta e analisi dei dati sull'host.


Utente
Script
Client
VM
Server
SpecWebClient
Specweb
Login
Configurazione del Sistema
avvia VM
avvia Script
avvia Apache
avvia script x Server
avvia SpecWebClient in background
avvia Server
genera Test.config
genera file dei pesi
svuota cartelle
avvia Client
invia i file Test.config
n volte per n Test
assegna il peso delle CPU x il test
avvia Monitor CPU
avvia Monitor CPU su VM
lancia il Test n-esimo
lancia SpecWeb
load generator

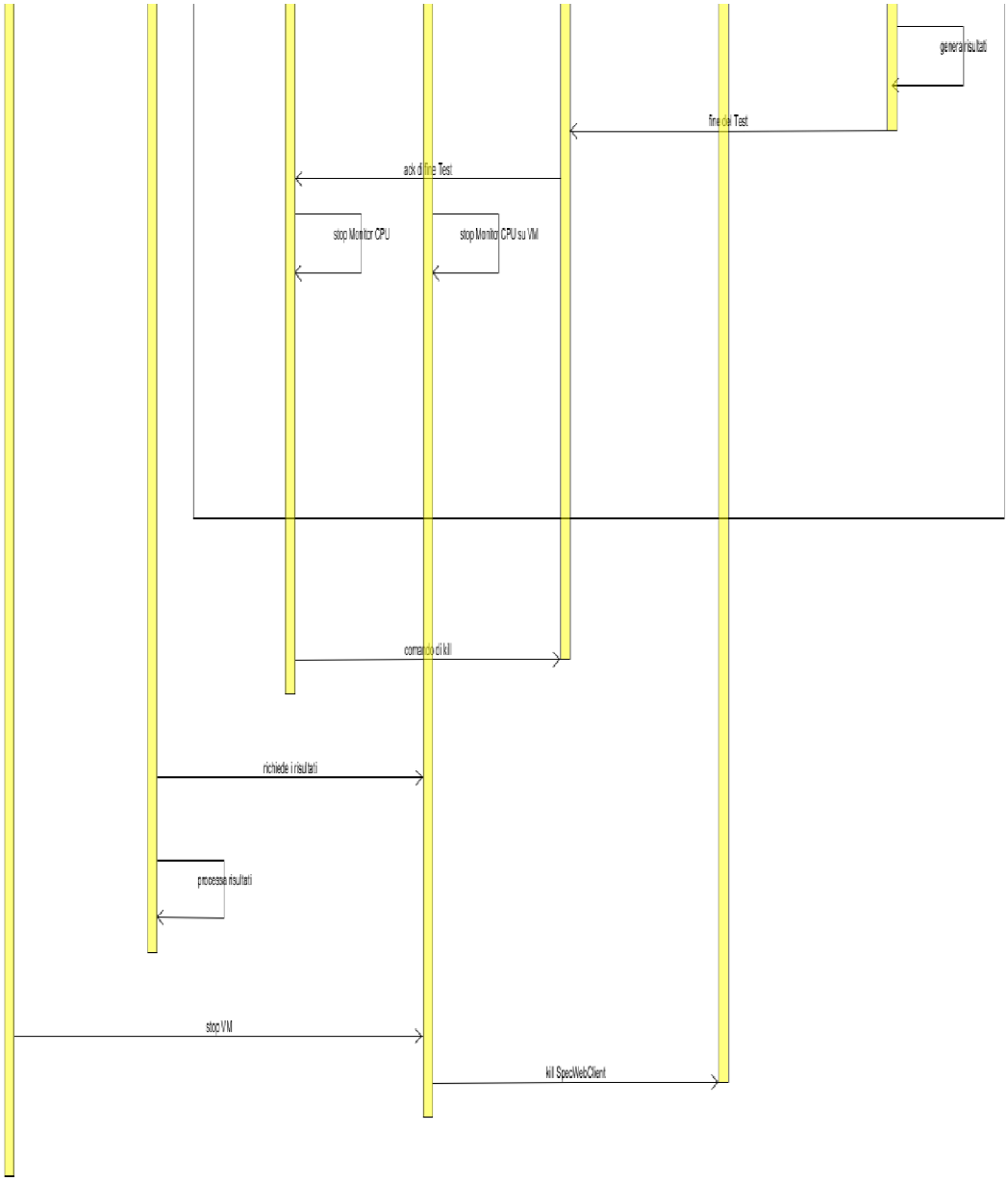


Figura 2.7 - Sequence diagram dell'intero test

2.3 Creazione del file dei pesi delle CPU

Una delle principali funzioni che deve avere il tool è quella di poter settare di volta in volta, per ogni test, i pesi da assegnare alle CPU virtuali nella contesa per l'accesso alle risorse della CPU fisica.

Attraverso questo sistema di gestione ad ogni CPU virtuale può essere assegnato un peso, che indica la probabilità che una certa VM venga servita in un determinato istante di tempo.

Per come è stato implementato, il file weight.cfg deve contenere tante righe quanti sono i test da effettuare. E' stato sviluppato un piccolo parser che crea, ad ogni avvio del tool, un nuovo file weight.cfg a partire dai file di configurazione phi.

Capitolo 3

Migrazione da Xen a KVM

Uno dei passaggi fondamentali di questo lavoro di tesi è stata la conversione delle immagini delle virtual machine da Xen, a KVM.

Nonostante vi siano diversi progetti, tra cui Fedora con virt-v2v e Red Hat, che stanno lavorando a un software per convertire immagini Xen in immagini KVM, di fatto non è ancora stato sviluppato un tool in grado di soddisfare questa esigenza, abbiamo dovuto quindi trovare un nostro metodo per convertire le immagini.

I problemi principali sono stati l'installazione di kernel e grub nelle nuove immagini, infatti visto che i comandi per installare kernel e grub devono agire su una device è stato necessario lanciare, tramite sistema virtuale, una system rescue che vede come device l'immagine KVM che vogliamo creare.

Dopo diversi tentativi la strada che si è rilevata vincente è stata:

I. Creare un' immagine grezza (.img) di dimensioni appropriate a ospitare l'immagine Xen da convertire, partizionarla in due parti, una parte dedicata alla logica primaria e una parte allo swap.

```
dd if=/dev/zero of=kvm.img bs=1M count=6000

parted kvm.img mklabel msdos

parted kvm.img mkpart primary ext2 0 5000

parted kvm.img mkpart primary linux-swap 5000 6000

parted kvm.img set 1 boot on
```

II. Montare l'immagine appena creata e copiare il contenuto dell'immagine Xen all'interno di essa.

```
losetup /dev/loop0 kvm.img

kpartx -a /dev/loop0

dd if=xen.img of=/dev/mapper/loop0p1 bs=1M

fsck.ext3 -f /dev/mapper/loop0p1
```

III. Convertire l'immagine di KVM da formato grezzo tramite VBoxManage in .vdi, unico formato utilizzabile con l'ambiente di virtualizzazione VirtualBox.

```
VBoxManage convertdd kvm.img kvm.vdi
```

IV. Avviare con VirtualBox una System Rescue che vede come device l'immagine .vdi appena convertita e montarla opportunamente. La soluzione migliore è utilizzare una versione Live della distribuzione dell'immagine Xen. Nel nostro caso, una versione live di Ubuntu Server Edition 8.04.

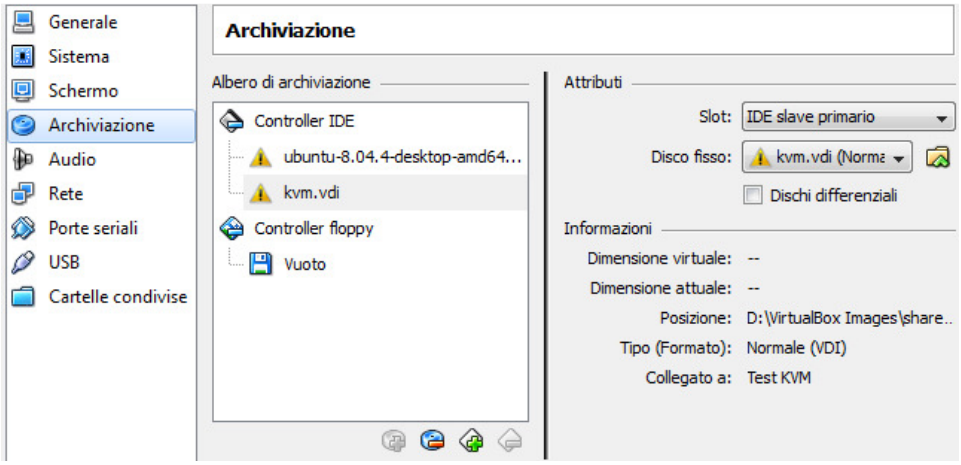


Figura 3.1 – Configurazione dei dischi di VirtualBox

V. Fare chroot all'interno della device e modificare i file di configurazione di rete in modo da avere accesso a internet.

```
mount /dev/sdb1 /mnt

mount --bind /dev /mnt/dev

mount --bind /proc /mnt/proc
mount --bind /sys /mnt/sys

chroot /mnt
```

VI. Scaricare, installare e configurare il kernel e grub.

```
apt-get install kernel-generic

apt-get install grub-pc

update-grub

grub-install /dev/sda

grub-install --recheck /dev/sda
```

VII. Modificare l'avvio del sistema operativo per permettere la corretta visualizzazione della console all'avvio.

```
Modificare il file /etc/event.d/tty1 sostituendo
xvc0 con tty1.
```

VIII. Infine, sempre tramite VirtualBox, riconvertire l'immagine vdi nuovamente in img, e ripristinare le vecchie configurazioni di rete.

```
VBoxManage clonehd --format RAW kvm.vdi kvm.img
```

In questo modo otterremo un'immagine del tutto simile a quella di Xen e perfettamente compatibile con KVM. Questo procedimento è particolarmente utile a tutti coloro che vogliono passare da un sistema virtualizzato ad un altro senza essere costretti a reinstallare e configurare nuovamente il sistema operativo e gli applicativi, modificare opzioni e preferenze e senza perdere i propri dati e contenuti.

