POLITECNICO DI MILANO
Sede di Como
Corso di Laurea in Ingegneria Informatica
Dipartimento di Elettronica e Informazione

Un confronto prestazionale tra tecnologie di virtualizzazione

Relatore: Prof. Paolo Cremonesi
Correlatore: Ing. Andrea Sansottera

Tesi di Laurea di:
Claudio Chinosi
Matricola 734365

Anno Accademico 2009-2010

Alla mia famiglia

Ringraziamenti

Ci siamo.

Dunque, sono giunto anche io alla fine di questo percorso. Se mi fermo a riflettere un attimo, mi rendo conto che tale esperienza non sarebbe stata possibile senza l'aiuto di alcune persone. Desidero in primo luogo ringraziare il mio relatore, il Prof. Paolo Cremonesi, per avermi dato l'opportunità di svolgere questo lavoro e per la disponibilità mostratami in ogni momento. Oltre ad avermi permesso di accrescere la mia conoscenza, desidero ringraziarlo ulteriormente per avermi concesso un'opportunità di stage in azienda presso Moviri S.r.l., un'azienda leader nella ottimizzazione delle performance IT e nelle soluzioni per il capacity planning. Colgo perciò l'occasione di ringraziare tale azienda nell'avermi avermi aiutato ad interfacciarmi con la realtà lavorativa. In particolare, un sentito ringraziamento per l'aiuto dimostratomi è rivolto all' Ing. Giorgio Gasparini.

Un altro ringraziamento molto sentito è rivolto all' Ing. Andrea Sansottera che, con la sua pazienza, chiarezza e disponibilità, mi ha spiegato ogni cosa nel dettaglio ed è sempre stato disposto a venirmi incontro per aiutarmi a capire. Inoltre, desidero ringraziare le persone presenti nel laboratorio VP-LAB per i momenti passati insieme, da quelli più faticosi di lavoro a quelli più leggeri e di svago.

C'era chi aspettava questo momento da tanto; vedete, accompagnandomi sempre con persone più grandi, una specie di eco mi è sempre risuonato nelle orecchie:"Vedrai che arriverà il momento anche per te!!", oppure:"Maledetto balordo, guardalo! Non sa neanche dov'è, lui è sveglio ma non sa dov'è, ha la bocca ancora impastata!" ed ancora:"Dura guera che mi resisti!". Esclamazioni a cui prontamente rispondevo:"Vero! Ma fino ad allora sbattimento a go go!" oppure:"Tutta invidia la vostra!". Figuratevi che c'erano persone che volevano che sostenessi gli esami ancora prima di aver seguito le lezioni pur di svegliarmi dal torpore studentesco. Cari amici e parenti, questo è il momento...sfogatevi! Ora non saprò più cosa rispondervi, perchè tra poco saremo sulla stessa barca

e, con una punta di nostalgia, ricorderò quei momenti di beato fannullonismo che mi hanno sempre contraddistinto. Anzi, in realtà sapete che qualcosa vi risponderò:"Oberato di lavoro!", ma purtroppo temo che non sarà più in senso ironico!

Scherzi a parte, un periodo della mia vita sta per concludersi e ad una persona corre l'obbligo di fare il bilancio tra quello che ha guadagnato e quello che avrebbe potuto guadagnare. In questi anni da studente, mi sono reso conto di quanto la volontà sia tutto nella vità. Sono convinto che è una persona sia ciò che vuole essere. Riconosco che, nonostante il risultato ottenuto, del tempo è andato perso. Probabilmente perchè, in certi momenti, non sono stato abbastanza maturo per capire l'importanza di quello che stavo facendo ed una mera superficialità mi ha portato a mettere in secondo piano le cose essenziali. Essenziali per me come persona.

Il passato non si può cambiare, rimane una piccolo rimpianto, ma il futuro sta a me crearlo e credo che già aver maturato un ragionamento del tipo sopra descritto sia un buon passo avanti. Spero fortemente che i fatti dimostreranno questo cambiamento nel modo di pensare.

Infine, desidero ringraziare in modo speciale le seguenti persone:

Mamma, Papà, Matteo e Jenny : un grazie grande come il mondo alla mia famiglia, per avermi permesso di poter scegliere come essere consogliandomi, spronandomi, aiutandomi ed indicandomi la via.

Zii e cugini : grazie a tutti per la vostra vicinanza e per il vostro sostegno. Siete importanti!

Elena : grazie per il modo in cui mi fai sentire che ci sei, per le discussioni costruttive che facciamo, per i momenti indimenticabili passati insieme e che spero non finiranno mai.

Varedo Basket : grazie alla squadra di basket dove milito, per avermi concesso lo sfogo di cui ho bisogno e di cui non potrei fare a meno.

Sommario

In questa tesi si vogliono evidenziare gli aspetti relativi alle performance andando a confrontare quattro tecnologie di virtualizzazione in modo che sia possibile implementare soluzioni per il relativo miglioramento. Ciò si ottiene nel caso di specie attraverso la generazione di un workload sintetico di tipo CPU e memory intensive. Viene dapprima presentato un lavoro sul throughput massimo ottenibile, nella seconda parte si considerano invece i tempi di risposta per vedere quali politiche gli hypervisor adottano per gestire i guest in esecuzione, soprattutto quando si agisce sulla modifica degli utilizzi delle singole virtual machine. I risultati mostrano che tutte le tecnologie testate introducono un overhead di virtualizzazione che si traduce in un decremento del throughput del sistema sotto test e che, variando l'utilizzo delle singole virtual machine ma mantenendo lo share uguale, i tempi di risposta non rispecchiano quelli descritti nei modelli classici della teoria delle code.

Indice

Ringraziamenti **I**

Sommario **III**

1 Introduzione **1**

2 Virtualizzazione: stato dell'arte **3**
2.1 Definizione e principali tipologie 3
2.2 Benefici e svantaggi introdotti 6
2.2.1 Come valutare i benefici 8
2.3 Tecniche di virtualizzazione 9
2.3.1 La macchina virtuale 10
2.3.2 Virtualizzazione dell'architettura x86 11
2.4 Maggiori prodotti disponibili nello specifico 15
2.4.1 VMware 15
2.4.2 Citrix 18
2.4.3 Microsoft 19
2.4.4 KVM 20
2.5 Virtualizzazione e benchmarking 21
2.5.1 Lavori relativi a valutazione delle performance in sistemi virtualizzati 22
2.6 Benchmark orientati alla virtualizzazione 28
2.6.1 SPECvirt-sc2010 28
2.6.2 Caso VP-LAB 31

3 Descrizione del benchmark **35**
3.1 Configurazione ed ambiente di testing 35
3.2 Rilevamento del throughput massimo 36
3.3 Rilevamento del response time 36
3.4 Descrizione del workload sintetico 37

4 Throughput massimo in sistemi virtualizzati **43**
4.1 Formulazione del problema 44
4.2 Risultati 49
4.3 Conclusioni 52

5 Tempi di risposta in sistemi virtualizzati **53**
5.1 Formulazione del problema 54
5.2 Risultati 58
5.3 Conclusioni 64

6 Conclusioni e sviluppi futuri **67**

Appendici **68**

A Grafici response time **69**

Bibliografia **82**

Elenco delle figure

2.1 Architettura concettuale del calcolatore 4
2.2 Memoria virtuale come percepita dall'applicazione 5
2.3 Astrazione e virtualizzazione delle risorse: (a) un disco viene suddiviso in file, (b) vengono percipiti più dischi virtuali anche se il disco fisico è uno solo 5
2.4 L'on-demand computing in risposta ad un picco di richieste . . . 8
2.5 Hypervisor type 1 e type 2 10
2.6 Full-virtualization 12
2.7 Paravirtualization 12
2.8 Gestione della memoria virtuale in 2 livelli 14
2.9 La virtualizzazione delle periferiche 15
2.10 Multiprocessore simmetrico costituito da più processori RISC . . 24
2.11 L'approccio sidecore in un ambiente virtualizzato 27
2.12 Configurazione dell'ambiente di test per SPECvirt-sc2010 30
2.13 Schema di funzionamento del test su di un singolo host 32

3.1 Configurazione dell'ambiente di testing 36
3.2 Workload sintetico: fase di esecuzione 39
3.3 Il tempo di esecuzione è O(n) rispetto al numero di richieste . . 40
3.4 Il tempo di esecuzione è O(h) rispetto alla dimensione del filtro h 40
3.5 Il tempo di esecuzione è O(p) rispetto al numero di stride p . . 41
3.6 Il tempo di esecuzione diminuisce all'aumentare della lunghezza della stride w 41

4.1 Modello chiuso per la rete di code 44
4.2 Scelta del numero di client per virtual machine 46
4.3 Throughput massimo con 16 virtual machine e 1 vcpu 49
4.4 Throughput massimo con 8 virtual machine e 2 vcpu 49
4.5 Throughput massimo con 4 virtual machine e 4 vcpu 50
4.6 Perdita percentuale con 16 virtual machine e 1 vcpu 50

4.7 Perdita percentuale con 8 virtual machine e 2 vcpu 51
4.8 Perdita percentuale con 4 virtual machine e 4 vcpu 51

5.1 Modello aperto per la rete di code 53
5.2 Andamento non lineare dell'utilizzo in funzione del throughput nel caso con hyperthreading 55
5.3 Confronto tra tecnologie per 16K - 1V con hyperthreading . . . 59
5.4 Confronto tra tecnologie per 16K - 1V senza hyperthreading . . 60
5.5 Xen per 16K - 1V con e senza hyperthreading: confronto tra $F = 0$ e $F = 0.75$ 61
5.6 KVM per 16K - 1V con e senza hyperthreading: confronto tra $F = 0$ e $F = 0.75$ 62
5.7 Hyper-V per 16K - 1V con e senza hyperthreading: confronto tra $F = 0$ e $F = 0.75$ 63

A.1 Confronto tra tecnologie per 8K - 2V con hyperthreading 69
A.2 Confronto tra tecnologie per 8K - 2V con hyperthreading 70
A.3 Confronto tra tecnologie per 4K - 4V con hyperthreading 71
A.4 Confronto tra tecnologie per 8K - 2V senza hyperthreading . . . 72
A.5 Confronto tra tecnologie per 4K - 4V senza hyperthreading . . . 73
A.6 Xen per 8K - 2V con e senza hyperthreading: confronto tra $F = 0$ e $F = 0.75$ 74
A.7 Xen per 4K - 4V con e senza hyperthreading: confronto tra $F = 0$ e $F = 0.75$ 75
A.8 KVM per 8K - 2V con e senza hyperthreading: confronto tra $F = 0$ e $F = 0.75$ 76
A.9 KVM per 4K - 4V con e senza hyperthreading: confronto tra $F = 0$ e $F = 0.75$ 77
A.10 Hyper-V per 8K - 2V con e senza hyperthreading: confronto tra $F = 0$ e $F = 0.75$ 78
A.11 Hyper-V per 4K - 4V con e senza hyperthreading: confronto tra $F = 0$ e $F = 0.75$ 79
A.12 VMware per 8K - 2V con e senza hyperthreading: confronto tra $F = 0$ e $F = 0.75$ 80
A.13 VMware per 4K - 4V con e senza hyperthreading: confronto tra $F = 0$ e $F = 0.75$ 81

Elenco delle tabelle

2.1 Come si ripartiscono le tecnologie in funzione dei tipi di hypervisor 13
2.2 Prodotti per la virtualizzione 16
2.3 Risultati scalando 1 tile 33

4.1 Perdite in percentuale nei casi di test per le 4 tecnologie 52

5.1 Calcolo dei periodi di campionamento nel caso bilanciato 57
5.2 Calcolo dei periodi di campionamento nel caso sbilanciato . . . 57
5.3 Utilizzi ripartiti equamente tra i guest per ogni test 58
5.4 Utilizzi ripartiti secondo il fattore $F = 0.75$ tra i guest per ogni test 58

Elenco dei procedimenti

1 Scelta dei parametri: secondsPerRun e numClients 45
2 Test A: Rilevamento throughput massimo senza hyperthreading 46
3 Test B: Rilevamento throughput massimo con hyperthreading . 47
4 Test C: Valutazione dei tempi di risposta 56

Capitolo 1

Introduzione

In questo elaborato di tesi si tratta un lavoro comparativo tra tecnologie di virtualizzazione con l'obiettivo di andare a valutarne le performance. Viene sviluppata un'attività di benchmarking per ottenere delle misurazioni utili quali il tempo di servizio, il throughput e i tempi di risposta per capire come i sistemi virtualizzati rispondono a certi tipi di carichi generati. L'obiettivo è quello di studiare il sistema e catturarne i comportamenti, in particolar modo quando si vanno a variare alcuni parametri di interesse, come il numero di virtual cpu per virtual machine, la memoria RAM assegnatavi ed altri.

Il workload è di tipo sintetico ed è stato implementato per stressare particolarmente la CPU, le cache e la memoria primaria. Sono stati implementati due generatori di carico: il primo si applica ai modelli chiusi per le reti di code, in cui l'obiettivo è misurare il throughput massimo ottenuto dalle diverse tecnologie di virtualizzazione; il secondo si descrive con i modelli aperti per le reti di code e si vanno a misurare i tempi risposta.

Le tecnologie di virtualizzazione che sono state scelte per questo benchmark sono: Citrix Xen, VMware ESXi, Microsoft Hyper-V e KVM.

In realtà questo lavoro si colloca in un disegno più ampio, il quale prevede la validazione di modelli per i sistemi virtualizzati; infatti sulla base dei risultati ottenuti sarà possibile fare un'analisi statistica per andare ad identificare dei modelli che possano descrivere il comportamento di questi sistemi.

Questo elaborato è strutturato nel modo seguente: il secondo capitolo riguarda lo stato dell'arte per i sistemi virtualizzati. Nel capitolo successivo vi è una spiegazione del generatore di carico implementato e del workload utilizzato. Seguono i capitoli in cui vengono presentati e commentati i risultati ottenuti relativamente al throughput massimo e ai tempi di risposta. Infine, si traggono alcune conclusioni sui risultati ottenuti e sui possibili sviluppi futuri.

Capitolo 2

Virtualizzazione: stato dell'arte

2.1 Definizione e principali tipologie

Il termine *virtualizzazione* è molto utilizzato in ambito informatico. Il significato più immediato ed intuitivo che se ne può dedurre è relativo ad un qualcosa che è il contrario di fisico, concreto; ossia una rappresentazione logica di ciò che viene espresso dalla realtà. Più nello specifico si può definire la virtualizzazione come (rif. [1]):

> *il disaccoppiamento del funzionamento logico delle risorse hardware e software dalla loro realizzazione fisica.*

Avendo questo in mente, il passo successivo è quello di cercare di capire a che *livello* si vuole operare la virtualizzazione. Per fare ciò è opportuno dapprima rappresentare a livello concettuale l'architettura del calcolatore e capire dove è possibile effettuarla.

Esistono vari livelli in cui si può introdurre la virtualizzazione. Come si può notare in figura 2.1, questi possono essere classificati principalmente nelle seguenti tipologie a seconda del grado di astrazione che si vuole introdurre:

- la virtualizzazione viene effettuata a livello di Instruction Set Architecture (ISA) e coincide con l'emulazione di tali istruzioni. Il processo di emulazione consiste nell'interpretare le istruzioni in modo completamente software. Per esempio, è possibile eseguire programmi compilati per un'architettura x86 su una macchina Sparc, grazie ad un emulatore x86.

- Un secondo tipo di virtualizzazione è introdotto tramite il Virtual Machine Monitor (VMM), il quale introduce un livello di astrazione tra l'OS e l'hardware sottostante. In questo modo il sistema operativo ha una visio-

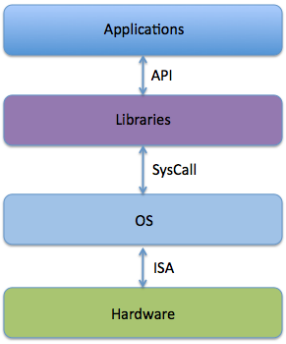


Figura 2.1: Architettura concettuale del calcolatore

ne virtualizzata dell'hardware, che non corrisponde alla reale composizione fisica dell'host (macchina fisica).

- Una terza classe di virtualizzazione avviene on-top-of OS o as-module-of OS e fornisce un'interfaccia virtualizzata per le system call.
- Un'ulteriore virtualizzazione si può effettuare a livello di librerie utilizzando le API (Application Programming Interface).
- Un'ultima tipologia avviene a livello di applicazione, ossia lo strato di virtualizzazione viene introdotto in un'applicazione (macchina virtuale) la quale può essere rappresentata da un interprete dei linguaggi come la Java Virtual Machine (JVM) o il Common Language Runtime (CLR).