Capitolo 4

Modifiche apportate al tool

Come già accennato l'obiettivo del presente progetto è studiare le performance del server, sia a una che a due VM, modificando le priorità di schedulazione dei relativi processi. Cercare una configurazione ottimale della macchina e confrontare i tempi di risposta così ottenuti con quelli relativi a Xen forniti da lavori di tesi di anni precedenti.

Visto che l'hypervisor Xen, diversamente da KVM, fornisce diverse funzionalità utili per analizzarne le performance, è stato necessario implementare tali funzioni separatamente.

Per poter modificare dinamicamente la priorità dei processi delle virtual machine (VM) di KVM è necessario usare la funzione renice fornita dal kernel linux; sono state quindi fatte sostanziali modifiche all'ambiente di test, sul codice Java, sugli script e sui file che si occupano del lancio e dell'esecuzione dei test.

Nel paragrafo 4.1 viene descritto l'adattamento apportato alla rete locale per far comunicare fra loro le VM durante i test. Nel paragrafo 4.2 e 4.3 sono riportate le modifiche fatte al codice java e ai file di configurazione per il corretto svolgimento dei test nel nuovo ambiente KVM. Nel paragrafo 4.4 infine sono riportate le modifiche fatte in ambiente Linux per la corretta automazione dei test e garantire la stabilità di questi.

4.1 Configurazione di rete

Per simulare l'ambiente di test su KVM, il modo ottimale che abbiamo trovato per le configurazioni della rete è stato impostare un bridge di rete nella macchina host. In questa maniera è stato possibile preservare tutte le impostazioni della rete locale usata dalle VM per comunicare fra loro nei test.

/etc/network/interfaces:

```
# This file describes the network interfaces available on
your system
# and how to activate them. For more information, see
interfaces(5).

# The loopback network interface
auto lo
iface lo inet loopback

# The primary network interface
auto eth0
iface eth0 inet dhcp

auto br0
iface br0 inet static
         address 192.168.0.2
         broadcast 192.168.0.255
         netmask 255.255.255.0
         gateway 192.168.0.1
         bridge_ports eth1
         bridge_fd 9
         bridge_hello 2
         bridge_maxage 12
         bridge_stp off
```

4.2 Modifiche al tool java

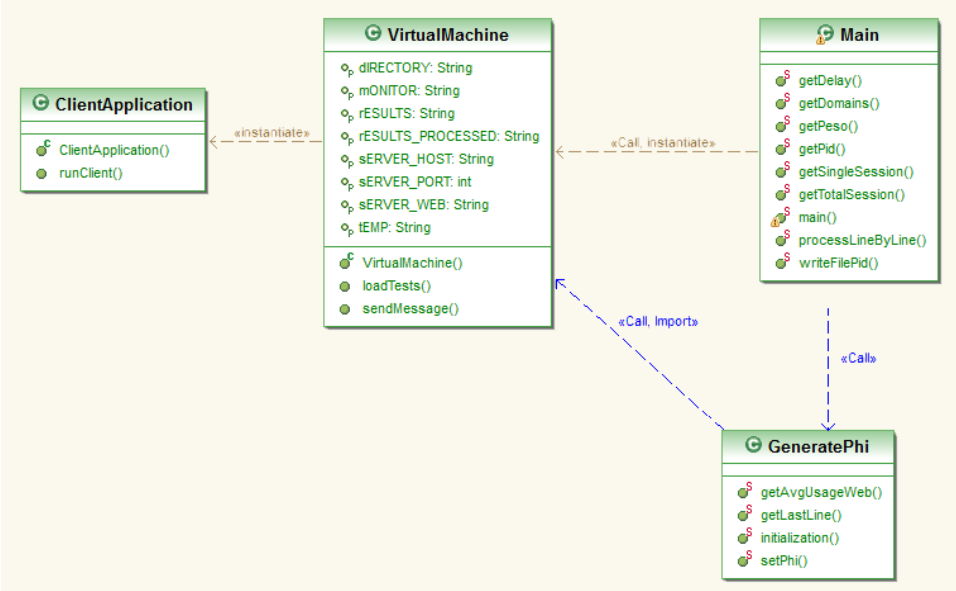


Figura 4.1 - Diagramma UML della componente client del tool

Il codice Java è stato arricchito coi metodi getPid (codice 4.1) e writeFilePid, quest'ultimo tramite uno script sh, si occupa di identificare i PID delle VM e inserirli in un file. Questo file viene poi letto dal metodo getPid, che provvederà a copiare i PID in un array che verrà poi utilizzato nel main per cambiare il nice value delle singole VM.

```
public static void getPid(int[] pid, String file, int
nVM) throws FileNotFoundException{


      Scanner scanner = new Scanner(new File(file));
      try {
            int nLine = 0;
            while (scanner.hasNextLine()&&nLine <nVM*3) {
                  pid[nLine] = scanner.nextInt();
                  System.out.println("PID: "+pid[nLine]);
                  nLine++;
            }
      } finally {scanner.close();}


}
```

Codice 4.1 – metodo getPid

Sono stati modificati, in modo da renderli compatibili con la nuova piattaforma di test, la classe WeightParser (codice 4.2) e il metodo getPeso (codice 4.3) che si occupano rispettivamente di creare e utilizzare, il file weight.cfg, che contiene i pesi da assegnare alle VM in ogni istante del test.

```
public class WeightParser {
public static void main(String[] args) throws IOException
{
int i=0;
int numLine=Integer.parseInt(args[4]);
File file = new File(path);
FileWriter fw = new FileWriter(file);
while(i<numLine){
      try  {fw.write(getLine(i,  args[0])+"  "+getLine(i,
args[1])+" "
      +getLine(i, args[2]))+"\r\n");}
      catch(IOException e) {e.printStackTrace();}
      i++;
      }
fw.flush();
fw.close();
}
```

Codice 4.2 – classe WeightParser

```
public static int[] getPeso(int line, String args, int[]
peso, int nVM)throws IOException {
Scanner scanner1 = new Scanner(new File(args));
try {
      int nLine = 0;
      while (true) {
            if (line == nLine) {
                  String aLine = scanner1.nextLine();
                  Scanner scanner2 = new Scanner(aLine);
                  int i = 0;
                  while (scanner2.hasNext()&&i < nVM*3){
                     peso[i]=(scanner2.nextInt()*(-2/10);
                     i++;
```

```
                    }
                    scanner2.close();
                    break;
                } else scanner1.nextLine();
                nLine++;
            }
} finally {scanner1.close();}
return peso;
}
```

Codice 4.3 – metodo getPeso

4.3 Modifica dei file di configurazione del tool

Grazie all'utilizzo del bridge di rete creato nell'host non è stato necessario modificare le configurazioni di rete delle varie VM. Sono state quindi utilizzate le stesse impostazioni usate negli ambienti di test Xen. L'unica modifica necessaria è stata apportata al file server.cfg delle VM client di SpecWeb, modificando l'IP dell'host che prima era l'IP esterno della macchina, con l'IP locale utilizzato dal bridge di rete.

server.cfg:

```
SERVER_PORT=9900
SAVE_DIR=/root/specWeb2005/Harness
OUTPUT_DIR=/root/specWeb2005/Harness/results
CLIENT_PORT=9910
CLIENT_HOST=192.168.0.2
SPECWEB_XMS = 512
SPECWEB_XMX = 512
```

4.4 Modifiche agli script Linux

Le modifiche apportate allo script (start.sh) per avviare il test sono state relative all'utilizzo del monitor delle VM a livello host e non più con l'utilizzo di XenMonitor. Sono cambiati inoltre alcuni file di configurazione per l'avvio del WebServer e inoltre in tutte le VM è stato aggiunto:

```
ulimit -n 1000000 -u 1000000
```

nel file /root/.bashrc che viene eseguito all'avvio del sistema operativo. Questa modifica è stata necessaria per aumentare il numero di processi utilizzabili dall'utente (-u) e il numero di file aperti (-n). Configurazione necessaria per ovviare ai problemi di saturazione nelle VM.