In realtà esistono anche casi più semplici di come la virtualizzazione possa essere sfruttata ampiamente dal calcolatore: un esempio immediato riguarda l'utilizzo della memoria virtuale. Quando si scrive un programma, il programmatore non deve curarsi dello spazio di indirizzamento che viene riservato, ma tramite la Memory Management Unit (MMU) gli indirizzi logici (virtuali) vengono tradotti in indirizzi fisici. Lo spazio di indirizzamento, come percepito dal programma, è diverso da quello fisico (fig.2.2). La MMU si avvale di due strutture dati per questo compito: il Translation Lookaside Buffer (TLB) e la Paging Table (PT). In sostanza, il TLB è un buffer - o in implementazioni più complicate una cache della CPU - che contiene parti della PT, che risiede in memoria primaria o su disco e si occupa di tenere traccia della mappatura tra indirizzi logici e indirizzi fisici.

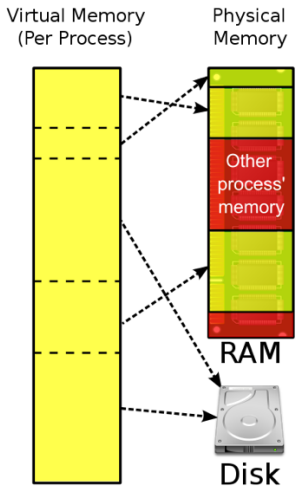


Figura 2.2: Memoria virtuale come percepita dall'applicazione

Esiste inoltre un terzo modo di operare la virtualizzazione, ed è quello che coincide con l'astrazione delle risorse. In particolare si tende ad astrarre risorse di storage, quali, ad esempio, i dischi (fig.2.3). In questo modo è possibile percepire il disco in modalità diverse a seconda di come vi si applica la virtualizzazione [2]: in generale si parla di *sharing* quando più risorse virtuali condividono un'unica risorsa fisica, mentre si parla di *aggregation* quando vengono agglomerate più risorse fisiche a costituire una unica grande risorsa virtuale.

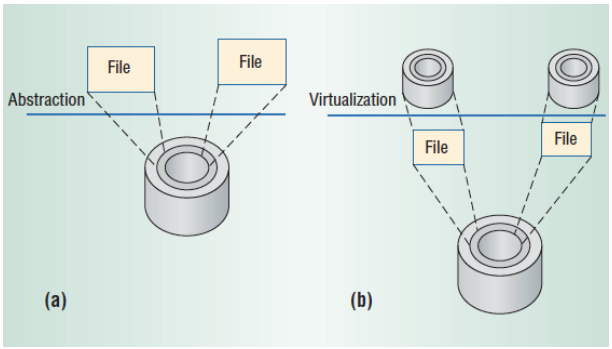


Figura 2.3: Astrazione e virtualizzazione delle risorse: (a) un disco viene suddiviso in file, (b) vengono percipiti più dischi virtuali anche se il disco fisico è uno solo

A fini del lavoro di tesi, il livello che più ci interessa rappresentare è quello relativo all'introduzione del VMM, detto anche *hypervisor*, il quale può essere

implementato in vari modi a seconda dei benefici che si vogliono trarre dal processo di virtualizzazione. Prima di addentrarci però nella spiegazione delle varie tecniche di implementazione, è opportuno citare i passi che hanno condotto ad un vero e proprio boom della virtualizzazione dai tardi anni '90 in poi, per capirne le ragioni ed in seguito individuarne i benefici e gli svantaggi introdotti.

2.2 Benefici e svantaggi introdotti

Come notato nel precedente paragrafo, si è ricorsi alla virtualizzazione principalmente per far fronte allo spreco di hardware sottoutilizzato ed agli elevati costi di gestione che la precedente politica aveva causato. Per quanto concerne l'impatto dovuto alla virtualizzazione, è opportuno chiedersi in che modo e se veramente ha introdotto benefici. E' chiaro che se dei vantaggi sono stati introdotti, si è dovuto pagare qualcosa in altri termini, e l'attività di ricerca odierna è volta a migliorare questi aspetti. Sicuramente alcuni vantaggi di evidente importanza sono il porting del codice su differenti sistemi, la migrazione di macchine virtuali, il checkpointing e in generale la server consolidation [3]. Più nello specifico, si può certamente dire che la virtualizzazione ha portato essenzialmente i seguenti benefici [4]:

- **Isolamento**: se un'applicazione all'interno di una macchina virtuale mostra dei bug, o se addirittura il sistema operativo mostra delle problematiche, tali errori non si ripercuotono nè sull'host che ospita tale macchina virtuale, nè sulle altre macchine virtuali (dette anche *guest*) [5].

- **Flessibilità**: su ogni macchina virtuale è possibile effettuare operazioni del tipo: boot, shutdown, resume, pause. Inoltre è possibile modificare le specifiche di ciascuna di esse in termini di virtual cpu, dischi, network e memoria.

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

- **Scalabilità**: aggiungere o rimuovere nodi è un compito che risulta molto più facile e sbrigativo. Se la domanda per la capacità da sostenere aumenta nel tempo, è facile installare un nodo fisico con una installazione di base che metta in esecuzione le macchine virtuali ed i loro servizi.

- **Utilizzo dell'hardware**: come visto, per la server consolidation, ora la quasi totalità di hardware presente in un nodo fisico viene sfruttata appieno.

- **Sicurezza**: l'isolamento comporta anche maggiore sicurezza poichè ogni virtual machine, in genere associata ad un solo servizio, non compromette lo svolgimento di altri servizi che risiedono su altri guest.

- **Costi**: ridotti in quanto viene comprato e manutenuto meno hardware (più potente rispetto al passato), viene coinvolto meno personale nella gestione dei server, lo spazio nelle sale server viene meglio sfruttato, le licenze software vengono ridotte.

- **Monitoraggio**: tramite l'introduzione dell'hypervisor, si introduce la possibilità di osservare il comportamento dei guest da fuori ed operare di conseguenza un monitoraggio continuo delle applicazioni in esecuzione [3].

- **Adattabilità a variazioni di workload**: è possibile far fronte a picchi di workload semplicemente riorganizzando l'allocazione delle risorse alle virtual machine. E' anche possibile rendere automatico tale processo.

- **Applicazioni Legacy**: nel caso si voglia decidere di cambiare OS, le applicazioni in esecuzione originariamente possono essere eseguite anche sul nuovo OS.

D'altro canto non si può fare a meno di notare alcuni svantaggi che l'introduzione della virtualizzazione comporta. Per lo più questi possono essere riassunti nei seguenti punti:

- **Overhead**: introducendo flessibilità anche una certa quantità di overhead viene prodotto, il che comporta in genere un sensibile calo delle performance rispetto al sistema nativo;

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

- **Interfaccia di gestione**: dipendente dalla piattaforma di virtualizzazione adottata.

2.2.1 Come valutare i benefici

Come visto, il primo passo nell'adottare la virtualizzazione è consistito nell'effettuare la server consolidation. Il secondo passo che si vuole affrontare in questi anni è quello relativo all'on-demand computing [3]: il problema che ci si pone in questo contesto è quello di poter fornire un ammontare di risorse adeguato per la domanda a cui si deve far fronte in un certo periodo. Se un'organizzazione A, ad esempio, sta affrontando un certo picco di domande, l'organizzazione B, le cui risorse sono sottoutilizzate, deve essere in grado di accettare un carico di richieste da parte di A, compiendo una sorta di compensazione del carico (fig. 2.4).

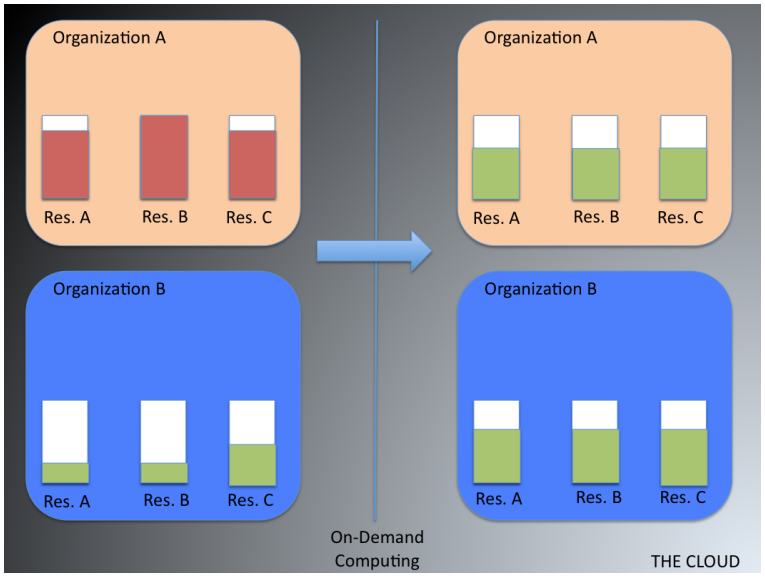


Figura 2.4: L'on-demand computing in risposta ad un picco di richieste

Al fine di valutare le tecniche di virtualizzazione oggi in commercio, è opportuno fissare alcuni criteri di valutazione, in base ai quali tali tecnologie possono essere confrontate. Secondo l'approccio poc'anzi descritto, alcuni dei criteri di valutazione che si devono tener presente sono:

- **Performance**: le prestazioni sono di vitale importanza. In generale l'assunzione di tecniche di virtualizzazione, introducendo overhead, può indurre ad un calo di prestazioni; oggi come oggi si accetta che tale calo rientri in una certa soglia, ma in alcuni ambiti, come l'High Performance Computing (HPC), si tenta di ridurlo al minimo.

- **Trust**: ci si basa sul fatto che il VMM sia abbastanza affidabile, immune da attacchi (sicuro) ed abbia bisogno di minor codice per essere implementato; seppur garantisca minori performance in relazione ad un OS che, a fronte di una maggiore richiesta di codice, garantisce migliori performance ma è più soggetto a bug di sicurezza.

- **Portabilità**: a seconda del tipo di virtualizzazione che si vuole utilizzare il supporto legacy può essere raggiunto in diversi modi. Con la full virtualization (paragrafo 2.3), i guest OS non hanno bisogno di essere modificati e la portabilità è garantita; tramite paravirtualization (paragrafo 2.3), si deve pagare in termini di modifica dei guest OS, che però sono in grado di sfruttare meglio le risorse.

2.3 Tecniche di virtualizzazione

Per come è stata presentata la virtualizzazione, in generale si pensa che sia solo un modo per rendere possibile l'esecuzione di più sistemi operativi contemporaneamente. Si pensi per esempio a soluzioni per una virtualizzazione del tipo OS-level come Solaris Zones Containers o IBM AIX Partitions in cui vi è un solo kernel che consente di isolare diverse istanze in user mode [6] [7].

In realtà in questa sede non è tanto il sistema operativo che ci interessa, quanto lo strato di virtualizzazione che rende possibile l'esecuzione di più sistemi operativi, indipendemente da quale di essi si scelga poi effettivamente. Si può dire che un ambiente virtuale è composto da un insieme di periferiche hardware implementate via software, che prende poi il nome di *macchina virtuale*. Tale macchina virtuale permette l'esecuzione di un OS ospite (host) ed una o più applicazioni che eseguono sopra di esso. Più macchine virtuali vengono eseguite a loro volta sopra uno strato di virtualizzazione detto hypervisor (supervisore). Ne esistono essenzialmente due tipologie [8]:

- **Tipo 1: bare-metal, nativo**;
  - **monolitico**;
  - **microkernel**.
- **Tipo 2: hosted, ospitato**.

Il primo risiede direttamente sull'hardware della macchina e ne possiede il controllo diretto e privilegiato: si parla in tal caso di virtualizzazione nativa. Si ottiene più affidabilità e sicurezza in quanto, essendo l'hypervisor implementato

attraverso l'utilizzo di un minor numero di linee di codice, offre meno esposizione a rischi. Il secondo tipo risiede su un sistema operativo ospitante, come ad esempio Windows o Linux (fig. 2.5). Risulta chiaro come evidentemente la prima topologia possa offrire delle prestazioni migliori, in quanto è a stretto contatto con l'hardware. La soluzione microkernel based utilizza ancora meno codice e, a discapito del fatto che possa essere ancora più efficiente, fornisce meno driver per le periferiche, creando possibili incompatibilità con l'hardware sottostante che ne precludono l'installazione. Entrambe possiedono la capacità di implementare macchine virtuali che, dal punto di vista del file system, qualunque esso sia, vengono viste come file (file della RAM, file dell'hard-disk, ecc.).

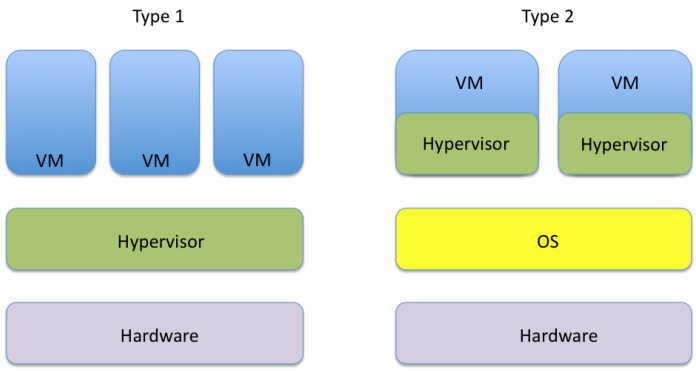


Figura 2.5: Hypervisor type 1 e type 2

2.3.1 La macchina virtuale

Una macchina virtuale può essere creata in più modi: creandone una nuova, trasformando una macchina fisica in una virtuale, oppure clonandola da una già esistente. Le relative caratteristiche sono principalmente le seguenti:

- può essere eseguita in paralello con altre macchine virtuali sfruttando appieno le risorse dell'host;
- può eseguire le applicazioni in completo isolamento;
- può essere clonata, migrata, archiviata o ripristinata;
- la migrazione avviene tra medesimi supporti di virtualizzazione;
- può essere creata tramite template al fine di velocizzare il processo.

2.3.2 Virtualizzazione dell'architettura x86

La sfida maggiore si è verificata a cavallo degli anni '90, quando il problema che si presentava era quello di virtualizzare l'architettura più diffusa in commercio (la x86), che, come detto, non era stata pensata in origine per esserlo. In sostanza era necessario trovare un modo per virtualizzare le 3 principali componenti di questa architettura: CPU, memoria e periferiche.

Virtualizzazione della CPU

Quando un sistema operativo viene installato "crede" di avere pieno possesso delle risorse hardware. Infatti esso le gestisce in maniera diretta e privilegiata. Introducendo però uno strato di virtualizzazione, tale contatto viene meno e, a questo punto, è lo strato introdotto che deve occuparsi di gestire le richieste fatte all'hardware per conto dell'OS, il quale continua a credere di avere diretto contatto con le risorse. Come anticipato però, alcune operazioni necessitano di essere eseguite direttamente sull'hardware, perchè altrimenti si possono generare dei problemi che causano l'instabilità del sistema.

Per ovviare a questo problema si è ricorsi alla *binary translation* [9] [10], la quale si occupa di tradurre le istruzioni da un instruction set source ad un instruction set target. Chiaramente tale procedimento introduce overhead, ma di fatto questa soluzione ha permesso la virtualizzazione dell'x86. Per ridurre l'overhead si è passati dall'implementazione di una binary translation di tipo statico ad una di tipo dinamico, dove al posto di tradurre singole istruzioni, ne vengono raggruppate un certo numero in gruppi (detti chunk), che vengono poi tradotti ed inviati dal source al target. Tale processo determina una diminuzione dell'overhead introdotto ed un miglioramento delle performance.

Più in generale, ad oggi, esistono essenzialmente 2 modi per operare la virtualizzazione di questa architettura e sono: *full-virtualization* e emphparavirtualization.

La prima rappresenta di fatto una emulazione dell'hardware sottostante e si ha il VMM che agisce al di sopra di un sistema operativo installato sull'hardware. In questa configurazione il VMM viene eseguito come un'applicazione nello user space. Ai guest viene offerta una replica precisa dell'hardware sottostante della macchina fisica ed il grande vantaggio è che non viene richiesta una modifica del sistema operativo in uso nelle macchine virtuali. Tale configurazione si può osservare in figura 2.6.

Al contrario della full-virtualization, nella paravirtualization i sistemi operativi guest in esecuzione devono essere modificati in modo che possano opera-

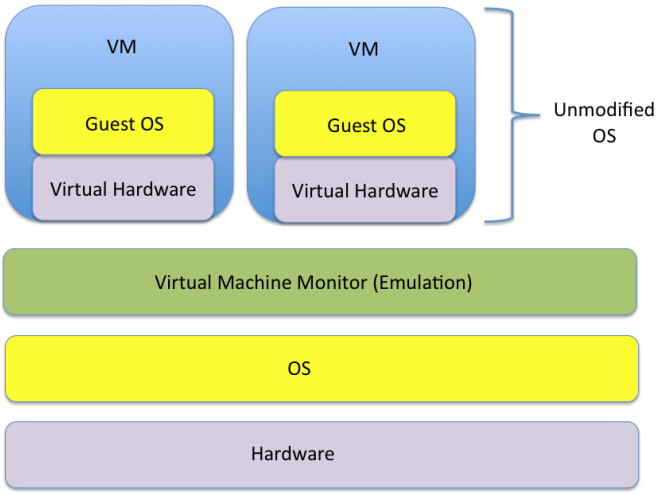


Figura 2.6: Full-virtualization

re nell'ambiente virtualizzato. Lo strato virtualizzato è in completo controllo dell'hardware ed agisce in modalità privilegiata (ring 0). Quindi il guest è a conoscenza di essere in esecuzione in un ambiente virtualizzato. Il VMM è implementato in maniera veramente essenziale e ciò permette di raggiungere quasi le stesse perforamance di un sistema nativo (ossia non virtualizzato). Questa configurazione è illustrata in figura 2.7.