Ecco un esempio del file start.sh per un test Ecom a singola VM:

```
PATH_VM="/home/trevisal/Ecom"


export iterazioni=4


cd $PATH_VM


#Svuoto le cartelle delle vm dai test vecchi
cd vm1
rm *
cd ../


#Avvio il server sulle vm
echo "Avvio Apache sulle vm"
./sshlogin.exp amdpoli08 192.168.0.11 sh start.sh
./sshlogin.exp amdpoli08 192.168.0.13 sh start.sh
./sshlogin.exp amdpoli08 192.168.0.14 sh start.sh



echo "Avvio specwebclient sulle 4 porte"
./sshlogin.exp amdpoli08 192.168.0.10 sh
/root/specWeb2005/Harness/specwebclient.sh &
./sshlogin.exp amdpoli08 192.168.0.20 sh
/root/specWeb2005/Harness/specwebclient.sh &
./sshlogin.exp amdpoli08 192.168.0.120 sh
/root/specWeb2005/Harness/specwebclient.sh &


echo "Avvio i server sulle vm"
./sshlogin.exp amdpoli08 192.168.0.10 sh
/root/specWeb2005/Harness/server_back_restart.sh &
./sshlogin.exp amdpoli08 192.168.0.20 sh
/root/specWeb2005/Harness/server_back_restart.sh &
./sshlogin.exp amdpoli08 192.168.0.120 sh
/root/specWeb2005/Harness/server_back_restart.sh &
```

```
echo "Aspetto che tutto sia operativo, 60 secondi"
sleep 60

#Genero i file Test_n.config
echo "Sto generando i file Test_n.config"
sh spec_parser/generate.sh $iterazioni

#Genero il file weight.cfg
echo "Genero il file weight.cfg a partire dai phi.txt
della cartella phi"
java -jar weight.jar phi/phi1.txt phi/phi2.txt
phi/phi3.txt phi/phi4.txt $iterazioni
echo "Ho generato il file weight.cfg ora svuoto le
cartelle dei results e avvio il test"

#Cancello i file results e monitor della vm1
cd vm1/results
rm *.html *.txt *.raw
cd ../results_processed
rm *.txt
cd ../monitor
rm *.*
cd ../monitor_interno
rm *.*
cd ../monitor_bes
rm *.*

echo "Vecchi risultati cancellati"
#Cancello i file monitor di Dom0
cd ../../
cd monitorDom0
rm *.*
rm *
cd ../
echo "File del dom0 eliminati"

./sshlogin.exp amdpoli08 192.168.0.11 rm /root/monitor/*
```

```
./sshlogin.exp amdpoli08 192.168.0.13 rm /root/monitor/*
echo "File monitor su web e bes cancellati, ora parte il
test"


#Lancio il monitor dell'host
sh java -jar testmonitor.jar
echo "Monitor avviato"


#lancio il test che avrà tutte le cartelle di
destinazione pulite
java -jar client.jar client.cfg weight.cfg domain.cfg
delay.cfg
echo "Test concluso"


echo "Inizio a trasferire i risultati"
./scplogin.exp amdpoli08 192.168.0.10
/root/specWeb2005/Harness/results/*.*
$PATH_VM"/vm1/results"
echo "File dal client scaricati"


./scplogin.exp amdpoli08 192.168.0.11 /root/monitor/*.*
$PATH_VM"/vm1/results"
echo "File dal web scaricati"


./scplogin.exp amdpoli08 192.168.0.13 /root/monitor/*.*
$PATH_VM"/vm1/results"
echo "File dal bes scaricati"


#Parser
echo "Avvio del parser"
cd /home/trevisal/Ecom/spec_parser
java -jar spec_out_directory.jar $PATH_VM"/vm1/results"
echo "Parser finito"
cd /$PATH_VM"/vm1/results"
echo "sposto outputA e B files"
mv output* $PATH_VM"/vm1/results_processed"
echo "sposto Results_summary.txt"
mv Results_summary.txt $PATH_VM"/vm1/results_processed"
cd ../../
```

Capitolo 5

Ambiente di Test

In questo capitolo verrà descritto l'ambiente a supporto delle analisi di prestazione e successivamente verranno illustrate le fasi di setup dei test e della scelta dei dati di input. Nel paragrafo 5.1 viene presentata la macchina fisica con la quale vengono svolte le analisi e la relativa configurazione. La scelta dei dati di input viene discussa nel paragrafo 5.2.

5.1 Descrizione dell'Ambiente di Test

L'ambiente di test è costituito da un server su cui sono installate le Virtual Machine che sono state utilizzate. Tutte le Virtual Machine si basano su un'installazione di KVM a sua volta installata su una distribuzione Ubuntu Server 10.04 con kernel 2.6.32-21 e sono supportate da un processore Intel Nehalem quad-core con due socket e 24 GB di RAM. Le macchine virtuali che ospitano il web server, il client e il back-end simulator operano su sistema operativo Linux Ubuntu Server 8.04.

La configurazione della RAM che si è dimostrata ottimale, sia per i test banking che per quelli Ecom, è stata la seguente: Client: 4GB, Web server: 2GB, Back-end simulator: 1,5GB.

Per i test che sono stati effettuati ci siamo limitati ad utilizzare solo due tipi di workload che SPECweb fornisce, eCommerce e Banking.

Abbiamo effettuato test sia ad una VM che a due VM, diversificando le priorità assegnate alle varie VM.

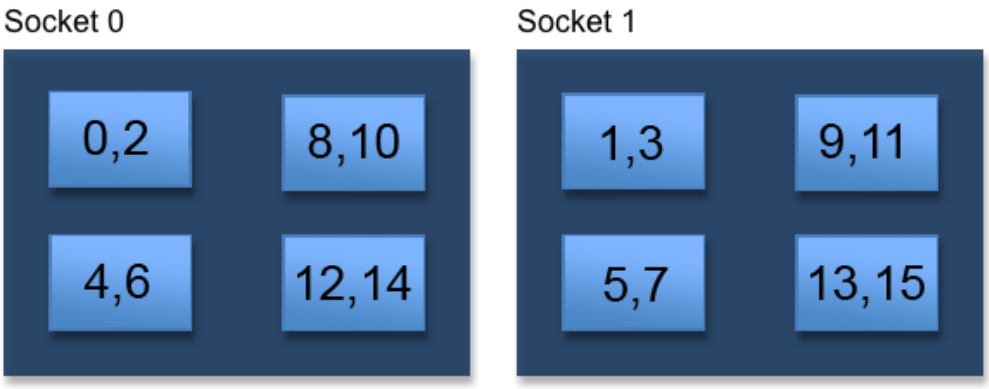


Figura 5.1.Struttura della CPU

La struttura di calcolo della macchina è mostrata in figura 5.1, la CPU è costituita da due socket, ognuno dei quali è suddiviso in 4 core fisici che sfruttano la tecnologia Hyper-Threading sviluppata da Intel.

Per quanto riguarda i test a singola VM sono stati effettuati una serie di test assegnando un core fisico su ogni macchina virtuale usando per esempio il comando

```
taskset -c -p 0,2 31415
```

e variando il carico di sessioni simultanee tra 200, 400, 600 e 800, lasciando in questo modo la gestione dei thread allo scheduler dell'HTTP Server.

Per quanto riguarda i test a due VM sono stati effettuati una serie di test facendo condividere un core fisico alle virtual machine dello stesso tipo. Modificando la priorità di esecuzione delle VM web, tramite il comando renice, si è studiata la variazione di performance al fine di trovare la configurazione ottimale della macchina.

5.2 Scelta dei dati di input

Per i test effettuati sono state fatte delle scelte circa i parametri del singolo test. Tranne i valori di SIMULTANEOUS SESSIONS, che come illustrato nel paragrafo precedente variano di test in test, gli altri rimangono costanti.

Questa scelta è stata ovviamente fatta per avere una omogeneità dei risultati tale da poter confrontare i diversi test tra di loro.

I parametri sono stati impostati in questo modo:

```
WARM UP SECONDS = 1200 sec
```

RAMPUP SECONDS = 0

RUN SECONDS = 1800

RAMPDOWN SECONDS = 0

ITERATIONS = 1

I valori di RAMPUP SECONDS e RAMPDOWN SECONDS sono stati settati a 0 per verificare il comportamento del web server quando viene sottoposto ad un salto immediato dalla situazione di riposo a quella di carico massimo.

A causa di questo salto il web server deve rispondere a dei carichi notevoli.

Per questo motivo, nei test ad elevato carico (in particolare in quelli dove le SIMULTANEOUS SESSIONS superano i 400 utenti), è stata verificata l' insufficienza di un solo client come generatore di carico.

Ciò ha avuto come conseguenza l'interruzione dei test, in quanto le singole sessioni superavano il limite massimo di THINK TIME prima di effettuare una nuova richiesta al web server, molto probabilmente a causa del fatto che il generatore di carico non era in grado di tornare a servire la singola sessione prima dell'esaurimento del tempo tra una richiesta e l'altra.

Per ovviare a tale problema, effettuando dei test di prova con il massimo carico, è stato verificato sperimentalmente che con 3 clients operanti come generatori di carico l'errore non si è più presentato.

A dispetto delle disposizioni di SPECweb al fine di generare dei test validi, i cosiddetti test compliant, il valore di ITERATIONS è stato impostato sempre ad 1, mentre il valore di default è 3.

I risultati ottenuti con questa variante non sono, come detto, dei risultati validi ai fini dell'applicazione di benchmark, ma le statistiche che vengono raccolte e i risultati che vengono generati non sono in alcun modo falsati, in quanto ogni tipo di test è stato ripetuto comunque più volte per avere un parco campioni più ampio.

Capitolo 6

Analisi dei dati sperimentali

In questo capitolo sono riportati i dati ottenuti dai test sia a singola che a doppia virtual machine, utilizzando workload eCommerce e Banking.

Con i test effettuati i confronti che si possono fare sono principalmente di tre tipi: analizzare il comportamento delle singole VM al variare del carico di utenti a cui sono state sottoposte. Sempre per le singole VM confrontare i dati sperimentali raccolti su KVM con quelli di Xen forniti da lavori di tesi precedenti valutando le differenze di performance, i vantaggi e gli svantaggi dei due hypervisor. Per quanto riguarda i test a due VM è stato inoltre analizzato l'impatto dei parametri di sheduling sulle performance del server e cercarne la configurazione ottimale.

Nel paragrafo 6.1 sono esposti i risultati ottenuti effettuando test a 1VM a vari carichi di lavoro. Confronti con le prestazioni dei medesimi test effettuati su Xen vengono fatti nel paragrafo 6.2. Infine nell'ultimo paragrafo, il 6.3, vengono studiati gli effetti dello scheduling di Linux sui test a 2VM.

6.1 Analisi del comportamento delle singole VM con KVM

Per quanto riguarda i test ecommerce sono stati fatti test con 200, 400, 600 utenti.

Nelle figure 6.1, 6.2, 6.3 viene mostrato l'utilizzo della risorsa cpu al variare del carico di utenza erogato.

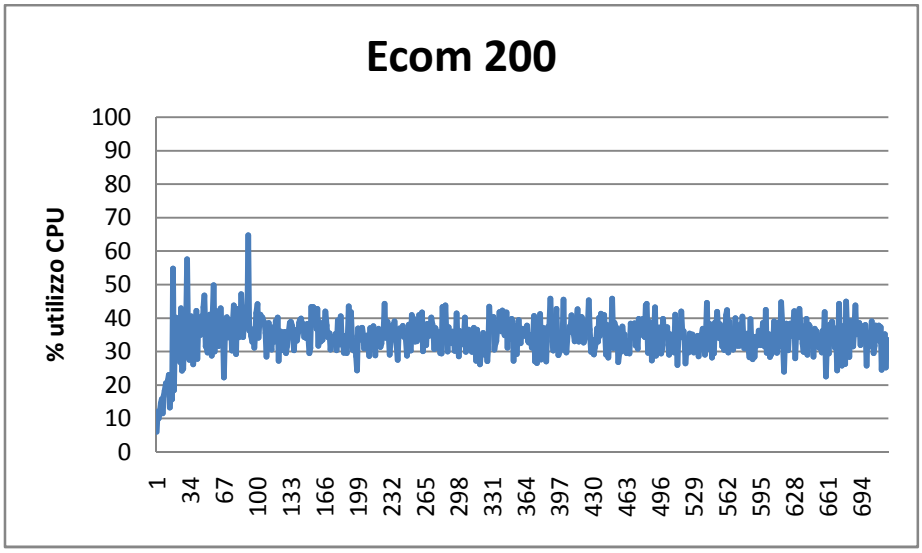


Figura 6.1 - Utilizzo CPU della VM ecom con 200 utenti

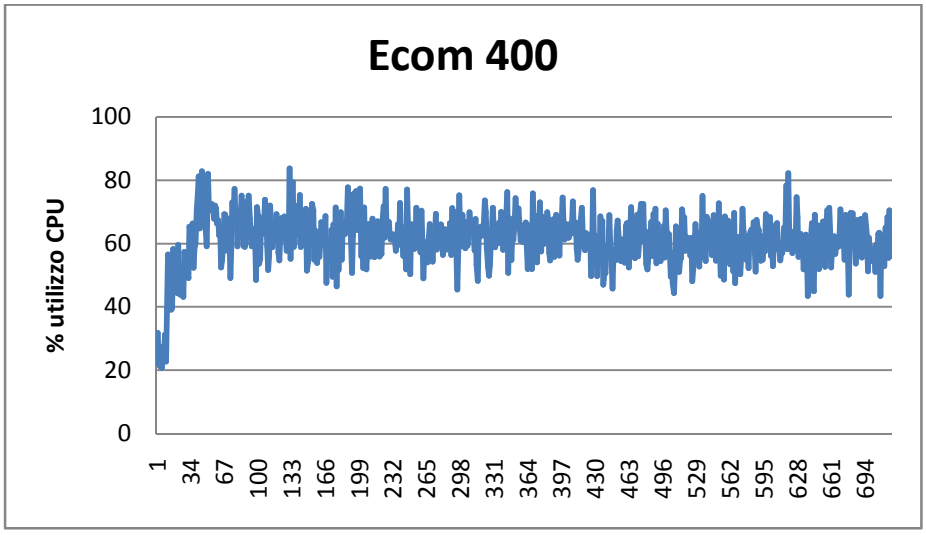


Figura 6.2 - Utilizzo CPU della VM ecom con 400 utenti

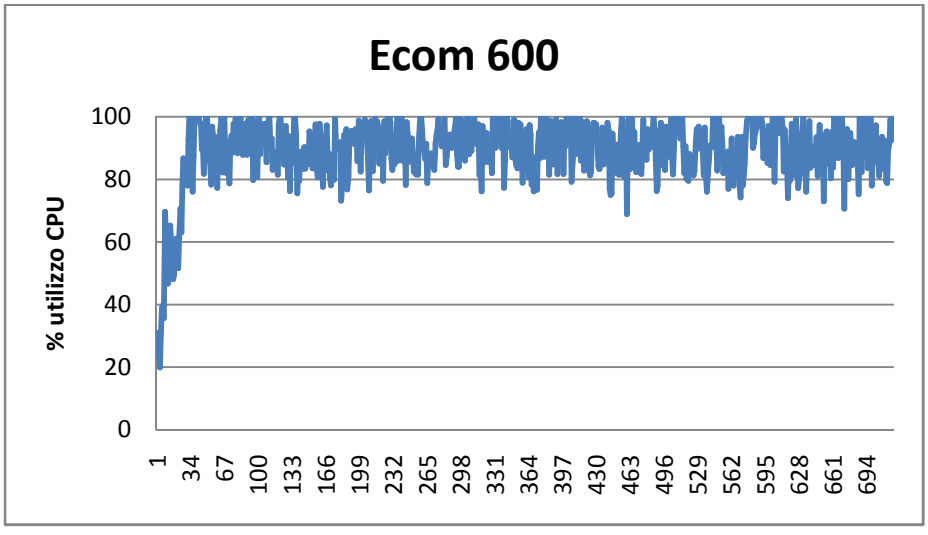


Figura 6.3 - Utilizzo CPU della VM ecom con 600 utenti

Come si vede dal grafico in figura 6.4 l'utilizzo medio di cpu al variare del numero di utenti è con buona approssimazione lineare.

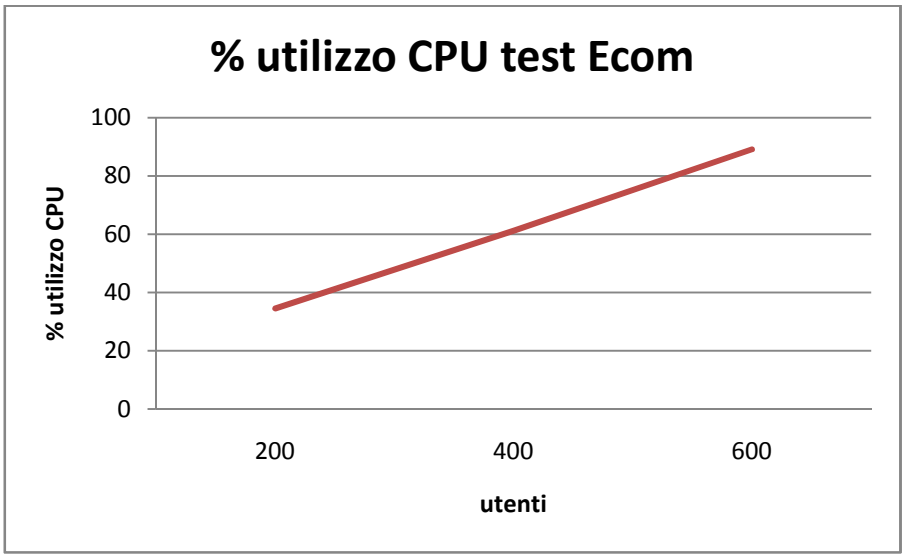


Figura 6.4 - Utilizzo medio CPU in funzione del numero di utenti nella VM ecom

In quanto ai test banking sono stati fatti test con 200, 400, 600, 800 utenti.

Nelle figure dalla 6.5 alla 6.8 viene mostrato l'utilizzo della risorsa cpu al variare del carico di utenza erogato.