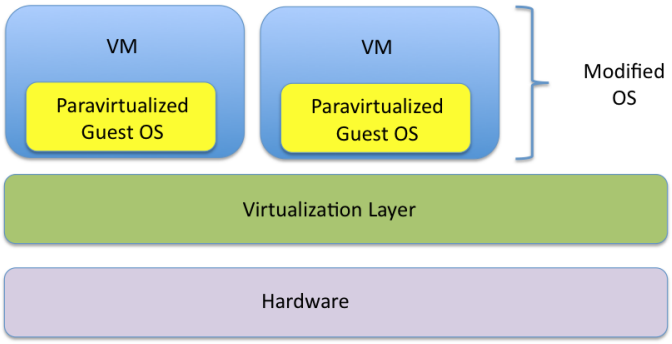


Figura 2.7: Paravirtualization

Esiste poi la *hardware-assisted virtualization* che è in realtà una variante della full-virtualization e, al posto di utilizzare la binary translation, gestisce le sensitive instruction dell'instruction set, utilizzando un modello trap-and-

emulate nell'hardware, in contrapposizione a quello software. In questo modo il VMM è in grado di virtualizzare in modo migliore l'architettura x86. Se da una parte riduce l'OH introdotto per la modifica dei guest OS che si può avere in una soluzione paravirtualizzata, dall'altra viene richiesto un supporto specifico da parte della CPU che non è sempre disponibile nelle architetture x86. Questo approccio, da un punto di vista teorico, risulta essere promettente e recentemente diversi esperimenti in tal senso sono stati effettuati.

A titolo esemplificativo viene mostrata in tabella 2.1 la collocazione delle tecnologie utilizzate nell'ambito di questa tesi:

	Full-virtualization		**Paravirtualization**
	Binary Translation	**HW assisted**	
bare-metal	VMWare ESX VMware Workstation Hyper-V	VMware ESX Hyper-V Xen	Xen
hosted	VMware Workstation	VMware workstation KVM	-

Tabella 2.1: Come si ripartiscono le tecnologie in funzione dei tipi di hypervisor

Virtualizzazione della memoria

Fondamentalmente il concetto principale è che ogni sistema operativo guest possiede la propria MMU, la quale, come spiegato in precedenza, si occupa di mappare indirizzi virtuali in indirizzi fisici. Il problema è che questi ultimi sono a loro volta virtuali e, essendo la memoria fisica dell'host situata ad un livello più basso, è compito del VMM trasformare gli indirizzi virtuali (ritenuti fisici dai guest) in indirizzi fisici veri e propri. Si ha quindi una gestione della memoria virtuale a due livelli (figura 2.8).

E' chiaro che in una configurazione di questo tipo si introduce overhead, ed è per questo motivo che è stata pensata una nuova page table, detta Shadow Page Table (SPT) [11]. Tale SPT si occupa di trasformare direttamente gli indirizzi virtuali dei guest in indirizzi fisici dell'host, bypassando di fatto il passo intermedio. Nonostante questo approccio però, la necessità di mantenere coerenza tra le PT e la SPT ha portato all'esigenza di fornire dell'hardware dedicato per la virtualizzazione della MMU e nella nuova generazione di processori, questo prende il nome di Extended Page Table (EPT) [12]. In questo modo si vuole im-

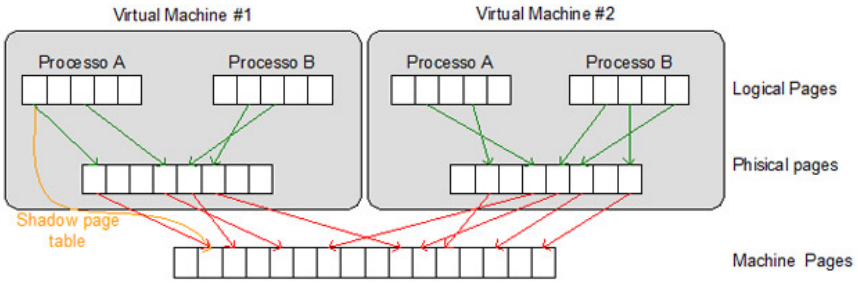


Figura 2.8: Gestione della memoria virtuale in 2 livelli

plementare la mappatura tra indirizzi virtuali dei guest e indirizzi fisici dell'host senza che l'hypervisor intervenga via software sulla MMU.

Virtualizzazione delle periferiche

A ciascuna delle macchine virtuali viene fornito un insieme di periferiche standard emulate come: DVD, tastiera, scheda video, periferiche USB, scheda di rete, scheda audio, ecc. La scelta di non specificare le periferiche emulate viene fatta per poter garantire portabilità verso altre piattaforme; infatti quando avviene la migrazione di una macchina virtuale, se le periferiche emulate scelte sono le più standard possibili, maggiore sarà la possibilità che tale macchina possa essere eseguita anche attraverso hardware che non corrisponde esattamente a quello di partenza.

In generale, la gestione da parte del VMM delle periferiche fisiche con quelle virtualizzate avviene nel seguente modo (figura 2.9):

- la periferica fisica comunica con il VMM tramite DMA (Direct Memory Access) e scrive in un buffer proprio del VMM;
- quando la copia dei dati giunge al termine, il VMM viene avvisato dalla periferica fisica tramite una interrupt;
- il VMM esamina i pacchetti ed individua la macchina virtuale di destinazione. Una volta identificata, copia i pacchetti nel buffer della macchina virtuale;
- la periferica virtuale della macchina avvisa il relativo guest OS che dei dati sono arrivati.

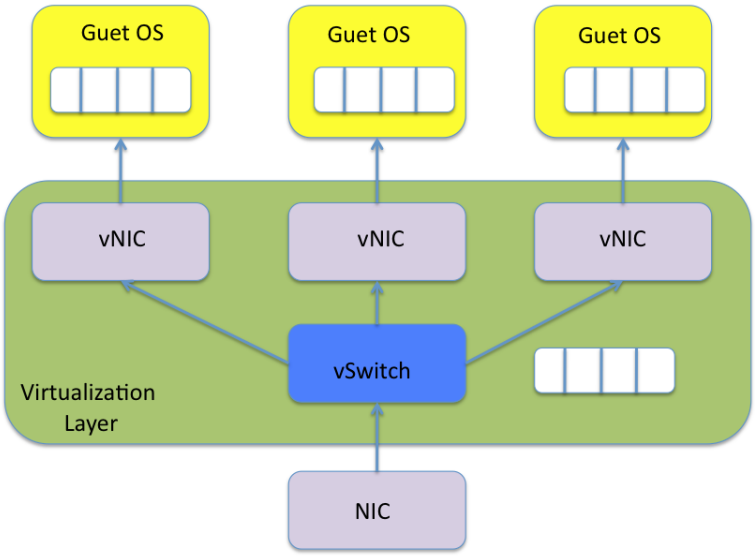


Figura 2.9: La virtualizzazione delle periferiche

Da ultimo è opportuno fare una distinzione tra periferiche emulate, come sopra descritte, e driver paravirtualizzati per soluzioni full-virtualized: in questo caso un disco o una scheda di rete, per esempio, interagisce direttamente con la piattaforma di virtualizzazione (senza emulazione) per la gestione efficiente degli accessi al disco o alla rete, permettendo un'esecuzione che raggiunge quasi le performance native, anche se si opera in un sistema virtualizzato.

2.4 Maggiori prodotti disponibili nello specifico

In questo paragrafo vogliamo soffermarci sulle principali aziende che offrono prodotti per la virtualizzazione. Oggi come oggi è sicuramente VMware il leader incontrastato nel settore, ma negli ultimi anni anche altre aziende stanno emergendo, offrendo nuove soluzioni e cercando di migliorare quelle già proposte. In tabella 2.2 troviamo invece illustrata una tabella riassuntiva dei prodotti oggi disponibili.

2.4.1 VMware

Fondata nel 1998 con circa una ventina di dipendenti, VMware Inc. è stata la prima azienda a proporre un prodotto volto alla virtualizzazione. Infatti nel

Progetti e Aziende	Xen	KVM	Hyper-V	VMware	OpenVZ	Parallels	FreeBSD Jails	Solaris Zone
Prima release	2003	2007	2008	1999	2006	2005	2000	2005
Tecnologia	Paravirtualization	Full virtualization	Full virtualization	Full virtualization Paravirtualization	OS-level	Full virtualization Paravirtualization	OS-level	OS-level
Hypervisor	bare-metal	hosted	bare-metal	bare-metal hosted	-	bare-metal hosted	-	-
Prodotti Desktop	XenApp XenDesktop	-	Virtual PC	Fusion Player Workstation	-	Desktop Desktop for Mac	-	-
Prodotti Server	XenServer	unix kernel included	Windows Server 2008 R2 Virtual Server 2005	ESX ESXi GSX	unix kernel included	Server Server for Mac Virtuozzo Containers	unix kernel included	Solaris included
Categoria	open-source licensed	open-source licensed	licensed	licensed	open-source licensed	licensed	open-source	licensed

Tabella 2.2: Prodotti per la virtualizzione

maggio 1999 si presenta sul mercato con WMware Workstation [13] e, successivamente nel 2001, con WMware GSX Server (hosted) e WMware ESX Server (bare-metal).

Il core business dell'azienda consiste nell'offrire soluzioni per virtualizzare le risorse di un sistema fisico, al fine di poterle meglio sfruttare attraverso più macchine virtuali installate su un'unica macchina fisica. Attraverso l'utilizzo di driver standard, permette la portabilità delle macchine virtuali tra più sistemi. Ad oggi rappresenta di fatto il metro di giudizio con cui le aziende concorrenti vegono valutate. I prodotti che propone possono essere suddivisi tra prodotti desktop e prodotti server.

Prodotti Desktop

- **Workstation**: fornisce la possibilità di eseguire macchine virtuali sulla piattaforma x86. Tali macchine virtuali possono essere installate su un sistema operativo Windows o Linux e all'interno di esse si può installare qualsivoglia OS.
- **Fusion**: è un prodotto software sviluppato per sistemi Macintosh basati su processori Intel. Mac OS X funge da sistema operativo host e permette l'esecuzione di più guest basati su architettura x86.
- **Player**: è la versione non-commerciale (freeware) di Workstation che ha permesso a molti utenti di accostarsi al mondo della virtualizzazione.

Prodotti Server

- **ESX Server**: è un prodotto che implementa una soluzione bare-metal per l'hypervisor. Riduce di molto l'overhead, rispetto per esempio a GSX Server, in quanto viene eseguito direttamente sopra l'hardware, gestendo direttamente le risorse assegnate ai server virtuali. Assieme a vCenter permette di gestire cluster di macchine virtuali distribuite su più host fisici in modo flessibile ed affidabile. Alcune delle peculiarità sono la possibilità di migrare un server virtuale in esecuzione su un altro host tramite vMotion, muovere hardware virtuale tramite Storage vMotion e, in caso di guasto su un nodo del cluster, il restart automatico del server virtuale su un altro host del cluster (HA - High Availability).
- **ESXi Server**: rappresenta di fatto la copia freeware di ESX Server. Viene solamente rimossa la console di gestione che viene sostituita con una minima interfaccia di gestione BusyBox.

- **GSX Server**: implementa un hypervisor di tipo hosted e permette l'esecuzione di più macchine virtuali. Infatti può essere eseguito su sistemi operativi Linux o Windows esistenti.

Altri prodotti per la gestione che vanno ricordati sono:

- *vSphere*: definito anche "Cloud OS", ha la capacità di gestire grandi pool di infrastrutture, sia software che hardware, sia da reti esterne che da reti interne;
- *vCenter Converter*: permette di trasformare una macchina fisica in una macchina virtuale (P2V), oppure una macchina virtuale di un tipo in un altro tipo (V2V);
- *ThinApp*: è una suite di virtualizzazione che consente la portabilità delle applicazioni. Questo software permette di eseguire applicazioni anche se non precedentemente installate.

2.4.2 Citrix

Citrix Systems Inc. viene fondata nel 1989 da Ed Iacobucci con lo scopo primario di implementare una nuova versione del sistema operativo OS/2, originariamente creato da Microsoft ed IBM e poi esclusivamente da IBM. L'obiettivo non era quindi relativo a prodotti per la virtualizzazione, anche perchè la ricerca non aveva ancora dato i suoi frutti in tal senso. L'IBM finanziava i progetti di Citrix, ma non essendo interessata alle vision di Iacobucci, quest'ultimo lasciò la guida dell'azienda nelle mani di Roger Roberts che, tra il 1990 e il 1993, tentò di rilanciare Citrix in un mercato competitivo senza però riuscirci.

Fu con l'acquisto di Netware Access Server da Novell, rinominato WinView sotto marchio Citrix, che l'azienda vide il suo primo prodotto di successo nel 1993. Nel 2007 entra nel mercato della virtualizzazione acquistando il prodotto Xen originariamente sviluppato dal laboratorio di informatica dell'Università di Cambridge nel 2003. Nato come prodotto open source, esso viene ancora sviluppato in tal modo, me è anche affiancato da prodotti commerciali per l'ambiente enterprise.

Il progetto Xen [14] nasce in risposta all'esigenza di poter meglio sfruttare le risorse dell'host su cui si applica la virtualizzazione. Fino ad allora infatti, l'unico modo proposto per virtualizzare la CPU era quello relativo alla full-virtualization tramite binary translation (rif. 2.3). Questo progetto propone quindi la paravirtualization in risposta ai problemi di overhead introdotti dalla precedente soluzione. Successivamente, il supporto alla paravirtualization

è stato affiancato da un'implementazione della full-virtualization basata sulle estensioni dell'architettura x86. Ad oggi si può dire che nel complesso Citrix propone soluzioni desktop e server per la virtualizzazione, SaaS (Software As A Service) e tecnologie per il Cloud Computing.

I prodotti principali in commercio sono:

- **XenApp**: ha lo scopo di proporre agli utenti un modo per connettersi a dei server centrali su cui sono installate delle applicazioni che possono essere utilizzate senza il bisogno di essere installate sul proprio computer (client). E' quindi possibile connettersi dall'esterno della rete aziendale tramite laptop, smartphone, service access point aeroportuali, ecc.

- **XenDesktop**: permette la separazione di sistemi operativi, applicazioni e hardware. Queste vengono proposte in livelli separati a seconda delle esigenze dell'utente. E' in grado di supportare altre tecnologie di virtualizzazione come anche la maggioranza dei sistemi operativi.

- **XenServer**: si basa sull'hypervisor paravirtualizzato spiegato in precedenza (rif. 2.3) che fornisce overhead basso e performance quasi simili a quelle di un sistema operativo nativo. Opera insieme a XenMotion e Resource Pools al fine di migrare a caldo macchine virtuali da un host fisico ad un altro.

2.4.3 Microsoft

Microsoft Corporation acquisisce il suo primo prodotto per la virtualizzazione da Connectix nel 2004, ancor prima che esso fosse messo in commercio. Si tratta di **Virtual Server 2005 R2** (tipo hosted) ed è un prodotto che permette infatti la virtualizzazione dei server. Inizialmente permetteva solamente la creazione di macchine virtuali in cui il guest OS fosse solo Windows, ma in seguito, tramite il SP1, venne introdotta anche la possibilità di installare sistemi operativi basati su kernel Linux. In concomitanza con questo prodotto, nel luglio del 2006 viene anche rilasciata la prima versione di **Virtual PC**, un prodotto desktop. In realtà una versione basata su binary translation era già stata rilasciata in precedenza e successivamente, con la versione del 2006, è stata introdotta anche la hardware assisted virtualization. Questo prodotto oggi viene distribuito come freeware.

L'azienda si ripropone nel mercato della virtualizzazione essenzialmente con due versioni dello stesso prodotto: una versione free chiamata **Microsoft Hyper-V Server 2008 R2** [15] ed una versione sotto licenza inclusa in Windows Server 2008 R2. La versione stand-alone viene rilasciata nel giugno 2008.

Hyper-V è fondamentalmente una soluzione basata su hypervisor bare-metal il quale viene situato nel ring -1, mentre il ring 0 viene suddiviso in partizioni. Una prima partizione definita "parent" è quella in cui risiede il kernel Windows e gestisce le partizioni definite "child". La partizione parent ha accesso diretto all'hardware e crea le partizioni child tramite le hypercall API; queste altro non sono che le application programming interface fornite da Hyper-V. I guest hanno una visione virtualizzata delle risorse hardware, e le operazioni come le interrupt sono gestite dall'hypervisor. I sistemi operativi che possono essere eseguiti sui guest sono essenzialmente Windows oppure Linux e solo le versioni a 64 bit. Inoltre viene fornito supporto per la paravirtualizzazione ai guest sui quali vengono eseguiti kernel Linux.