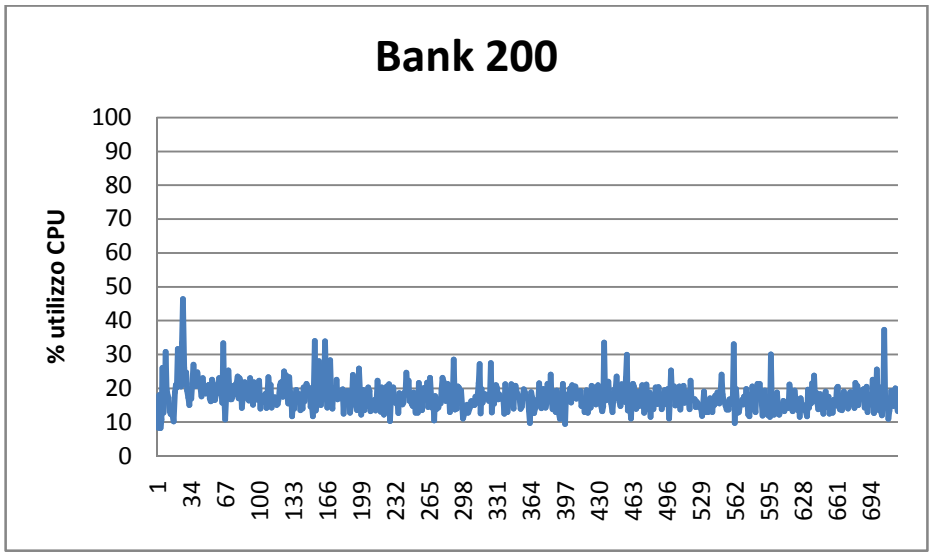


Figura 6.5 - Utilizzo CPU della VM banking con 200 utenti

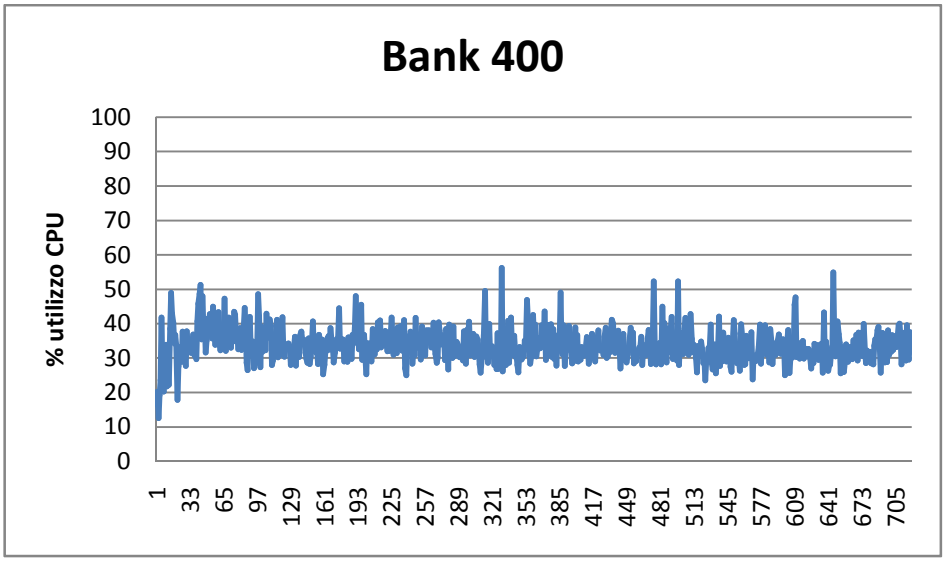


Figura 6.6 - Utilizzo CPU della VM banking con 400 utenti

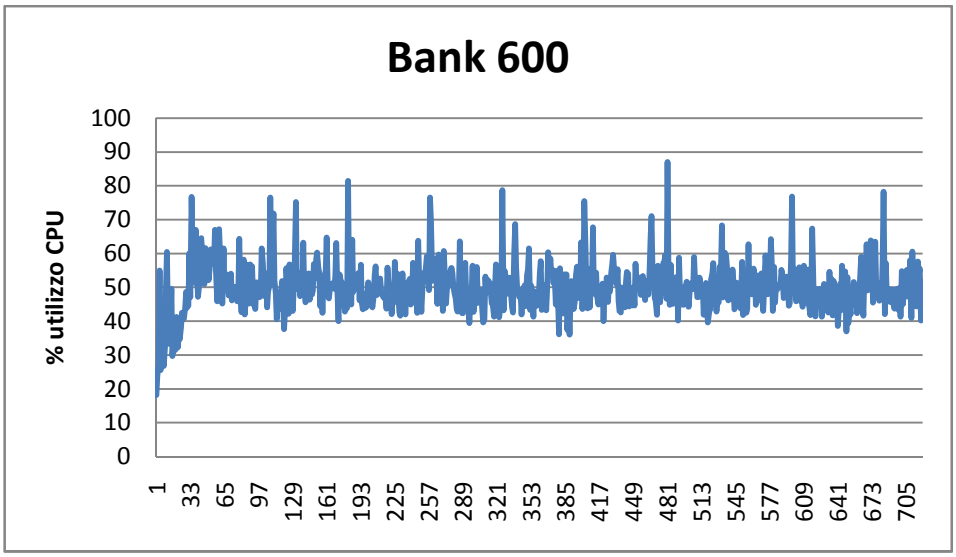


Figura 6.7 - Utilizzo CPU della VM banking con 600 utenti

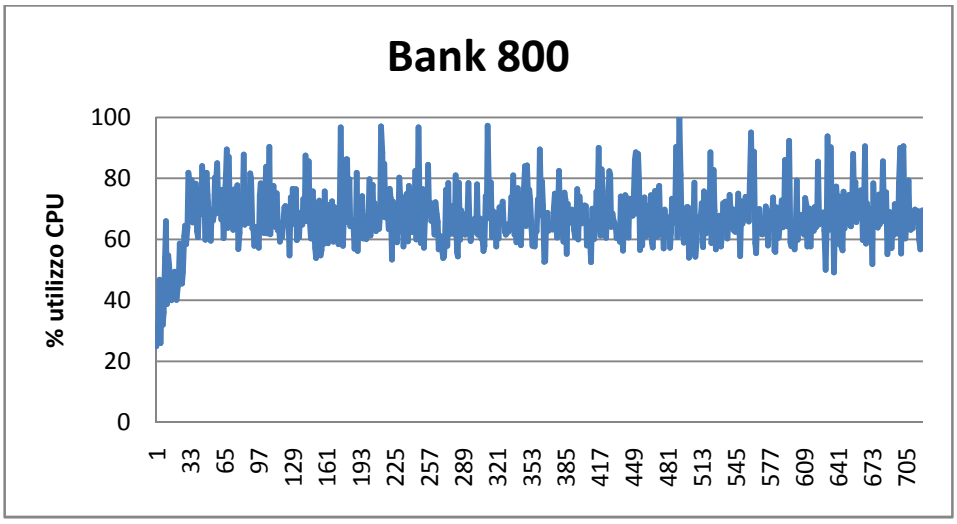


Figura 6.8 - Utilizzo CPU della VM banking con 800 utenti

Anche per i test banking, come si vede dal grafico in figura 6.9 l'utilizzo medio di cpu al variare del numero di utenti è, con buona approssimazione, lineare.

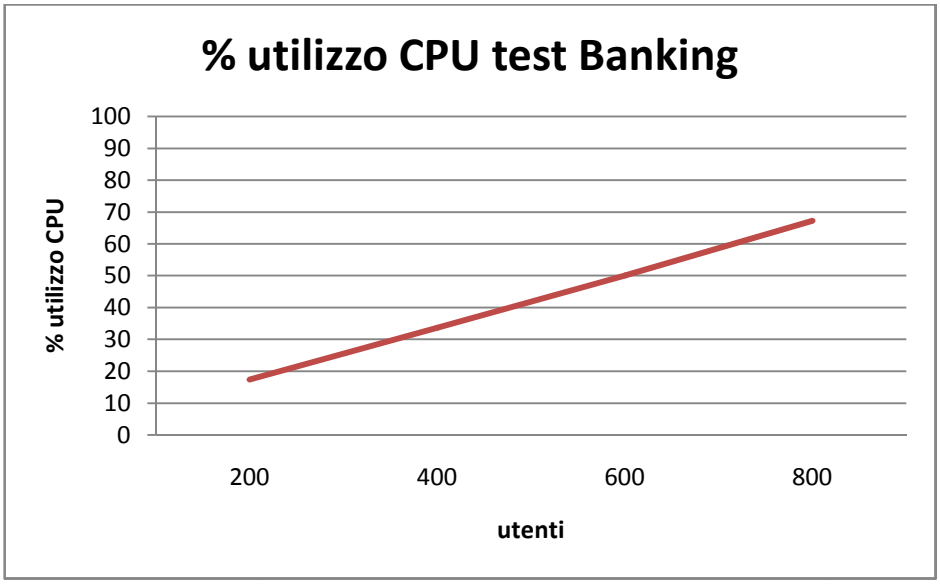


Figura 6.9 - Utilizzo CPU in funzione del numero di utenti nella VM banking

Le due tipologie di test sono messe a confronto nei grafici in figura 6.10 e 6.11 dove sono rappresentati rispettivamente l'utilizzo medio di cpu e il tempo medio di risposta in funzione del numero di utenti per entrambe le tipologie di lavoro.

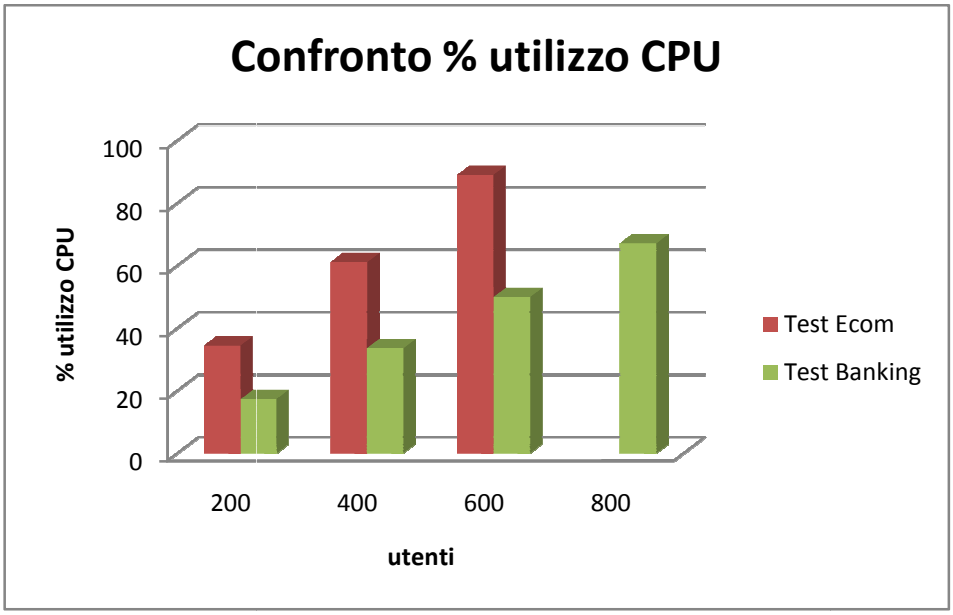


Figura 6.10 - Confronto utilizzo medio CPU tra test ecom e banking

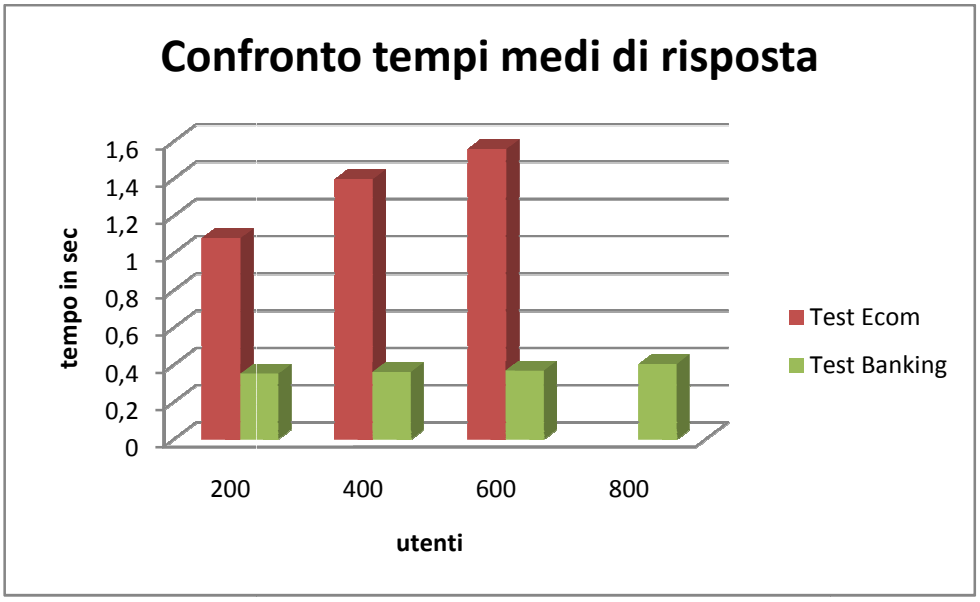


Figura 6.11 - Confronto tempi medi di risposta tra test ecom e banking

Dai grafici appare evidente che, sia per il tempo di risposta, che per l'utilizzo della cpu i workload banking risulta decisamente più performante di quello ecom dove infatti non è stato possibile effettuare un test con 800 utenti per problemi di saturazione della CPU nella VM web.

Questa differenza di prestazioni è dovuta al fatto che le VM di tipo ecom effettuano molti più accessi al disco rispetto a quelle banking. Sono proprio questo tipo di operazioni di I/O che implicano l'intervento dell' hypervisor, causando ulteriore overhead computazionale.

6.2 Confronto tra Xen e KVM

Dai nostri test si riscontra un grave insuccesso di KVM. Le prestazioni, come rappresentato nelle figure 6.12 e 6.13, raggiungono all'incirca il 35% di quelle ottenibili da sistemi basati su Xen. I tempi di risposta nei test sono pressoché identici, ma l'utilizzo della CPU su KVM è quasi il triplo di quello su Xen per svolgere lo stesso lavoro, con lo stesso carico di utenza.

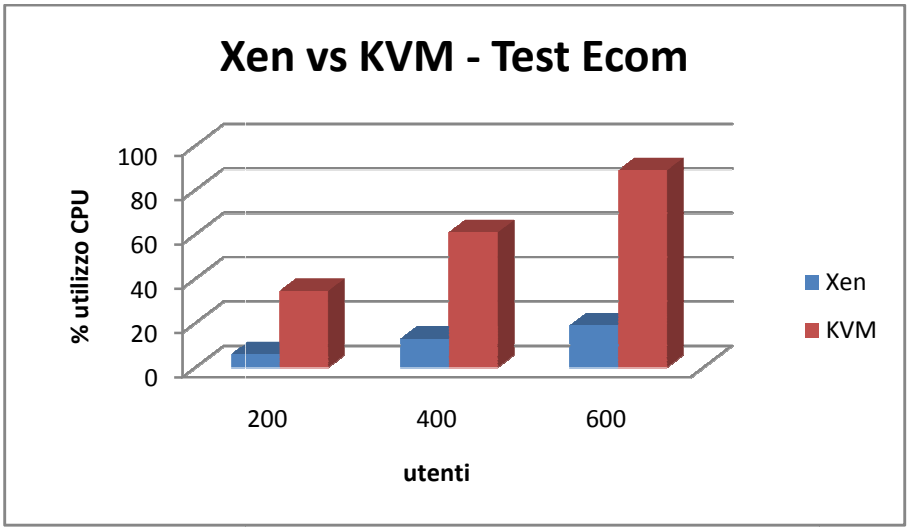


Figura 6.12 - Confronto utilizzo medio CPU tra Xen e KVM nei test ecom

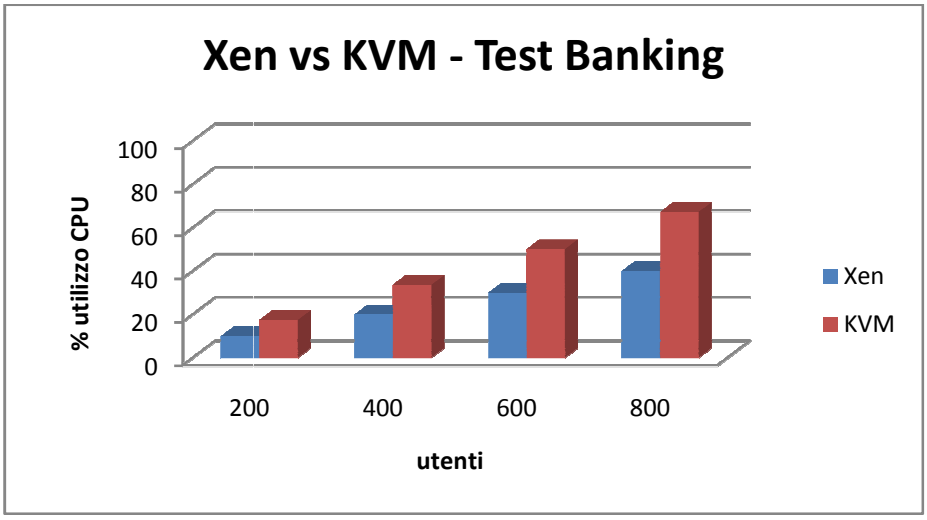


Figura 6.13 - Confronto utilizzo medio CPU tra Xen e KVM nei test banking

KVM è una tecnologia nuova e in forte sviluppo. Come dice il nome stesso (Kernel-based Virtual Machine) è legata fortemente al kernel, e quindi le sue prestazioni sono basate sull' ottimizzazione dei moduli che vengono utilizzati dall'hypervisor. I kernel attualmente utilizzati non sono ancora stati ottimizzati al punto da eguagliare le prestazioni di quelli compilati ad hoc per Xen.

La comunità Linux, con il supporto della ricerca Intel è molto impegnata nello sviluppo e nello studio di librerie opensource per la virtualizzazione.

Test effettuati da Intel (figura 6.14) rilevano dati equiparabili a quelli ottenuti nel presente lavoro di tesi. Studi hanno portato alla conclusione che la perdita di prestazione può essere dovuta a un considerevole overhead causato dalla gestione non ottimale dell'accesso su disco.