Per la gestione dei cluster composti da host fisici, viene fornita una interfaccia di gestione chiamata Hyper-V Manager, attraverso la quale è possibile gestire le risorse virtuali fornite ad ogni singola macchina virtuale; è inoltre possibile stabilire dei pesi per ciascuna virtual machine in termini di percentuale di risorse disponibili.

2.4.4 KVM

La sviluppo di KVM (Kernel-based Virtual Machine) nasce ad opera di Qumranet come progetto di sviluppo di nuove tecnologie nell'ambito della virtualizzazione e viene in seguito comprato e finanziato da Red Hat Inc., un'azienda che viene fondata nel 1995 da Bob Young. Questa azienda venne acquistata dallo stesso Young da Marc Ewing, il quale aveva precedentemente creato una nuova distribuzione Linux chiamata appunto Red Hat Linux. Tale azienda è nata con l'intento di sviluppare software libero, ma è con un prodotto per le aziende, chiamato Red Hat Enterprise Linux (RHEL), che seppe imporsi sul mercato. Contribuisce tutt'ora allo sviluppo di molti progetti per la distribuzione di software libero e fra questi possiamo citare Fedora, che è di fatto il ramo testing per il software Red Hat.

KVM [16] presenta una soluzione che implementa l'hypervisor all'interno del kernel di un sistema operativo. Nell'architettura x86 esistono due modi con differenti privilegi di operare: il kernel-mode e lo user-mode. KVM si propone di inserire un terzo mode chiamato appunto guest-mode per l'esecuzione delle macchine virtuali. Quello che si intende gestire in modo virtualizzato con il guest-mode è tutto ciò che non ha a che fare con codice relativo ad operazioni di input e output, che viene emulato attraverso altri programmi user-space come QEMU. A sistemi operativi guest come Linux e Windows viene anche fornito supporto alla paravirtualizzazione tramite driver paravirtualizzati.

A discapito della credenza diffusa in questi anni che la virtualizzazione possa essere una sorta di toccasana per tutte le aziende, in realtà è una scelta che va ponderata nel migliore dei modi, previo uno studio di fattibilità adeguato. Infatti, come in tutte le cose, va sempre tenuta presente la dimensione nella quale si deve operare: per aziende con piccoli datacenter, la virtualizzazione potrebbe garantire un ROI in termini di risparmio energetico relativamente basso, mentre sarà alto per grandi datacenter. D'altro canto però, per grandi datacenter, il numero di licenze utilizzate sarà più elevato rispetto a piccoli datacenter, causando una perdita iniziale ben più considerevole.

2.5 Virtualizzazione e benchmarking

Valutare le prestazioni dei server è un compito di primaria importanza nell'area informatica. Effettuare un'attività di *benchmark* del server nasce come esigenza di valutare e confrontare diverse tecnologie ed architetture tra loro, per estrapolare delle informazioni utili per attività quali capacity planning, provisioning e consolidation. Più formalmente un benchmark è un insieme di condizioni che vengono utilizzate come riferimento (rif. [17]). Infatti si fa ampio utilizzo di questo termine anche in ambiti diversi, come ad esempio l'economia e la finanza.

Principalmente, quello che si vuole valutare in ambito informatico sono le quattro caratteristiche essenziali di ogni server, ossia: cpu, memoria, dischi e network. Fondamentalmente, le varie tecniche di valutazione si basano sull'esecuzione di vari test esaustivi di questi quattro componenti, al fine di identificare il massimo carico di lavoro che possono sopportare. Sulla base dei risultati ottenuti viene stilato in genere un documento all'interno del quale viene attribuito un punteggio in base al carico di lavoro raggiunto; tale punteggio viene poi usato come riferimento per poter essere confrontato con altri server, eseguendovi lo stesso tipo di test effettuato in precedenza.

Esistono essenzialmente due enti che garantiscono il rispetto dei punteggi e le modalità standard di svolgimento dei test: Standard Performance Evaluation Council (SPEC) [18] e Transaction Processing Performance Council (TPC) [19].

Il primo è un'organizzazione no-profit, i cui membri possono essere venditori hardware, aziende software, università, organizzazioni di ricerca, integratori di sistemi, pubblicitari, consulenti e ha l'obiettivo principale di mantenere ed approvare un insieme di benchmark standard per sistemi informatici. Un esempio di tecnica di benchmark disponibile può essere per esempio SPECcpu2006: si focalizza sulle performance della CPU, della memoria e del compilatore. Esistono due sottotipologie relative a compute-intensive integer performance (CINT2006)

e compute-intensive floating point performance (CFP2006). SPEC fornisce i benchmark sotto forma di codice che deve essere compilato, e questa compilazione può essere di vari tipi: ad esempio valutando la SPEED (quanto velocemente viene eseguito un task) oppure la RATE (quanti task un computer può realizzare in un certo periodo di tempo).

TPC è anch'esso un ente no-profit fondato principalmente per definire benchmark di database e processi transazionali, al fine di offrire dati obiettivi e verificabili all'industria. TPC concepisce la transazione come uno scambio commerciale di beni, servizi oppure denaro. I benchmark relativi alle performance di database transazionali hanno come scopo quello di comunicare quante transazioni un certo sistema è in grado di compiere nell'unità di tempo. Il benchmark che viene più utilizzato è TPC-C. Esso si basa su un sistema OLTP (On-Line Transaction Processes) in cui vengono considerate diverse tipologie di transazioni: new order, payment, delivery, order status e stock level.

2.5.1 Lavori relativi a valutazione delle performance in sistemi virtualizzati

La maggior parte dei lavori effettuati tendono a confrontare alcune tecnologie tra di loro utilizzando diversi benchmark, il più delle volte benchmark standard in modo che possano essere riutilizzati da altri ricercatori. Questo confronto si basa in generale sui seguenti aspetti:

- come la virtualizzazione di un host differisce dall'esecuzione di un sistema nativo;
- dove viene introdotto overhead;
- come diversi OS interagiscono con tecnologie di virtualizzazione;
- quali possibili soluzioni possono fare convergere sistemi virtualizzati a sistemi nativi dal punto di vista delle performance;
- quali sono le differenze sostanziali tra le tecnologie.

In questo contesto diversi lavori sono stati svolti. Infatti, in [13] viene verificato se una VM con un certo sistema operativo installato sia o meno un ambiente di testing e sviluppo valido. Vengono utilizzate varie configurazioni, sistemi nativi Ubuntu e Windows, Ubuntu (guest) su Windows, Windows (Guest) su Ubuntu, Windows (guest) su Windows e Ubuntu (guest) su Ubuntu. Il test si suddivide fondamentalmente in tre parti: la prima parte consiste nel misurare quanto tempo ciascuno dei guest impiega a creare 500.000 oggetti Java

rispetto ai sistemi nativi; la seconda parte agisce sull'adapter network in cui si vanno ad effettuare 99.900 query tramite JDBC a dei database installati, misurando il tempo impiegato nelle varie configurazioni; mentre nell'ultimo caso si valuta il tempo che ci vuole per leggere e scrivere da file (I/O).

Un altro tipo di esperimento è indirizzato invece al confronto tra due tecnologie di virtualizzazione, come ad esempio Xen e OpenVZ in [20]. Avviene in questo caso la consolidation di uno o più sistemi multi-tiered su uno o due nodi e viene effettuato su di essi un certo tipo di workload chiamato RUBiS. Xen ed OpenVZ vengono valutati in termini di performance delle applicazioni, consumo delle risorse, scalabilità, metriche come cache-miss rate e il consumo di risorse del "Dominio-0" in Xen.

Altri esperimenti ancora valutano le differenze tra diverse tecnologie di virtualizzazione con un sistema operativo nativo. E' questo il caso di [21] in cui Xen e KVM vengono valutati tra loro ed anche rispetto a Linux nativo. Viene qui valutato il grado di isolamento che possono offrire le due tecnologie di virtualizzazione, le perfomance complessive derivanti dalla compilazione del kernel per impegnare la CPU, la scalabilità in termini di quante virtual machine è in grado di mandare in esecuzione una certa tecnologia senza accusare perdite di performance.

Infine, mentre per quanto riguarda la server consolidation, le diverse tecnologie sono in grado di offrire soluzioni pressochè simili, esiste un ramo della ricerca che è oggi particolarmente impegnato nel valutare l'impatto che ha la virtualizzazione su applicazioni per il calcolo ad alte prestazioni. Esso viene definito più comunemente High Performance Computing (HPC) e verrà meglio illustrato nel paragrafo successivo.

Virtualizzazione e benchmarking per l'HPC

L'High Performance Computing (HPC) nasce come esigenza di operare calcoli di varie tipologie nell'ambito della ricerca scientifica. L'idea di base è quella di sfruttare la potenza dei processori di ultima generazione per eseguire questi calcoli in modo sistematico ed ottenere risposte umanamente impensabili in tempi ragionevolmente brevi.

Un primo passo verso l'HPC [22] è rappresentato dall'implementazione ed introduzione dei supercomputer, ossia delle architetture hardware in cui il processore viene pensato per essere parallelizzato e favorire così una maggiore velocità di calcolo. Vengono quindi usati da principio gli SMP (multiprocessori simmetrici) (fig. 2.10) in cui sono presenti più processori RISC (Reduced Instruction Set Computer) che possono accedere in modo parallelo alla memoria RAM. Il

problema principale sorse quando ci si rese conto che sarebbe stato impensabile aumentare la scalabilità fino ad un elevato numero di CPU. L'idea successiva fu

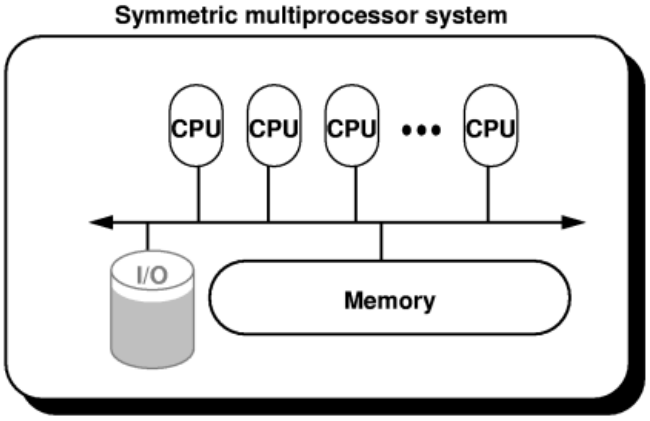


Figura 2.10: Multiprocessore simmetrico costituito da più processori RISC

quella di pensare che se non è possibile centralizzare il calcolo, bisognava trovare un modo per distribuirlo su tanti computer che eseguono una istruzione alla volta ma su un insieme di dati in modo simultaneo. Ecco quindi che vengono introdotte le piattaforme parallele a memoria distribuita (DMP) i cui computer si basano su un'architettura di tipo Single Instruction Multiple Data (SIMD). Sebbene questa morfologia portasse diversi benefici, il costo di acquisto rimaneva comunque non accessibile. Questo paradigma portò però le aziende clienti a creare delle configurazioni proprie, in cui tanti computer venivano collegati tra di loro tramite reti ad alta velocità con un sistema che garantisse un rapido trasporto dei dati tra le diverse memorie. Nonostante la velocità di trasporto non fosse così elevata, era possibile mettere a punto una struttura di questo tipo spendendo relativamente poco. In questo modo si incentivò la collaborazione di diverse workstation collegate tra loro a svolgere un unico lavoro e fu così che nacquero i Cluster of Workstation (COW), come alternativa alle soluzioni tradizionali.

Con l'aumento della frequenza di clock e delle dimensioni delle memorie, alcune applicazioni tipiche dell'HPC poterono essere eseguite su di un solo SMP ed i computer paralleli vennero resi accessibili ai più. Inoltre, era anche possibile eseguire i calcoli su di una CPU che accedeva ai dei dati in memoria appartenenti ad un altro SMP, tramite l'indirizzamento globale e dei collegamenti di rete appropriati. In definitiva, oggi come oggi, molte aziende sfruttano i benefici degli SMP e delle piattaforme parallele a memoria distribuita in modo combinato per

eseguire in modo efficiente e veloce i calcoli.

Risulta chiaro come in un contesto in cui la complessità del calcolo è elevata ed il tempo che si richiede per l'esecuzione debba essere minimo, la virtualizzazione abbia il potenziale per favorire l'HPC. Mentre nel settore enterprise la virtualizzazione è ormai affermata (si pensi alla server consolidation, alla riduzione di costi e dell'energia, alla fault-tolerance), l'adozione nel dominio dell'high performance computing rimane oggi carente.

Mentre la server consolidation è un fattore meno importante per l'HPC, rilevante è invece la specializzazione di molteplici OS che sono in completo controllo delle risorse hardware. L'hypervisor gestisce in modo sicuro le risorse hardware dell'host fisico, ma lascia l'allocazione delle specifiche risorse hardware al sistema operativo guest. Diversi OS possono quindi coesistere; essi possono essere specializzati per certi tipi di workload e vengono anche chiamati library OSs.

Quello che sta succedendo nell'ambito della ricerca in questi anni, è volto a capire come l'HPC può beneficiare della virtualizzazione al fine di migliorare le performance complessive che si vogliono raggiungere in fase di calcolo. Ecco quindi che entra in gioco il benchmark di sistemi virtualizzati, su cui vengono eseguite applicazioni HPC che vanno a sfruttare al massimo le risorse dei server virtualizzati per capire dove sia possibile diminuire l'overhead di virtualizzazione.

Da un punto di vista prettamente teorico, alcuni fattori chiave che possono essere citati a titolo di esempio sono [23]:

- in un ambiente virtualizzato una VM può essere impiegata per effettuare operazioni di monitoraggio del tipo: monitoraggio dello stato della memoria, interrupt e comunicazioni (call all'hypervisor) di un'altra VM per il debugging e l'analisi delle relative performance. Il codice per il monitoraggio viene eseguito al di fuori della VM che viene testata e per questo motivo non ne influenza il corretto funzionamento.

- L'hypervisor può fornire un cluster di VM virtuali distribuite su un numero più piccolo di host fisici (nodi), addirittura un singolo nodo e testare applicazioni HPC in modo realistico su scala più piccola, sfruttando un minor numero di risorse hardware.

- Il disaccoppiamento del software dall'hardware porta inoltre ad un guadagno di tempo sul restarting del sistema, che non deve essere più fatto a livello hardware dal BIOS ma viene fatto in maniera software, permettendo un'accensione più veloce.

- Di fondamentale importanza è sicuramente la possibilità che fornisce la virtualizzazione di migrare task all'occorenza e l'isolamento che permette l'esecuzione in modo sicuro delle operazioni.

Appurato che dei vantaggi in questo connubio HPC-virtualizzazione esistono, una serie di esperimenti per verificare le convenienze ed i limiti che questa strada comporta di recente sono stati effettuati. In alcuni casi [24], si è arrivati alla conclusione che non si può predire il comportamento di applicazioni HPC in ambito virtualizzato (nel caso in discussione su Xen), in quanto è impossibile, data la loro eterogeneità, stabilire a priori come l'hypervisor le gestirà. E' possibile che entrambe le applicazioni ottengano lo stesso punteggio in un benchmark, ma come questo è stato ottenuto può risultare diverso. Può capitare infatti che un'applicazione abbia impiegato più tempo nell'application code, mentre la seconda nel system code.

Altri rami di ricerca si occupano di identificare come la tecnologia di virtualizzazione può portare vantaggi qualitativi e quantitativi alle macchine ed alle applicazioni high-performance, verificandone la scalabilità verso sistemi futuri peta-core [25]. Si considerino per esempio gli hypervisor moderni che si basano per lo più su microkernel monolitici: questi kernel forniscono solo le funzionalità di base per gli OS come meccanismi per lo spazio di indirizzamento, astrazione per gestire l'allocazione della CPU e comunicazioni intra-processo. In questo contesto, in genere, tutti i core a disposizione eseguono la stessa funzionalità VMM. Se pensiamo però alle future architetture, questo può risultare un limite, in quanto viene introdotto un overhead significativo dovuto al costo di compiere operazioni sincronizzate e frequenti switch degli stati del processore, causando problemi alla cache ed al TLB. Qui viene proposta una soluzione definita "sidecore": il VMM viene strutturato in componenti multiple, ognuna delle quali è responsabile per una certa funzionalità; ogni componente viene inoltre assegnato ad un singolo core (fig. 2.11). A titolo d'esempio si consideri il meccanismo delle page table in Xen: in questo ambiente si ha una page table addizionale per ciascuna page table dei guest. Cambiamenti nelle pagetable dei guest causano cambiamenti in quella dell'hypervisor, per questo motivo un tipico page fault causa due VM-entry e due VM-exit. L'approccio sidecore riduce entrambi a uno: un core funge da sidecore, mentre l'altro funge per il "dominio 0" ed i guest.