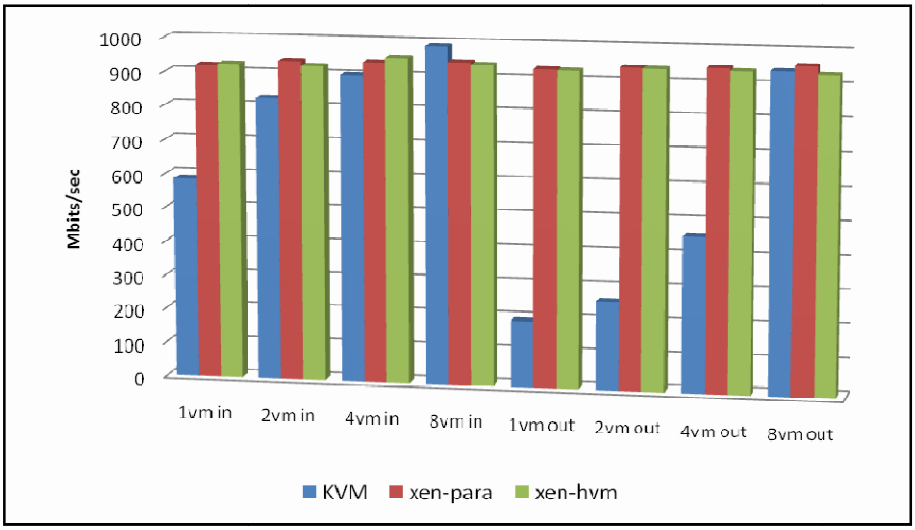


Figura 6.14 - Throughput I/O in funzione del numero di VM attive

Nei test svolti su Xen si è monitorato, a parità di utenti, un minore utilizzo della CPU nei test ecommerce rispetto a quelli banking. Questa differenza è dovuta al fatto che Xen offre il 90% delle prestazioni del sistema nativo, nei quali un test ecommerce risulta meno impegnativo per la CPU. Su KVM, invece, come si nota dalla figura 6.15, a causa di una non ottimale gestione dell'I\O si ha un forte decadimento di prestazione nei test ecommerce, che generano una notevole richiesta di pagine su disco rispetto ai test banking.

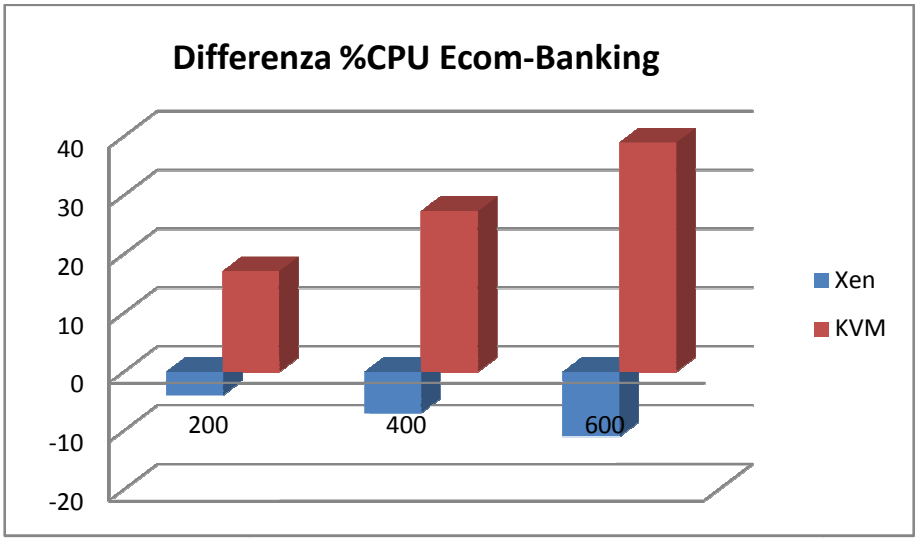


Figura 6.15 - Differenza utilizzo medio CPU tra test ecom e banking per entrambi gli hypervisor

Ad ogni modo è importante sottolineare che, come per Xen, anche utilizzando KVM si sono registrati percentuali di fallimento (failure rate) pari a 0, alla conferma del fatto che il sistema è configurato correttamente.

6.3 Impatto dei parametri di scheduling sui test 2VM

Per quanto riguarda i test a 2VM si sono riscontrati problemi di saturazione già con test a 400 utenti per ogni VM. Siamo riusciti ad effettuare sessioni con un carico di lavoro di 200 utenti.

Abbiamo studiato la variazione di prestazioni nei vari carichi di lavoro in funzione della priorità assegnata ai WebServer. In particolare sono stati eseguiti test con le seguenti configurazione dei pesi:

- 30% ecommerce, 70% banking (figura 6.16)
- 50% ecommerce, 50% banking (figura 6.17)
- 70% ecommerce, 30% banking (figura 6.18)

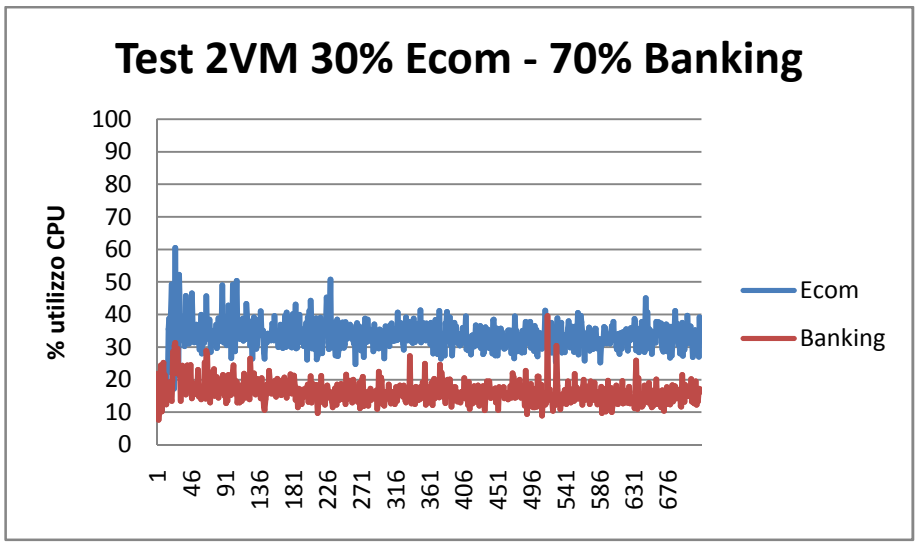


Figura 6.16 – Test 2VM 30% ecom, 70% banking

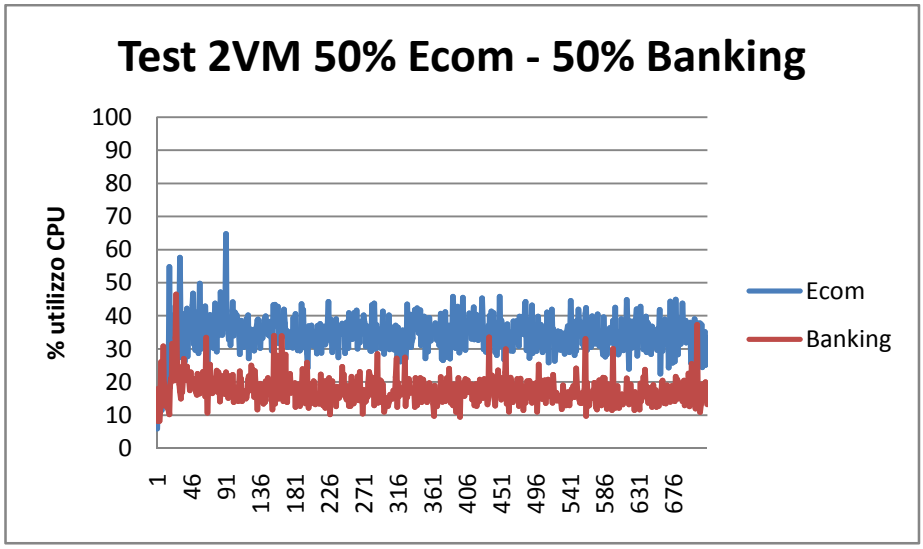


Figura 6.17 – Test 2VM 50% ecom, 50% banking

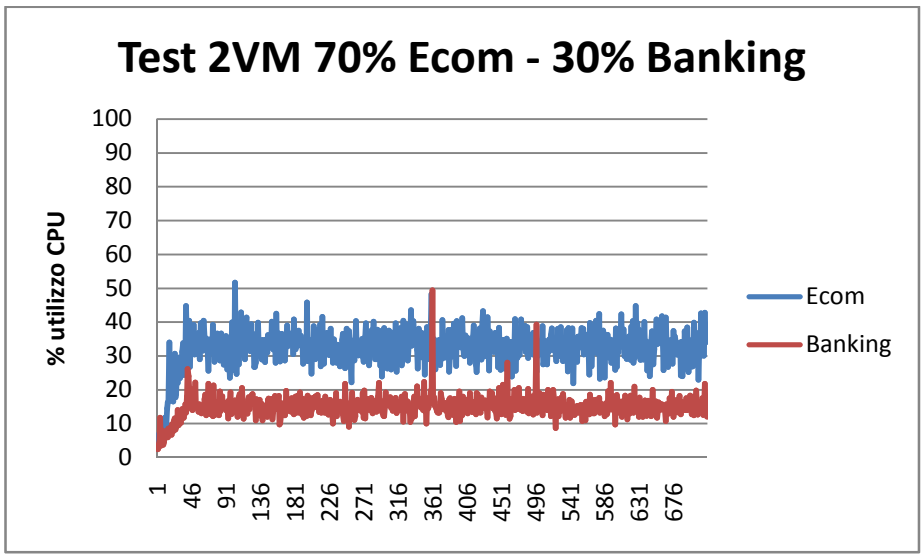


Figura 6.18 – Test 2VM 70% ecom, 30% banking

Distribuzione CPU	% utilizzo medio CPU Ecom	% utilizzo medio CPU Bank
Ecom 70% - Bank 30%	33,3%	16,3%
Ecom 50% - Bank 50%	34,4%	17.3%
Ecom 30% - Bank 70%	32,3%	15,1%