Un'altra soluzione è quella relativa alla "self-virtualization" delle periferiche. In un odierno ambiente virtualizzato, l'I/O viene tipicamente effettuato da un driver domain. Ed è una VM che ha direttamente accesso alla risorsa fisica. Sarebbe opportuno, se possibile, cercare di "scaricare" parte della virtua-

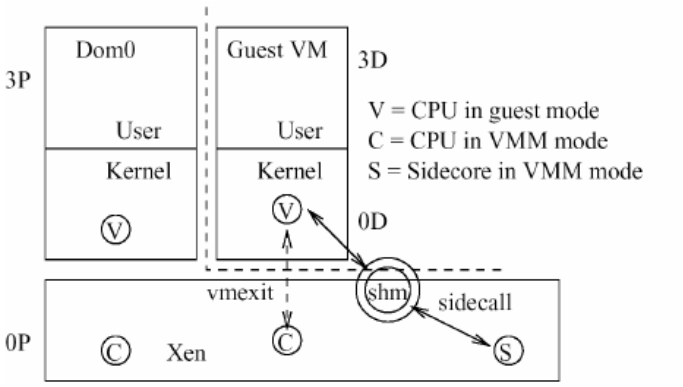


Figura 2.11: L'approccio sidecore in un ambiente virtualizzato

lizzazione sulla risorsa fisica stessa. Queste periferiche, che vengono chiamate "self-virtualized I/O devices", possono fornire un path di latenza bassa tra il guest e la risorsa fisica con un coinvolgimento minimo del VMM, fornendo un miglioramento delle performace e della scalabilità. Questa tipologia di hardware è gia disponbile, ma per sfruttare appieno le sue potenzialità, andrebbero modificati opportunamente l'hypervisor, l'OS ed i sistemi di interconnessione di I/O.

Vi sono poi altri rami della ricerca che si dedicano invece a confrontare le tecnologie di virtualizzazione andando a scalarle per vedere come l'overhead produce perdita nelle performance [26]. Lo scopo è quello di inviduarne le motivazioni e aiutare gli sviluppatori a migliorare i loro prodotti. Sempre in questo contesto, ma con una sfaccettatura diversa, ci sono altre ricerche che mettono a confronto le tecniche di virtualizzazione (full-virtualization, paravirtualization e hardware-assisted virtualization), andando ad effettuare dei benchmark prestabiliti su di esse [27].

2.6 Benchmark orientati alla virtualizzazione

Esistono inoltre benchmark di recente implementazione che vanno a valutare le performance di uno o più sever virtualizzati nella loro interezza. Con l'introduzione delle varie tecniche di virtualizzazione, l'interesse del cliente è risultato essere più rivolto alle performance complessive del server piuttosto che ai singoli componenti. In quest'area sono stati sviluppati essenzialmente due benchmark: VMMark, sviluppato da VMware e SPECvirtsc2010, implementato da SPEC. Essi in raltà si assomigliano molto e, in genere, presuppongono entrambi un client o più client fisici ed un server fisico su cui esercitare il carico di lavoro. Sulla macchina server, attraverso la virtualizzazione, vengono creati gruppi di sei macchine virtuali su cui vengono installate sei tipologie di server: webserver, database server, application server, file server, mail server e stand-by server. Questi sei guest insieme vanno a formare un tile. Il punteggio che viene assegnato nel documento è in funzione di quanti tile è stato possibile mandare in esecuzione, senza riduzione delle performance complessive dell'host. Ultimamente però è stata rilasciata una nuova versione di VMMark (VMmark 2.0) [28], la quale costruisce il tile in maniera diversa e va a valutare quali sono i tipi di carico con cui realmente deve interagire un datacenter odierno: vengono simulate infatti delle migrazioni di macchine virtuali, delle clonazioni e creazioni di macchine virtuali e dei bilanciamenti automatici del workload tipici dei datacenter.

2.6.1 SPECvirt-sc2010

SPECvirt-sc2010 viene progettato per rappresentare un metodo di misura standard per verificare le capacità di gestione della server consolidation di una piattaforma virtualizzata e per poter confrontare le performance complessive tra ambienti virtualizzati. Viene pensato per misurare le prestazioni hardware, software ed a livello applicativo in un ambiente virtualizzato. E' indirizzato a coloro che vogliono effettuare un benchmark dell'intera piattaforma e questi possono essere venditori di hardware, venditori di software per la virtualizzazione, ricercatori ed altri. Il prodotto è progettato per scalare un'ampia gamma di sistemi e comprende un ben definito insieme di workload tipico di applicazioni comuni.

Design del workload

In realtà SPECvirt-sc2010 si basa su un'ossatura ben consolidata implementata negli anni da SPEC. Infatti raggruppa un insieme di workload sviluppati in

passato per l'industria e costruisce un prodotto unico. Questi workload sono stati quindi adattati e modificati per essere idonei ad uno scenario tipico di una server consolidation.

In sostanza i workload utilizzati sono i seguenti:

- SPECweb2005: questo workload rappresenta un web server, un file server ed un infrastructure server.
- SPECjAppServer2004: esso rappresenta un application server ed un database server di back-end.
- SPECmail2008: rappresenta un mailserver.
- SPECpoll: aggiunto per SPECvirt-sc2010 e genera dei network pings ed attende i riscontri per testare le connettività.

Tramite un'attività di ricerca dei parametri, il test è stato implementato con una metodologia che permette al risultato di essere scalato sulle capacità del sistema. Il benchmark richiede una grande quantità di memoria RAM, un significativo quantitativo di storage e networking oltre ai processori presenti nel SUT. Anche il client, o più client, devono essere configurato correttamente per prevenire l'overloading.

Virtual machine e tile

Come sopra anticipato, è prevista la configurazione di 6 virtual machine in esecuzione su un host fisico. Esse sono:

- appserver
- dbserver (back-end per appserver)
- webserver
- infraserver (back-end per webserver)
- mailserver
- idleserver

L'insieme di queste 6 VM lato server va a formare un tile. L'idea di base è quella di replicare questo tile n volte, fino a quando un certo server è in grado di rispondere in maniera performante. E' anche possibile ridurre il workload di 1 tile per farlo agire ad una percentuale minore, in modo da sfruttare fino in fondo le caratteristiche del server. La configurazione è illustrata in figura 2.12.

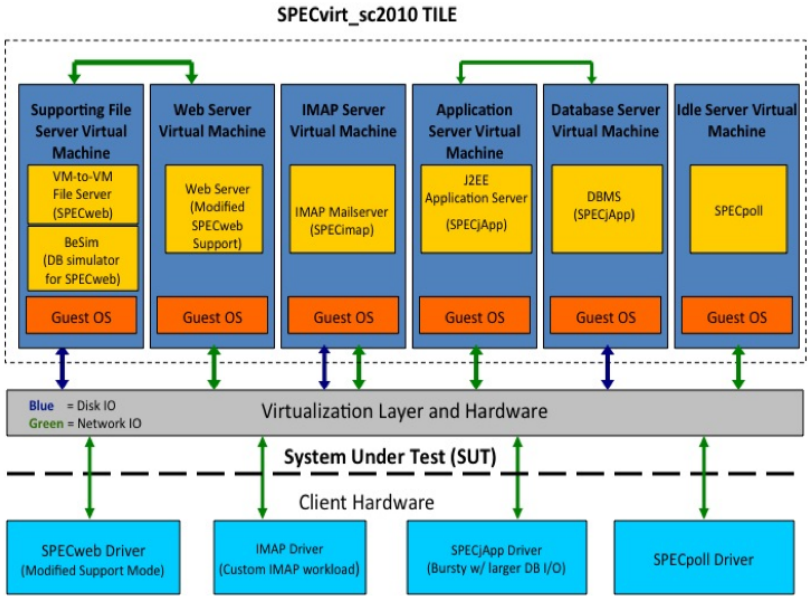


Figura 2.12: Configurazione dell'ambiente di test per SPECvirt-sc2010

Misure

Il punteggio ottenuto da un certo test si basa su una metrica complessiva che viene espressa come "SPECvirt-sc2010 $< Overall - Score >$ @ $< 6 \times NumberOfTiles >$ VMs" sul documento finale. Questo punteggio complessivo si basa essenzialmente su 3 misure derivanti dai workload generati:

- Webserver: richieste/secondo ad un certo numero di sessioni simultanee.
- Mailserver: la somma delle operazioni/secondo ad un certo numero di utenti.
- Application server: operazioni/secondo ad un certo injection rate, fattore di carico e busty curve
- Idleserver: msec/network ping (il quale però non fa parte del calcolo per il risultato finale)

Viene calcolato il punteggio complessivo prendendo ciascuna componente del workload per ogni tile e normalizzandolo sul suo massimo teorico per il fattore di carico predefinito. Ai tre throughput normalizzati per ciascun tile ottenuti viene applicata la media matematica in modo da creare una sottometrica riferita

ad ogni tile e queste vengono aggiunte per ottenere la metrica del punteggio complessivo. Un altro aspetto interessante è che nel documento è presente la Quality of Service che una certa macchina virtuale è in grado di offrire durante il test. Inoltre, la QoS deve essere superiore ad un livello minimo affichè il risultato del benchmark sia ritenuto valido.

2.6.2 Caso VP-LAB

Inizializzazione

Siccome l'obiettivo di questo lavoro di tesi è mettere a confronto diverse tecnologie di virtualizzazione, SPECvirt-sc2010 ben si offre per lo scopo che ci siamo proposti in partenza. Come tecnologia di testing primaria abbiamo utilizzato Citrix XenServer 5.5 su di un host presente al VP-LAB, le cui specifiche tecniche sono le seguenti:

- 1 CPU, 4 core Intel i7 920
- 12 GB di memoria
- 5 dischi da 470 GB in RAID 0
- 1 scheda di rete Gigabit

Per quanto riguarda la scelta dei guest OS e del software client, SPECvirt-sc2010 lascia libera scelta purchè siano rispettati certi criteri di base. In questa sede le 6 virtual machine sono state installate scegliendo come OS Red Hat 5.5 e configurate come segue:

- Appserver: Glassfish (2 Vcpu, 2 GB)
- Dbserver: Postgresql (2 Vcpu, 2 GB)
- Mailserver: Dovecot (2 Vcpu, 2 GB)
- Infraserver: NFS server (2 Vcpu, 2 GB)
- Webserver: Apache, Tomcat (2 Vcpu, 2 GB)
- Idleserver (1 Vcpu, 512 MB)

Esecuzione

La durata prevista per il test è di circa 3 ore e, in questo primo step, abbiamo mantenuto la configurazione di base provando ad effettuare il workload su di un solo tile. Lo schema di funzionamento del test per un tile è riportato in figura 2.13.

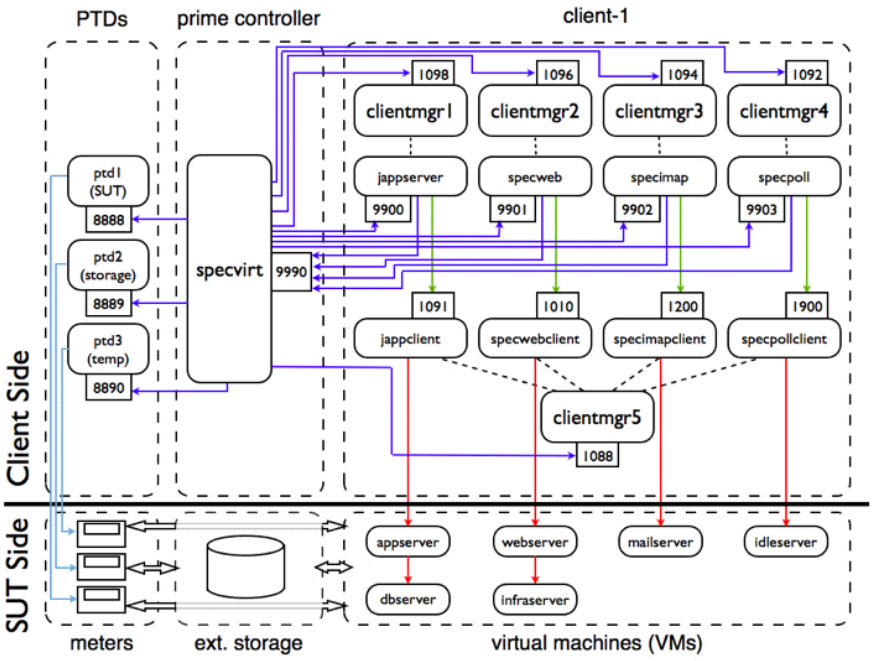


Figura 2.13: Schema di funzionamento del test su di un singolo host

Risultati

L'esecuzione del test, anche se predisposto e configurato correttamente, ha dato esito negativo. Dopo una serie di accertamenti ci siamo resi conto che il bottleneck si identificava essenzialmente nell'I/O prodotto dal webserver sul RAID 0 composto da 5 dischi durante la scrittura dei file sull'infraserver.

Visto che il benchmark, per come strutturato, permette di scalare il fattore di carico del tile desiderato, abbiamo deciso di effettuare una serie di test partendo da un carico molto basso per vedere fino a che punto il sistema in esame è in grado di sopportare il workload generato in maniera corretta. Infatti abbiamo predisposto i seguenti step:

1tile@0.1 : il carico è pari al 10% del carico totale per 1 tile.

1tile@0.2 : il carico è pari al 20% del carico totale per 1 tile.

1tile@0.3 : il carico è pari al 30% del carico totale per 1 tile.

1tile@0.4 : il carico è pari al 40% del carico totale per 1 tile.

1tile@0.5 : il carico è pari al 50% del carico totale per 1 tile.

I risultati[1] sono riportati in tabella 2.3. Risulta chiaro che il sistema in questione è in grado di sopportare al massimo il 40% del workload generato tramite SPECvirt-sc2010 per un tile a causa della grande quantità di I/O che viene effettuato su disco. In realtà, consultando i primi risultati riportati da SPEC, ci siamo resi conto che questo tipo di benchmark è progettato per testare server virtualizzati di veri e propri datacenter, infatti alcuni di essi riportano una configurazione dei dischi pari a 96 dischi × 73 GB in RAID 5.

	App server		**Web server**		**Mail server**	
	Throughput	QoS	Throughput	QoS	Throughput	QoS
1tile@1.0	X	X (0,68)	X	V	V	V
1tile@0.1	V	V	V	V	V	V
1tile@0.2	V	V	V	V	V	V
1tile@0.3	V	V	V	V	V	V
1tile@0.4	V	V	V	V	V	V
1tile@0.5	V	V	X	X (0.96)	V	V

Tabella 2.3: Risultati scalando 1 tile

[1]check "V" significa che i valori rispettano quelli specificati da SPEC, "X" significa che i valori attesi non sono rispettati.

Capitolo 3

Descrizione del benchmark

3.1 Configurazione ed ambiente di testing

I processi di performance testing sono stati svolti al Politecnico di Milano all'interno del laboratorio di valutazione delle prestazioni (VPLAB). In questo ambiente sono state predisposte due macchine fisiche (client e server) le cui caratteristiche tecniche hardware sono le seguenti:

- 1 CPU, 4 core Intel i7 920
- 12 GB di memoria
- 1 disco da 470 GB
- 1 scheda di rete Gigabit

Per quanto concerne il software abbiamo optato per un'installazione di CentOS 5.5 sul client fisico ed i guest OS, mentre per il server fisico sono stati installati gli hypervisor per le 4 tecnologie oggetto del confronto compiuto in questa sede: Xen 3.5.1, KVM incluso nel kernel linux 2.6.32, Hyper-V incluso in Microsoft Windows Server 2008 R2, ESXi 4.1. Su ciascuna virtual machine è stata installata un'istanza di Tomcat in modo da rendere ciascuna virtual machine un web server virtuale. Sul client, sul server fisico e sui guest OS è stato installato il JDK 1.6.0 update 22 per permettere l'esecuzione della JVM. Per il client è stata quindi implementata e installata un'applicazione java che esegue un workload sintetico, descritto nella sezione 3.4, sull'host ed effettua monitoring riportando dati sensibili di interesse. La relativa configurazione è illustrata in figura 3.1.

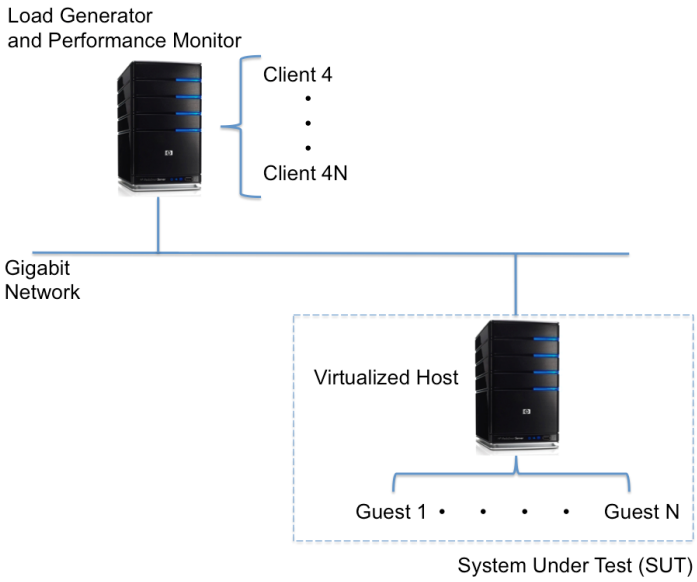


Figura 3.1: Configurazione dell'ambiente di testing

3.2 Rilevamento del throughput massimo

Sostanzialmente ci sono 2 entità che agiscono per il rilevamento del throughput massimo durante la fase del test. Viene dapprima inizializzato il numero di "clientWorker" che coincide col numero di client previsti (4) per il numero di target dell'esperimento (i quali valori variano a seconda del test da 1 a 16). Ogni clientWorker si occupa di eseguire delle richieste al server virtuale designato e ne attende la relativa risposta, in seguito si occupa di registrare la risposta sulla seconda entità.