Tabella 6.1 – Confronto utilizzo CPU in funzione della priorità

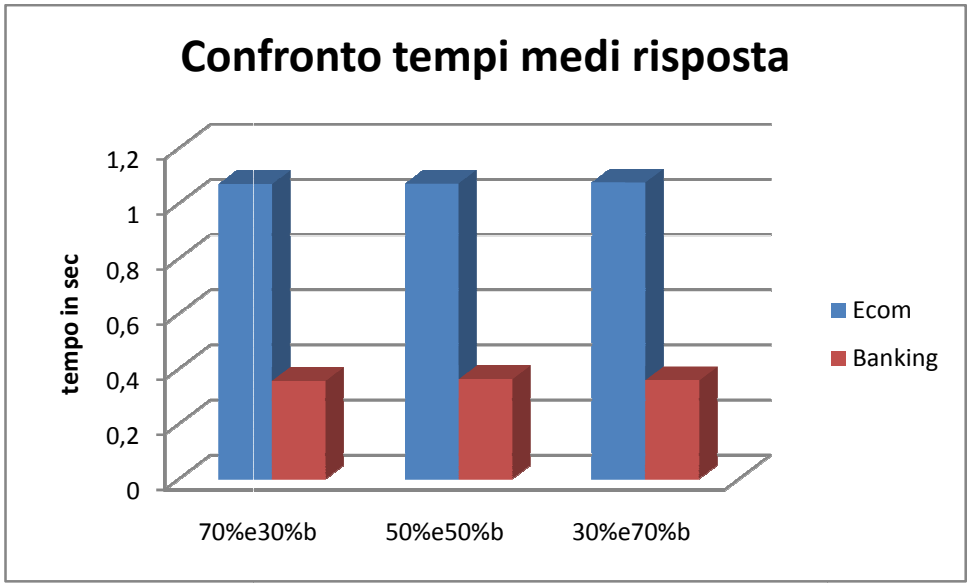


Figura 6.19 – Confronto tempi di risposta in funzione della priorità

Distribuzione CPU	Tempo di risposta Ecom	Tempo di risposta Bank
Ecom 70% - Bank 30%	1,070 s	0,356 s
Ecom 50% - Bank 50%	1,072 s	0,362 s
Ecom 30% - Bank 70%	1,076 s	0,358 s

Tabella 6.2 – Confronto tempi di risposta in funzione della priorità

Come si nota dai grafici dalla figura 6.16 alle 6.19 e dalle tabelle 6.1 e 6.2 al variare della priorità assegnata alle singole VM non cambiano ne gli utilizzi del processore ne i tempi di risposta. E' un dato che ci si aspettava, poiché anche nei test Xen con un basso carico di lavoro non si notano sostanziali differenze nei risultati.

Conclusioni

Questo lavoro di tesi ha avuto come obiettivo primario lo sviluppo di un tool per l'automazione del lancio di test di performance per web server su macchine virtuali, modificando un tool sviluppato per Xen, adattandolo alla nuova piattaforma KVM.
La virtualizzazione rappresenta oggi una delle principali tecniche per la riduzione del numero di macchine all'interno dei data center aziendali, senza alterare la qualità di servizio offerta e ottimizzando lo sfruttamento delle risorse infrastrutturali.
Tra le varie tecnologie di virtualizzazione presenti oggi sul mercato, è stato scelto il Virtual Machine Monitor KVM (Kernel-based Virtual Manager), un software opensource, sviluppato inizialmente da Qumranet, una compagnia emergente acquistata nel 2008 da Red Hat. KVM è un software che si integra nel kernel Linux e permette di mantenere un controllo efficiente dello sfruttamento delle risorse di calcolo da parte delle singole Virtual Machine.
In una prima fase è stato adattato il tool per l'automazione del lancio di test SPECweb. Il tool si basa su due componenti secondo il paradigma client/server: il componente lato client esporta un insieme di comandi per lo start e il retrieve dei risultati dei test, mentre il componente lato server opera con l'applicazione di benchmarking SPECweb che sottopone il server a carichi variabili nel tempo modellando situazioni di traffico reali, simulando 3 tipologie di workload (eCommerce, Banking e Support). Tramite l'applicazione di benchmarking sono stati eseguiti test con diverso numero di sessioni simultanee e di Virtual Machine al fine di confrontare inizialmente le performance dell'hypervisor Xen con quello KVM e infine di analizzare il comportamento dello scheduler di Linux. L'esecuzione dei test è stata automatizzata in modo da ottimizzare i tempi di esecuzione di lunghe sessioni.
I dati ottenuti dimostrano che la nuova piattaforma KVM è ancora lontana dall'offrire le stesse prestazioni del collaudato Xen. I problemi di overhead creati dall'hypervisor nell'utilizzo del disco limitano fortemente le prestazioni di KVM. La community Linux in collaborazione con Intel sta lavorando intensamente allo sviluppo di questo sistema di virtualizzazione poiché, per come è stato strutturato il progetto di base, con un buon lavoro di sviluppo KVM può diventare l'hypervisor della nuova generazione sfruttando al meglio le potenzialità offerte dalle tecnologie, Intel-VT e AMD-V, integrate in tutti i nuovi processori.
I kernel Linux dalla versione 2.6.20 hanno inoltre integrati i moduli KVM per offrire a tutte le distribuzioni e ai nuovi utenti un facile utilizzo di questa piattaforma, al contrario di Xen che necessita di sostanziali modifiche al kernel, richiedendo una compilazione ad hoc dello stesso.

Bibliografia

- F. Amarilli, D. Ardagna, M. G. Fugini, and R. Tedesco. Impianti informatici: Tecnologie e Applicazioni, 2007.
- D. Ardagna, M. Trubian, and L. Zhang. SLA based resource allocation policies in autonomic environments. Journal of Parallel and Distributed Computing, 2007.
- Paul Barham, Boris Dragovic, Keir Fraser, Steven Hand, Tim Harris, Alex Ho, Rolf Neugebauer, Ian Pratt, and Andrew Wareld. Xen and the art of virtualization. In SOSP '03: Proceedings of the nineteenth ACM symposium on Operating systems principles, New York, NY, USA, 2003. ACM.
- Ludmila Cherkasova, Diwaker Gupta, and Amin Vahdat. Comparison of the three cpu schedulers in xen. SIGMETRICS Perform. Eval., 2007.
- S. Govindan, A. R. Nath, A. Das, B. Urgaonkar, and A. Sivasubramaniam. Xen and co.: communication-aware cpu scheduling for consolidated xen-based hosting platforms. ACM, 2007.
- D. Gupta, L. Cherkasova, R. Gardner, and A. Vahdat. Enforcing performance isolation across virtual machines in xen. In Middleware, 2006.
- Dara Kusic, Jerey O. Kephart, James E. Hanson, Nagarajan Kandasamy, and Guofei Jiang. Power and performance management of virtualized computing environments via lookahead control. Autonomic Computing, International Conference on, 2008.
- Gerald J. Popek and Robert P. Goldberg. Formal requirements for virtualizable third generation architectures. Communications of the ACM, 1974.
- VMware. VMware: Virtualization via hypervisor, virtual machine & server consolidation. http://www.vmware.com/.
- T. Wood, P. J. Shenoy, A. Venkataramani, and M. S. Yousif. Black-box and Gray-box Strategies for Virtual Machine Migration. In USENIX, 2007.
- Xen. The Xen hypervisor, the powerful open source industry standard for virtualization. http://www.xen.org/.
- KVM. Kernel Based Virtual Machine. http://www.linux-kvm.org/.

- Ubuntu. The computer operating system based on the Debian GNU/Linux distribution. http://www.ubuntu.com/.

- J. Assfalg. Appunti di Sistemi Operativi - Università degli studi di Firenze.

- Daniele Masini. Lo scheduler di Linux - http://vandali.org/danielemasini/.

- Red Hat - KVM Documentation - 2009. http://www.redhat.com/.

- IOP Science - A quantitative comparison between Xen and KVM - http://iopscience.iop.org/.

- Intel open source software – KVM and Xen, performance comparison - http://software.intel.com/