La seconda entità è rappresentata dal "metricsWorker", che si mette in attesa fino alla fine del periodo di campionamento, si sincronizza con il clientWorker, conta le TPS del periodo con una cadenza di circa 5 secondi e registra il campione relativo a tale periodo; infine riazzera il conteggio.

3.3 Rilevamento del response time

Per quanto riguarda la seconda parte, a fronte di un carico di richieste fatte sempre dal client ai server virtuali, vengono misurati i tempi di risposta tramite un agent che è in esecuzione sul server fisico stesso e che in sostanza va a monitorare e collezionare alcune metriche come l'utilizzo della CPU da parte delle

macchine virtuali e del "dominio 0". L'HypervisorAgent comunica con il client, generatore di carico, tramite una connessione RMI[1].

Per ogni macchina virtuale in esecuzione viene rilavato l'utilizzo della CPU attraverso:

$$U_{CPU} = \frac{t_2 - t_1}{\tau_2 - \tau_1}$$

dove t_1 e t_2 rappresentano i timestamp dell'orologio rispettivamente al momento di start e stop dell'attività di monitoraggio, e τ_1 e τ_2 il tempo CPU consumato dall'accensione al momento di start e di stop.

Una volta ottenuto l'utilizzo - espresso in percentuale - è possibile andare a campionare con un intervallo deciso arbitrariamente ma in modo che sia sufficientemente significativo. Il response time relativo a ciascuna macchina viene rilevato semplicemente attraverso il load generator, il quale misura il tempo che intercorre tra una richiesta HTTP e l'arrivo della relativa risposta per ogni singola macchina virtuale soggetta al test.

3.4 Descrizione del workload sintetico

Quello che ci si è proposti di fare inizialmente è stato stabilire come generare il workload in modo che fosse di tipo sintetico, ossia che stressasse uno o più componenti in modo particolare, per studiarne le prestazioni massime ottenibili in termini di throughput X, tempi di risposta R e utilizzo U. In questo lavoro di tesi è stato scelto di stressare principalmente due componenti dell'architettura del calcolatore, ossia CPU e memoria.

Per studiare quindi gli effetti che hyperthreading e virtualizzazione comportano in questo contesto, il workload deve essere strutturato in modo che:

- la complessità del tempo che ci impiega una richiesta ad essere processata sia *O(n)*, ossia l'andamento al crescere delle richieste sia lineare;
- il workload deve stressare le gerarchie di memoria;
- il workload deve essere parzialmente memory-bound e CPU-bound;
- deve essere possibile fare tuning degli accessi in memoria.

[1] La connessione Remote Method Invocation è una tecnologia che consente a processi java distribuiti di comunicare attraverso una rete. In sostanza rende possibile la chiamata remota di opportuni metodi java.

L'implementazione del workload prevedere essenzialmente due fasi: inizializzazione ed esecuzione.

Inizializzazione

1. Vengono generati **d** vettori che rappresentano il working set.
2. La loro lunghezza è uniformemente distribuita tra **cMin** e **cMax**.
3. Viene generato un filtro **f** di lunghezza **h**.

Esecuzione

1. Viene selezionato un vettore random **v** tra i **d** vettori generati.
2. Si allocano due vettori **a** e **b** di lunghezza **n**.
3. Si riempie **a** con le somme di **p** stride casuali di lunghezza **w** provenienti da **v**.
4. Si effettua la convoluzione del filtro **f** con **a** e si salva il risultato in **b**.
5. Viene infine ritornata la somma degli elementi di **b**.

E' possibile vedere quello che accade durante la fase di esecuzione in figura 3.2. Si noti che gli obiettivi che ci siamo preposti sono rispettati, ossia il riempimento di **a** è *O(n)* in quanto *O(***p**×**w**×**n***)* ed il salvataggio in **b** della convoluzione tra **a** ed **f** è *O(***n**×**h***)*. In riferimento all'utilizzo della cache, se consideriamo un floating point a doppia precisione che occupa 8 bytes e vogliamo sfruttare tutta la cache a disposizione, dobbiamo necessariamente ottenere che la lunghezza dei vettori ecceda la dimensione della cache di ultimo livello (LLC):

$$d \times 8 \times \left(cMin + \frac{(cMax - cMin)}{2} \right) \gg LLC_{size}$$

Per quanto riguarda invece i vettori sorgenti e destinazione, devono risiedere in una cache privata (L1 o L2), quindi si ha che:

$$n \times 2 \times 8 \ll LX_{size}$$

Infine il filtro deve risiedere nella cache di primo livello L1:

$$h \times 8 \ll L1D_{size}$$

La probabilità di una hit nella cache di ultimo livello (L3) durante le operazioni di caricamento e somma di **p**×**w**×**n** elementi, dipende dalla dimensione del

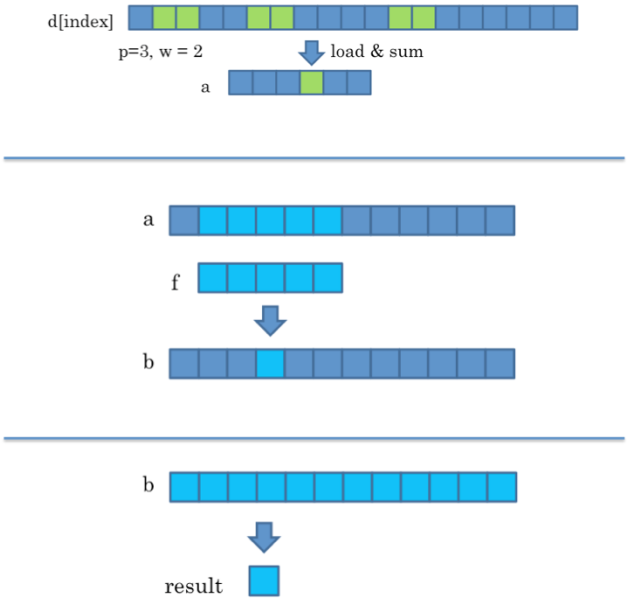


Figura 3.2: Workload sintetico: fase di esecuzione

dataset. Per quanto concerne l'intensità aritmetica delle operazioni, questa può essere stabilita tramite il filtro **h**, il quale però deve sempre essere $\ll$ della dimensione delle cache L1/L2. In riferimento agli esperimenti che sono stati effettuati per questo lavoro di tesi abbiamo scelto di utilizzare i seguenti valori per i parametri sopra citati:

d = 10;

cMin = 1048576;

cMax = 1048576;

p = 40;

w = 4;

n = 6500;

h = 1536.

In base alla scelta di questi parametri si ha che:

$$10 \times 8 \times \left(1048576 + \frac{(1048576 - 1048576)}{2}\right) = 80MB \gg 8MB,$$

dove 8 MB rappresenta la dimensione della cache LLC.

Tali scelte mostrano che l'andamento che ci si aspetta è rispettato, ossia all'aumentare della dimensione **n** dei vettori **a** e **b**, il tempo aumenta in modo lineare eccezion fatta per alcuni outlier (figura 3.3). Per quanto riguarda la

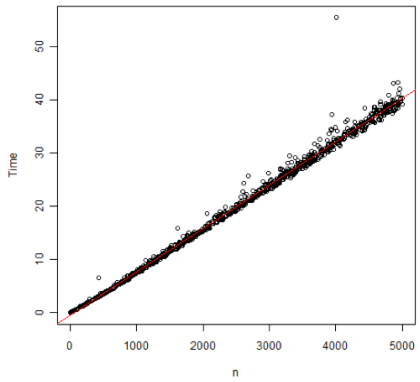


Figura 3.3: Il tempo di esecuzione è O(n) rispetto al numero di richieste

scelta del filtro **h** possiamo notare in figura 3.4 che al crescere della dimensione del filtro, il tempo cresce linearmente, ossia l'andamento è *O(h)*. Durante la generazione del workload, n è ottenuto estraendo un naturale dalla distribuzione esponenziale con media 6500 e arrotondando all'intero più vicino. Essendo il tempo di servizio *O(n)*, la sua distibuzione sarà approssimativamente esponenziale. Questo è stato verificato con un test di Kolmogorov-Smirnov. In questo modo vengono rispettate le ipotesi dei più usati modelli a reti di code. Nelle fi-

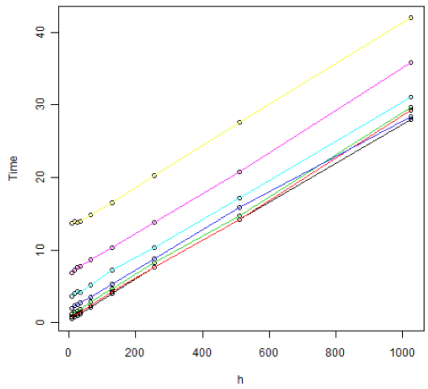


Figura 3.4: Il tempo di esecuzione è O(h) rispetto alla dimensione del filtro h

gure 3.5 e 3.6 possiamo infine notare l'impatto del numero di **p** stride sul tempo di esecuzione: nel primo caso si vede che esso aumenta linearmente al crescere di **p**; mentre nel secondo caso si evince che maggiore è la lunghezza della stride **w**, minore sarà il tempo di esecuzione che diminuisce asintoticamente. Questo effetto è dovuto all'ampiezza della cache line e al comportamento del prefetcher.

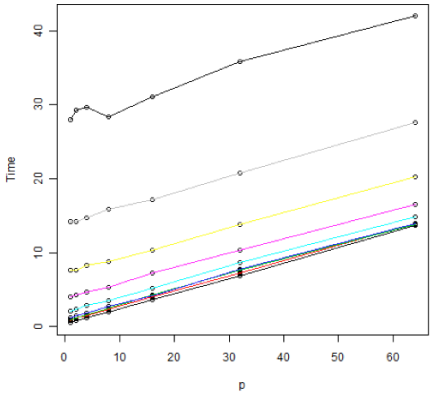


Figura 3.5: Il tempo di esecuzione è O(p) rispetto al numero di stride p

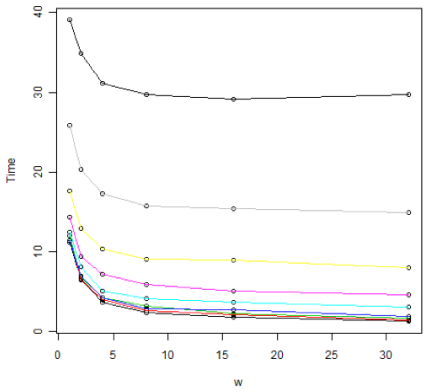


Figura 3.6: Il tempo di esecuzione diminuisce all'aumentare della lunghezza della stride w

Capitolo 4

Throughput massimo in sistemi virtualizzati

L'idea di base per il primo lavoro di questa tesi è stata quella di verificare l'impatto della virtualizzazione in termini di overhead introdotto dalle varie tecnologie messe a confronto. Per fare questo è stato necessario misurare il throughput X derivante da ciascuna virtual machine in gioco. Lo scopo del workload sintetico, come vedremo meglio nei paragrafi successivi, era quello di stressare la CPU al punto che, raggiunto l'utilizzo totale dei 4 core a disposizione, si assistesse ad un calo in termini di prestazioni e quindi di transazioni al secondo.

In un contesto di questo tipo, si deve ricorrere ad un modello chiuso per le reti di code [29] in cui si hanno le risorse sotto test (nel nostro caso CPU e memoria) che fungono da nodi di servizio per la rete in questione. Nel modello chiuso si può impostare il think time Z equivalente a 0, per il fatto che, se si vuole stressare la CPU, delle richieste continue ai web server virtuali devono essere eseguite costantemente ed in modo ripetitivo senza tempi di attesa sui vari nodi. Il modello che è stato utilizzato è illustrato in figura 4.1. Si noti che nel modello vengono rappresentati solo queste due componenti in quanto l'applicazione generatrice di carico è stata implementata per non stressare nè coinvolgere in maniera significativa le due componenti rimanenti: disco e rete.

Al fine di sfruttare appieno le potenzialità della macchina, i test sono stati eseguiti dapprima con e senza *hyperthreading* [30] e sono state messe a confronto le quattro tecnologie in oggetto.

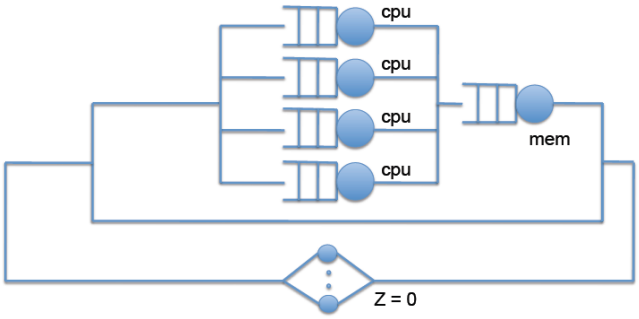


Figura 4.1: Modello chiuso per la rete di code

4.1 Formulazione del problema

- Date K virtual machine con V virtual cpu, si misura il throughput medio per ciascuna virtual machine. Tale esperimento è da ripetere e confrontare con e senza hyperthreading per diversi valori di V. Intuitivamente ci si aspetta che il throughput totale raggiunga il picco nel momento in cui $V \times K$ è uguale al numero di core fisici (o, se attivato l'hyperthreding, virtuali). E' interessante notare come aumentando K si assiste al decrescere del thrpughput totale (ossia quanto overhead viene introdotto).

- Per diversi valori di V, rappresentati da diverse linee nello stesso grafico, osservare il throughput totale della macchina (ossia la somma dei throughput medi) in funzione delle virtual cpu totali ($V \times K$). In questo modo è possibile, per esempio, notare la differenza tra impostare 16 guest con 1 virtual cpu, piuttosto che 8 guest con 2 virtual cpu.

Formulando in questo modo il problema si evince subito che le uniche variabili indipendenti in gioco sono K e V, mentre le variabili dipendenti risultano essere X e l'utilizzo (U) dei core e della memoria. Occorre però effattuare una scelta di altri parametri per lo svolgimento dei test, dato il contesto nel quale devono essere eseguiti. Questi parametri sono essenzialmente i seguenti: memoria, tempo di esecuzione, numero di client per virtual machine e heap. Per la relativa valutazione è stata usata come tecnologia di prova quella proposta da Xen.

La scelta di questi parametri è dettata dal fatto che l'architettura in questione pone certi limiti. L'upper bound per quanto riguarda la memoria è 12 GB da ripartire equamente tra le diverse virtual machine a seconda dei test che si

vogliono effettuare. Tenendo conto del fatto che l'hypervisor, per essere in esecuzione ha bisogno di almeno 1 GB, abbiamo deciso di impostare per il sistema operativo guest almeno 680 MB di memoria nel caso limite; così facendo il numero massimo di virtual machine che possiamo utilizzare contemporaneamente è 16.

$$680\text{MB} \times 16 = 10880\text{GB} \sim 11\text{GB}$$

La dimensione dell'heap è stata impostata a 296 MB, riservando i rimanenti 384 MB al sistema operativo. Dopo una prima serie di prove per verificare il funzionamento dell'ambiente di test è emerso che, quando una singola virtual machine ha a disposizione molte risorse CPU, il garbage collector non riesce a sostenere il throughput dell'applicazione e si verifica pertanto un esaurimento dell'heap. Nei test con 2 vcpu e 4 vcpu è stata quindi aumentata la quantità di memoria RAM assegnata a ciascuna virtual machine e la dimensione dell'heap. Da ultimo bisognava stabilire per quanto tempo monitorare il throughput delle macchine virtuali e quanti client (thread paralleli sul client fisico) assegnare ad ogni virtual machine. Per fare ciò abbiamo eseguito una serie di esperimenti che prevedevano le seguenti configurazioni:

Procedimento 1 Scelta dei parametri: secondsPerRun e numClients

- Siano date 16 VM con 16 istanze di Tomcat installate e funzionanti
- I possibili valori di secondsPerRun sono: 600 s, 1200 s, 1800 s, 3600 s
- I possibili valori di numClients sono = 1, 2, 4, 8
- Si eseguano $2^4 = 16$ run e si confrontino gli intervalli di confidenza
- Laddove l'intervallo di confidenza risulterà minimo, si scelga il numero di client

Il tempo di esecuzione del test secondsPerRun è stato impostato a 1800 s, in quanto è un periodo sufficientemente lungo per prelevare campioni di throughput rilevanti con una cadenza di circa 5 secondi. La scelta per quanto riguarda numClients = 4 è stata fatta in base al valore minimo tra gli intervalli di confidenza, come è possibile notare in figura 4.2.

Fatta questa scelta, il passo successivo è stato quello di effettuare il test effettivo argomento di discussione. Da qui in poi i test sotto descritti sono stati effettuati con e senza hyperthreading per valutare quale delle due configurazioni potesse risultare più significativa.

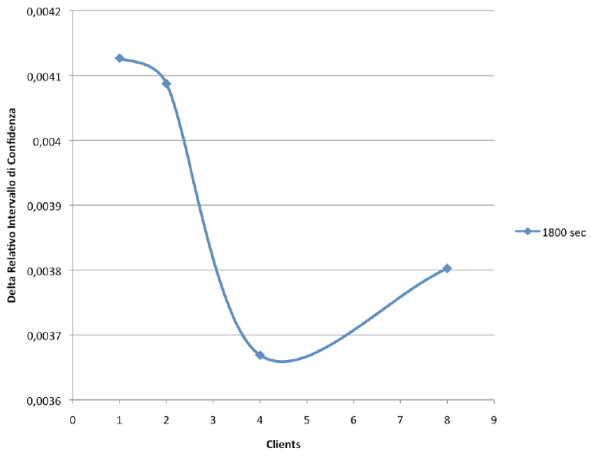


Figura 4.2: Scelta del numero di client per virtual machine

Procedimento 2 Test A: Rilevamento throughput massimo senza hyperthreading

- Test 1: 16 K, 1V, Heap = 296, Mem = 680. 16 run: da 1 a 16 aggiungendo 1 K ad ogni run
- Test 2: 8 K, 2V, Heap = 976, Mem = 1360. 8 run: da 1 a 8 aggiungendo 1 K ad ogni run
- Test 3: 4 K, 4V, Heap = 2336, Mem = 2720. 4 run: da 1 a 4 aggiungendo 1 K ad ogni run

Si noti come l'heap varia in funzione della memoria (lasciando sempre spazio sufficiente per il sistema operativo) e la memoria in funzione del numero di virtual machine e virtual cpu. Questo avviene in quanto il dimensionamento ottimale della memoria è dipendente dalla capacità del throughput. Chiaramente il Test B (con hyperthreading) prevede un sotto-test in più per il fatto che, siccome si sfruttano otto core logici, si è in grado di mandare in esecuzione 2 virtual machine con 8 virtual cpu ciascuna. I risultati mostrano che, utilizzando l'hyperthreading si ottengono sempre migliori performance rispetto all'utilizzo di soli 4 core fisici. In generale si raggiungono dei picchi di throughput X quando si giunge prima a sfruttare i 4 core fisici e poi gli 8 logici, punto oltre il quale il throughput comincia a decrescere. Questo è dovuto al fatto che la CPU è satura, ovvero il suo utilizzo U è vicino al 100%, e non riesce più a gestire le richieste effettuate dai client ai web server. Si assiste quindi ad un calo sensibile delle performance dovuto all'introduzione della virtualizzazione.

Per spiegare questo è stato effettuato un esperimento simile al precedente sulla macchina fisica, installando direttamente sull'host sedici istanze di Tomcat. Chiaramente quest'ultimo non è completamente comparabile al precendente, in quanto le sedici istanze avranno subito a disposizione i 4 o gli 8 core forniti dall'architettura. Nonostante questo, lo svolgimento in questo contesto fornisce un'idea anzitutto del livello massimo di throughput che la macchina può sopportare, inoltre mette in luce la diversità di andamento rispetto al caso virtualizzato. E' evidente che l'andamento decrescente che si osserva nel caso virtualizzato dipende dall'overhead introdotto dalla virtualizzazione stessa (rif. 4.6).

Procedimento 3 Test B: Rilevamento throughput massimo con hyperthreading

- Test 1: 16 K, 1V, Heap = 296, Mem = 680. 16 run: da 1 a 16 aggiungendo 1 K ad ogni run
- Test 2: 8 K, 2V, Heap = 976, Mem = 1360. 8 run: da 1 a 8 aggiungendo 1 K ad ogni run
- Test 3: 4 K, 4V, Heap = 2336, Mem = 2720. 4 run: da 1 a 4 aggiungendo 1 K ad ogni run
- Test 4: 2 K, 8V, Heap = 5056, Mem = 5440. 2 run: da 1 a 2 aggiungendo 1 K ad ogni run

I Test A e il Test B, con hyperthreading attivato, sono stati effettuati sulle

4 tecnologie che si vogliono confrontare in questo lavoro di tesi. I risultati sono mostrati nelle figure 4.3, 4.4, 4.5, 4.6, 4.7, 4.8. In questi grafici si evince come venga introdotto overhead di virtualizzazione da parte delle quattro tecnologie in questione: per 16 virtual machine la tecnologia che perde maggiormente in termini di throughput è KVM, mentre la migliore risulta essere VMware; per 8 virtual machine si ha che, dopo il picco atteso, relativo a 8 vcpu VMware non mostra perdita mentre Xen, HyperV e KVM mostrano un decremento significativo; infine per 4 virtual machine VMware e HyperV non mostrano perdita, mentre Xen e KVM perdono. Le perdite sono riassunte in tabella 4.1.

4.2 Risultati

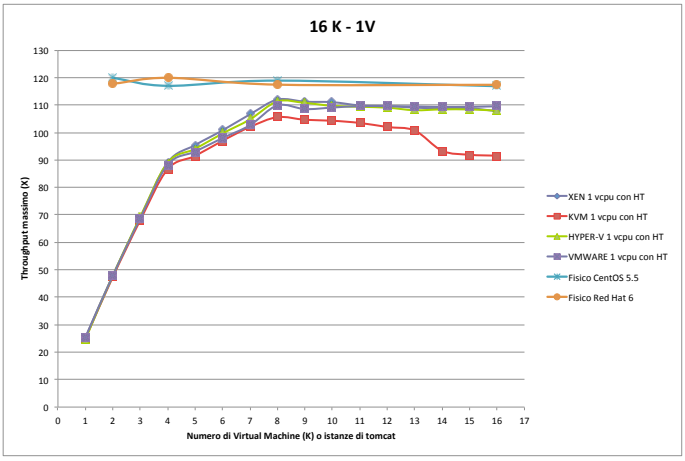


Figura 4.3: Throughput massimo con 16 virtual machine e 1 vcpu

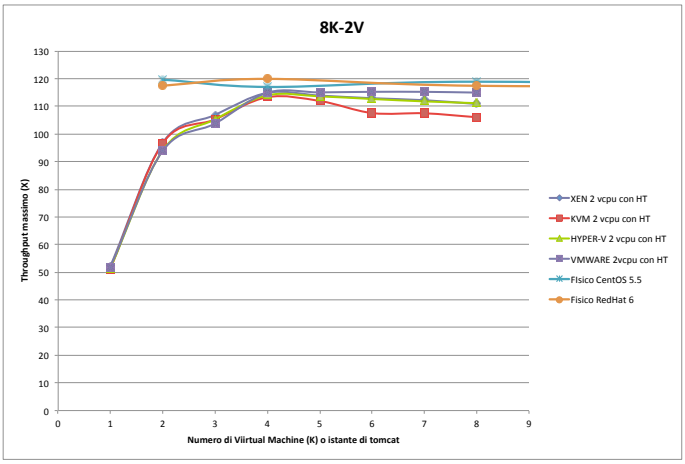


Figura 4.4: Throughput massimo con 8 virtual machine e 2 vcpu

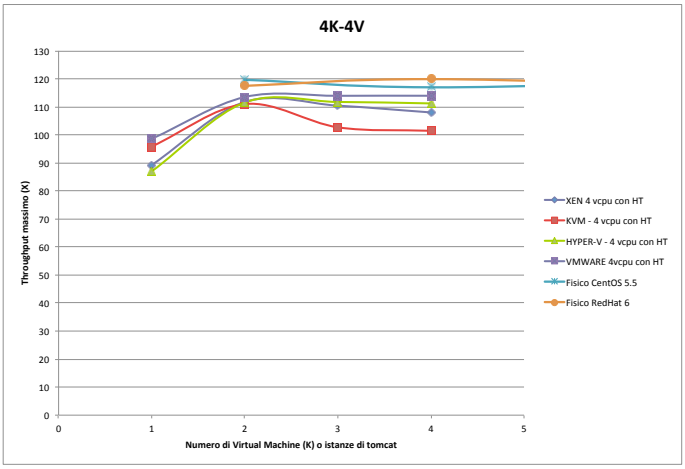


Figura 4.5: Throughput massimo con 4 virtual machine e 4 vcpu

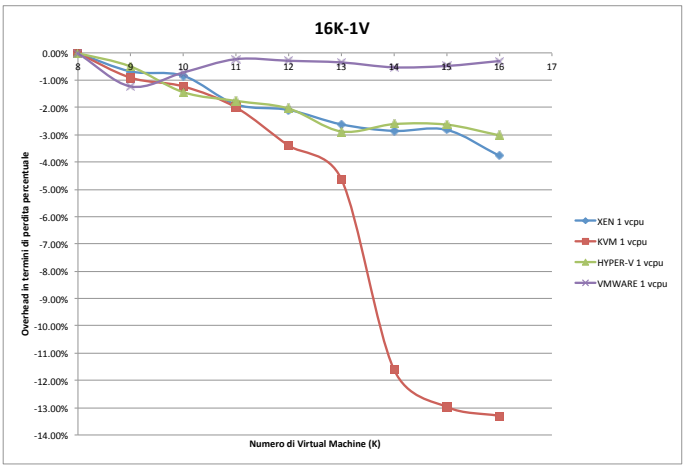


Figura 4.6: Perdita percentuale con 16 virtual machine e 1 vcpu

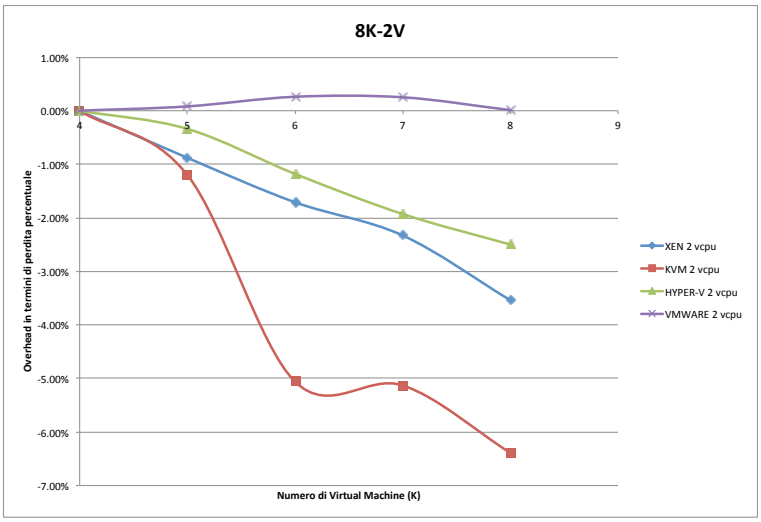


Figura 4.7: Perdita percentuale con 8 virtual machine e 2 vcpu

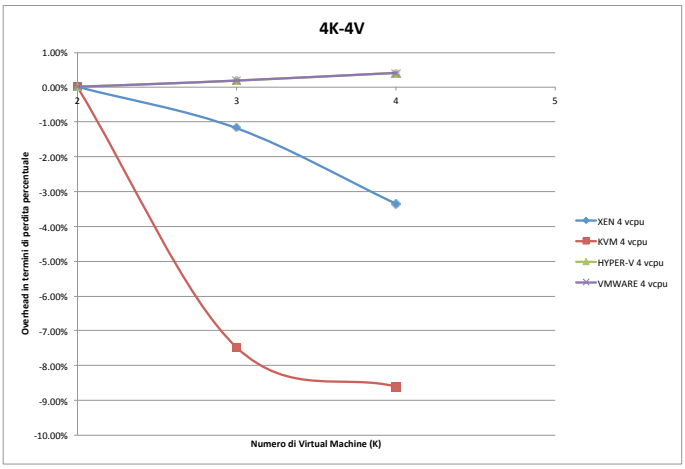


Figura 4.8: Perdita percentuale con 4 virtual machine e 4 vcpu

	Xen	KVM	Hyper-V	VMware
16 K - 1 V	-3,77%	-13,30%	-3,02%	-1,23%
8 K - 2 V	-3,54%	-6,39%	-2,50%	-
4 K - 4 V	-3,35%	-8,60%	-	-

Tabella 4.1: Perdite in percentuale nei casi di test per le 4 tecnologie

4.3 Conclusioni

Le conclusioni che si possono trarre dopo questa serie di esperimenti sono essenzialmente le seguenti:

- per $K \leq$ numero di core fisici, le quattro tecnologie sono HT-aware;
- le prestazioni migliori si ottengono per 2 $V \ \forall \ K$, per ogni tecnologia;
- la virtualizzazione introduce OH causando una perdita per Xen, KVM, HyperV e VMware, ma VMware mostra una perdita solo per 16 K e 1 V;
- lo scheduler di VMware è il più efficiente in termini di prestazioni ottenute soprattutto per un elevato numero di vcpu.

Durante questi test ci siamo chiesti come il *ballooning* [31] potesse migliorare le prestazioni o comunque modificarle. Purtroppo alcune delle versioni degli hypervisor installate non gestiscono la memoria in maniera dinamica, cioè l'hypervisor non gestisce il ballooning in modo automatico e, non potendo poi confrontare i risultati, abbiamo rimandato ad una verifica futura, da effettuare con hypervisor più aggiornati che supportino questa funzionalità.

Capitolo 5

Tempi di risposta in sistemi virtualizzati

Per la seconda parte di questo elaborato, ci siamo soffermati sulla valutazione dei tempi di risposta in sistemi virtualizzati. A differenza del caso precedente, se ciò che si vuole monitorare è il tempo che impiega un certo sistema a rispondere ad una determinata sollecitazione, è opportuno utilizzare un modello aperto per le reti di code come quello illustrato in figura 5.1.

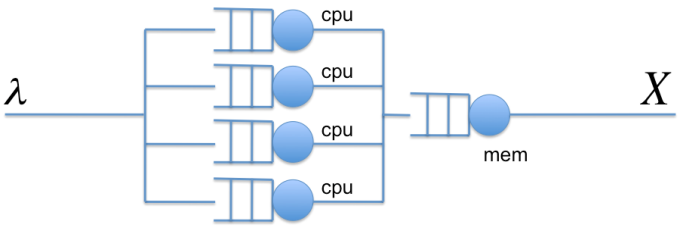


Figura 5.1: Modello aperto per la rete di code

Anche in questo caso in figura 5.1 possiamo notare che vengono rappresentati solo i componenti del sistema oggetto della discussione. Mentre prima avevamo un upper bound sul numero di richieste nella rete di code (closed model), in questo caso il sistema può ricevere un numero illimitato di richieste da processare (almeno a priori). Intuitivamente ci si aspetta che quando il sistema raggiungerà il massimo di richieste sopportabile - cioè le code dei vari centri di servizio (nel nostro caso CPU e memoria principale) saranno piene - il sistema comincerà a rispondere più lentamente andando ad incrementare il tempo di risposta dei vari centri e complessivo. E' in questo contesto che ci interessa appunto andare

a valutare come il sistema virtualizzato risponderà ad un carico di richieste che presenta un interarrival time λ con una distribuzione di tipo esponenziale[1], in cui le richieste generate in modo casuale in un certo intervallo di tempo non dipendono da quelle generate in precedenza.

5.1 Formulazione del problema

Come è noto, per la *legge del bilanciamento del flusso* [29], in un sistema open model si ha che λ equivale al throughput X della rete. Quindi sapendo l'interarrival time - parametro noto richiesto per la specifica del modello - siamo sempre in grado di conoscere il throughput di un modello aperto. Inoltre il *service demand* S si ottiene come:

$$S = \frac{1}{X_{max}}.$$

Nella parte precedente di questo lavoro di tesi ci siamo focalizzati sulla misurazione del throughput massimo della rete, quindi siamo in grado di calcolare S il quale rappresenta quanti job al massimo per secondo il nostro sistema è in grado di evacuare. Se vogliamo che il nostro sistema mantenga questo service demand il periodo di campionamento P a fronte di un certo carico di richieste, A dovrà essere necessariamente espresso nel seguente modo:

$$P = \frac{A}{\lambda},$$

dove $\lambda = X$ e quindi,

$$\lambda = \frac{U}{S}.$$

U rappresenta l'utilizzo totale dell'host e, siccome nel nostro caso abbiamo deciso di impiegare al massimo $k = 16$ virtual machine, possiamo dire che in sintesi:

$$P = \frac{A}{\frac{U}{k} \times \frac{1}{S}}.$$

Tale periodo chiaramente varierà a seconda del numero di virtual machine in gioco e di virtual cpu per virtual machine assegnate; inoltre il service demand S varierà a seconda che si utilizzi l'hyperthreading o meno. Per quanto riguarda

[1] Esistono diversi tipi di distribuzioni con le quali si può rappresentare λ, ad esempio distribuzione uniforme o Pareto

la stima del service demand nel caso in cui l'hyperthreading sia attivo, è stato necessario sovrastimare quello reale. La spiegazione per questa scelta si evince in figura 5.2.

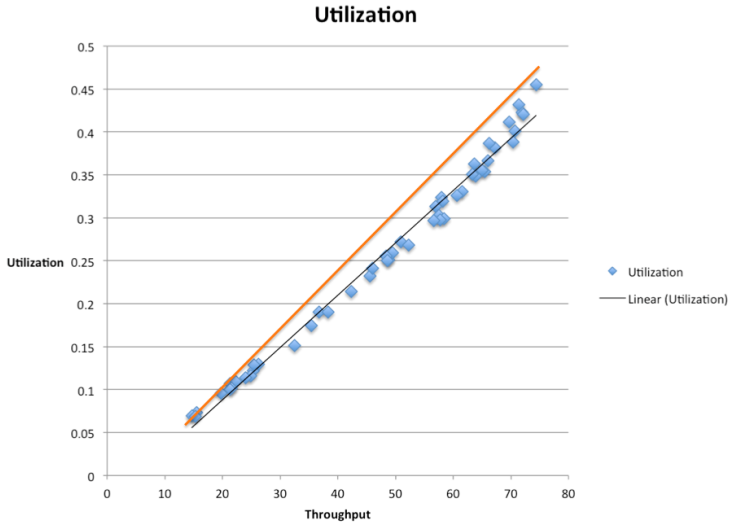


Figura 5.2: Andamento non lineare dell'utilizzo in funzione del throughput nel caso con hyperthreading

La teoria afferma che al crescere del throughput l'utilizzo della CPU dovrebbe aumentare in maniera lineare; in realtà questo non avviene nel caso con hyperthreading, il quale introduce una sorta di concavità alla funzione che rende necessaria una sovrastima lineare di tale andamento. Questo comportamento è dettato dal fatto che il sistema "crede" di avere 8 core fisici, quando in realtà ne ha 4 fisici con hyperthreading che corrispondono a 8 logici, ma che non vengono sfruttati come fossero fisici.

Come esperimenti per la valutazione dei tempi di risposta tra le tecnologie abbiamo pensato di dividere i test in 2 categorie:

Categoria 1 share e utilizzi equamente distribuiti tra le virtual machine;

Categoria 2 share uguali e utilizzi sbilanciati tra le virtual machine di un fattore F.

In entrambe le categorie il processo di misurazione dei tempi di risposta è stato il medesimo. Nella prima categoria (procedimento 4), i test sono stati eseguiti con ripartizione dell'utilizzo totale bilanciato fra i vari guest e sono stati ripetuti per ogni tecnologia, con e senza hyperthreading.

Procedimento 4 Test C: Valutazione dei tempi di risposta

- Si misuri il throughput massimo X_{max} dell'host in assenza di virtualizzazione
- Si calcoli il service demand S dell'host
- Si determini il periodo P al variare di K e V
- Si effettuino i test per U=10%, 30%, 50%, 70% da ripartire equamente o in modo sbilanciato tra i guest
- Si traccino i grafici relativi a response time R in funzione dell'utilizzo U

In questo caso abbiamo ottenuto le seguenti misure, le quali ovviamente valgono anche nel caso sbilanciato:

$$X_{max-HT} = 120.505534\ TPS,$$

$$X_{max-noHT} = 100.684891\ TPS,$$

$$S_{HT} = \frac{1}{120.505534} = 0.008298374\ secondi\ per\ transazione,$$

$$S_{noHT} = \frac{1}{100.684891} = 0.009931977\ secondi\ per\ transazione,$$

$$A = 2000\ richieste,$$

ed abbiamo determinato i periodi di campionamento come riportati in tabella 5.1. In tabella 5.3 vengono invece riassunti gli utilizzi ripartiti secondo il numero di virtual machine per test.

L'esperimento che abbiamo sviluppato successivamente vede l'introduzione di un fattore di sbilanciamento F dell'utilizzo delle singole macchine virtuali. In questo modo stiamo quindi sbilanciando l'utilizzo delle virtual machine mantenendo però sempre un utilizzo totale dell'host invariato pari a U. In concreto, quello che viene tolto ai guest che si vogliono utilizzare di meno viene aggiunto a quelli che si vogliono utilizzare di più.

Per quanto riguarda i test effettuati abbiamo scelto di utilizzare un fattore $F = 0.75$, ed in questo modo abbiamo ottenuto 3 classi di guest che si differenziano per l'utilizzo assegnato loro:

	hyperthreading attivo			
Utilizzo	0,1	0,3	0,5	0,7
16 K - 1V	$P = \frac{2000}{\frac{0.1}{16} \times \frac{1}{0.008298374}} = 2656s$	$P = 886s$	$P = 532s$	$P = 380s$
8 K - 2 V	$P = 1328s$	$P = 443s$	$P = 266s$	$P = 190s$
4 K - 4 V	$P = 664s$	$P = 222s$	$P = 133s$	$P = 95s$
	hyperthreading disattivato			
Utilizzo	0,1	0,3	0,5	0,7
16 K - 1V	$P = \frac{2000}{\frac{0.1}{16} \times \frac{1}{0.009931977}} = 3179s$	$P = 1060s$	$P = 636s$	$P = 455s$
8 K - 2 V	$P = 1590s$	$P = 530s$	$P = 318s$	$P = 228s$
4 K - 4 V	$P = 795s$	$P = 2265s$	$P = 159s$	$P = 114s$

Tabella 5.1: Calcolo dei periodi di campionamento nel caso bilanciato

- $U_{low} = \frac{U}{k} \times (1 - F)$;
- $U_{high} = \frac{U}{k} \times (1 + F)$;
- $U_{medium} = \frac{U}{k}$, in questo caso $F = 0$ (ossia l'utilizzo non viene sbilanciato).

	hyperthreading attivo			
Utilizzo	0,1	0,3	0,5	0,7
16 K - 1V	$P = \frac{2000}{[\frac{0.1}{16} \times (1-0.75)] \times \frac{1}{0.008298374}} = 10622s$	$P = 3541s$	$P = 2125s$	$P = 1518s$
8 K - 2 V	$P = 5311s$	$P = 1771s$	$P = 1063s$	$P = 759s$
4 K - 4 V	$P = 2656s$	$P = 886s$	$P = 532s$	$P = 380s$
	hyperthreading disattivato			
Utilizzo	0,1	0,3	0,5	0,7
16 K - 1V	$P = \frac{2000}{[\frac{0.1}{16} \times (1-0.75)] \times \frac{1}{0.009931977}} = 12713s$	$P = 4238s$	$P = 2543s$	$P = 1817s$
8 K - 2 V	$P = 6357s$	$P = 2119s$	$P = 1272s$	$P = 909s$
4 K - 4 V	$P = 3179s$	$P = 1060s$	$P = 636s$	$P = 455s$

Tabella 5.2: Calcolo dei periodi di campionamento nel caso sbilanciato

Applicando questo fattore di sbilanciamento, i periodi vengono chiaramente ricalcolati nel seguente modo:

$$P = \frac{A}{[\frac{U}{k} \times (1 - F)] \times \frac{1}{S}},$$

tenendo conto del fatto che vogliamo garantire un'esecuzione di richieste pari ad A, anche per le virtual machine meno utilizzate. Agendo in questa maniera otteniamo i nuovi periodi di campionamento come espressi in tabella 5.2.

Gli utilizzi[2] relativi al caso sbilanciato sono mostrati in tabella 5.4.

K	U	U_k
16	0,1	0.00625
16	0,3	0.01875
16	0,5	0.03125
16	0,7	0.04375
8	0,1	0.0125
8	0,3	0.0375
8	0,5	0.0625
8	0,7	0.0875
4	0,1	0.025
4	0,3	0.075
4	0,5	0.125
4	0,7	0.175

Tabella 5.3: Utilizzi ripartiti equamente tra i guest per ogni test

K	U	$U_{\frac{k}{4}}$ *high*	$U_{\frac{k}{2}}$ *medium*	$U_{\frac{k}{4}}$ *low*
16	0,1	0.0109375	0.00625	0.0015625
16	0,3	0.0328125	0.01875	0.0046875
16	0,5	0.0546875	0.03125	0.0078125
16	0,7	0.0765625	0.04375	0.0109375
8	0,1	0.021875	0.0125	0.003125
8	0,3	0.065625	0.0375	0.009375
8	0,5	0.109375	0.0625	0.015625
8	0,7	0.153125	0.0875	0.021875
4	0,1	0.04375	0.025	0.00625
4	0,3	0.13125	0.075	0.01875
4	0,5	0.21875	0.125	0.03125
4	0,7	0.40625	0.175	0.04375

Tabella 5.4: Utilizzi ripartiti secondo il fattore $F = 0.75$ tra i guest per ogni test

5.2 Risultati

Di seguito sono riportati i risultati ottenuti per quanto riguarda i tempi di risposta.

[2]es. $U_{\frac{k}{4}}$ *high* significa che per $\frac{1}{4}$ delle virtual machine su un totale di K è stato applicato quell'utilizzo

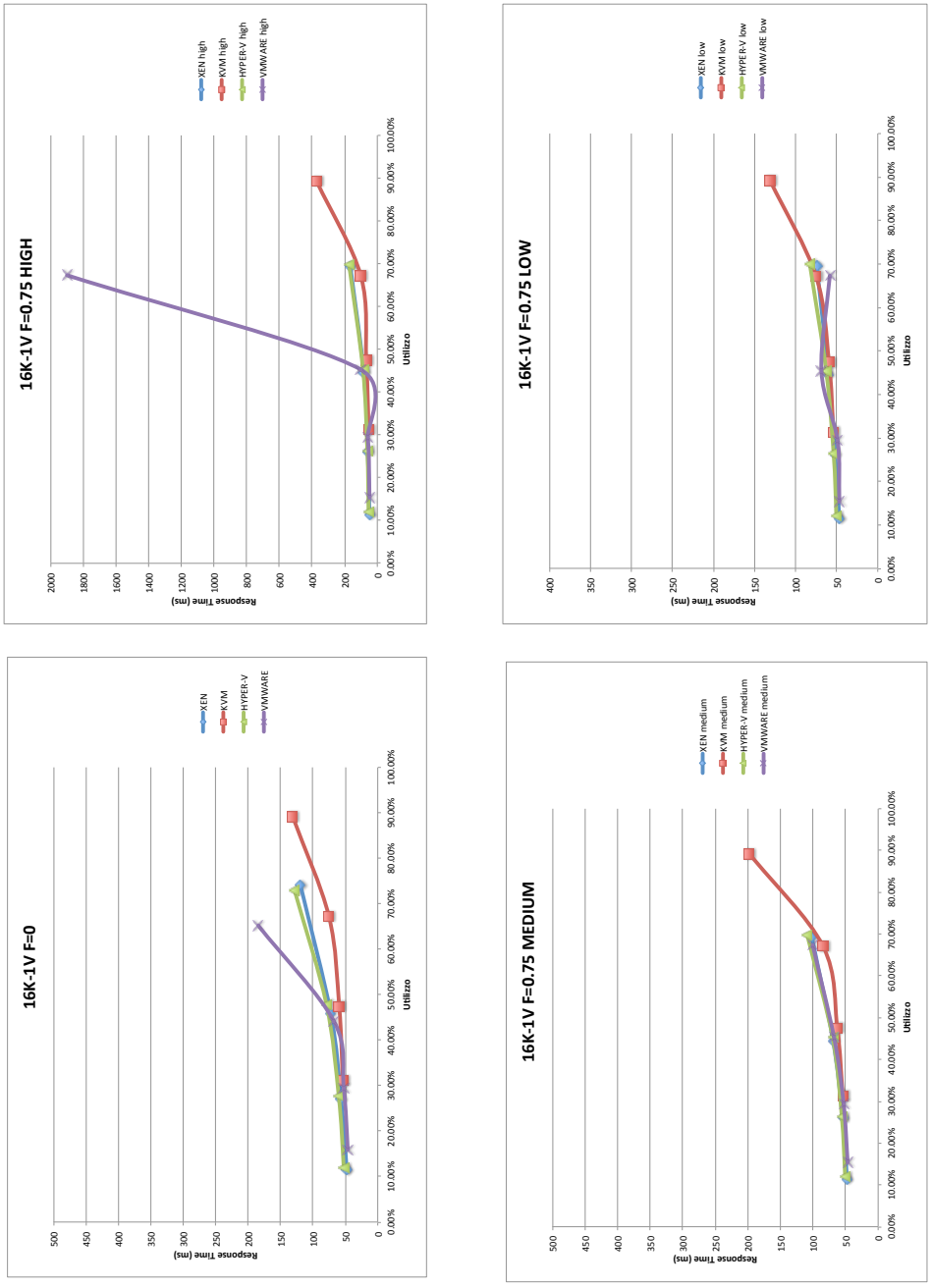


Figura 5.3: Confronto tra tecnologie per 16K - 1V con hyperthreading

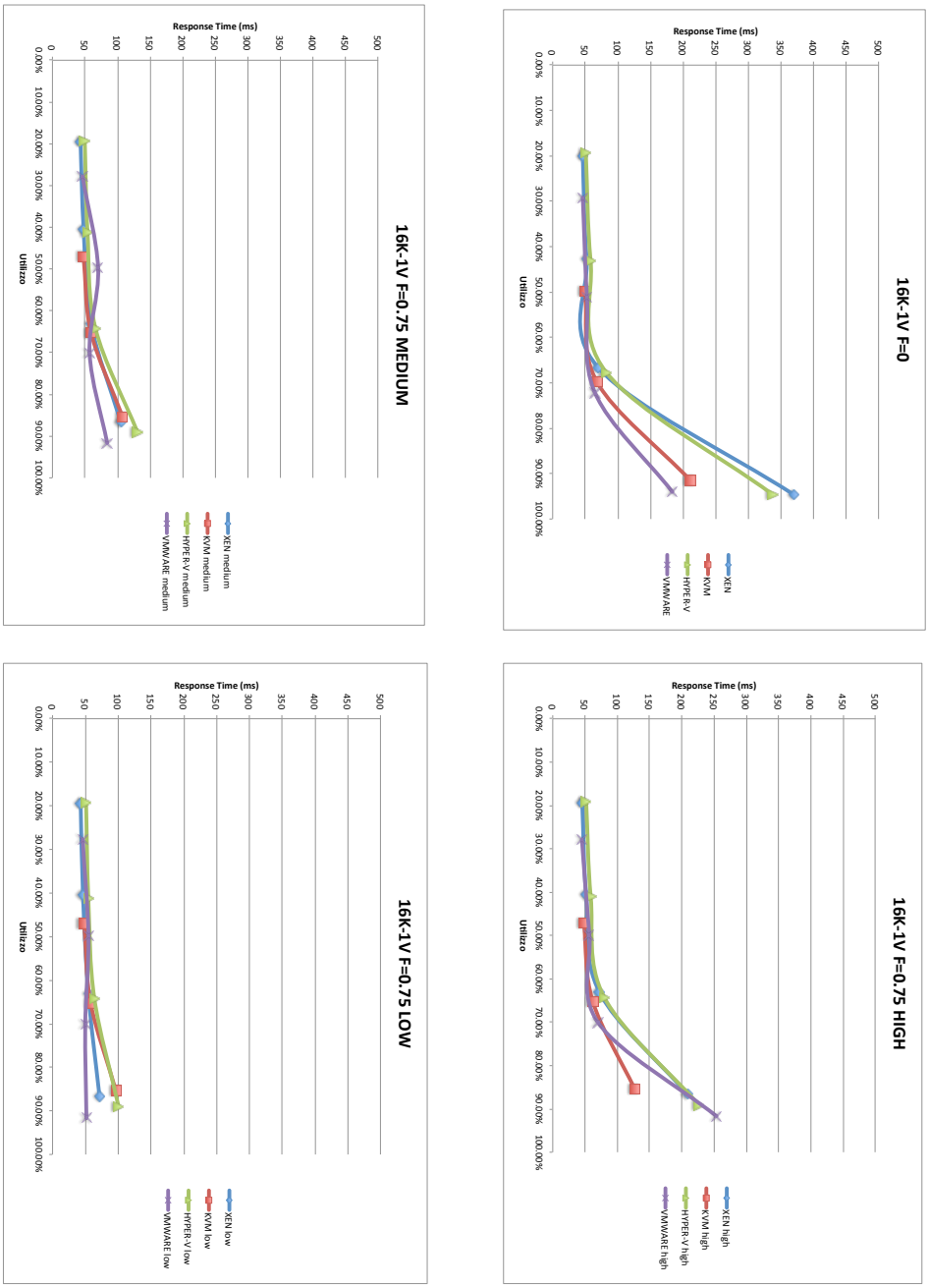


Figura 5.4: Confronto tra tecnologie per 16K - 1V senza hyperthreading

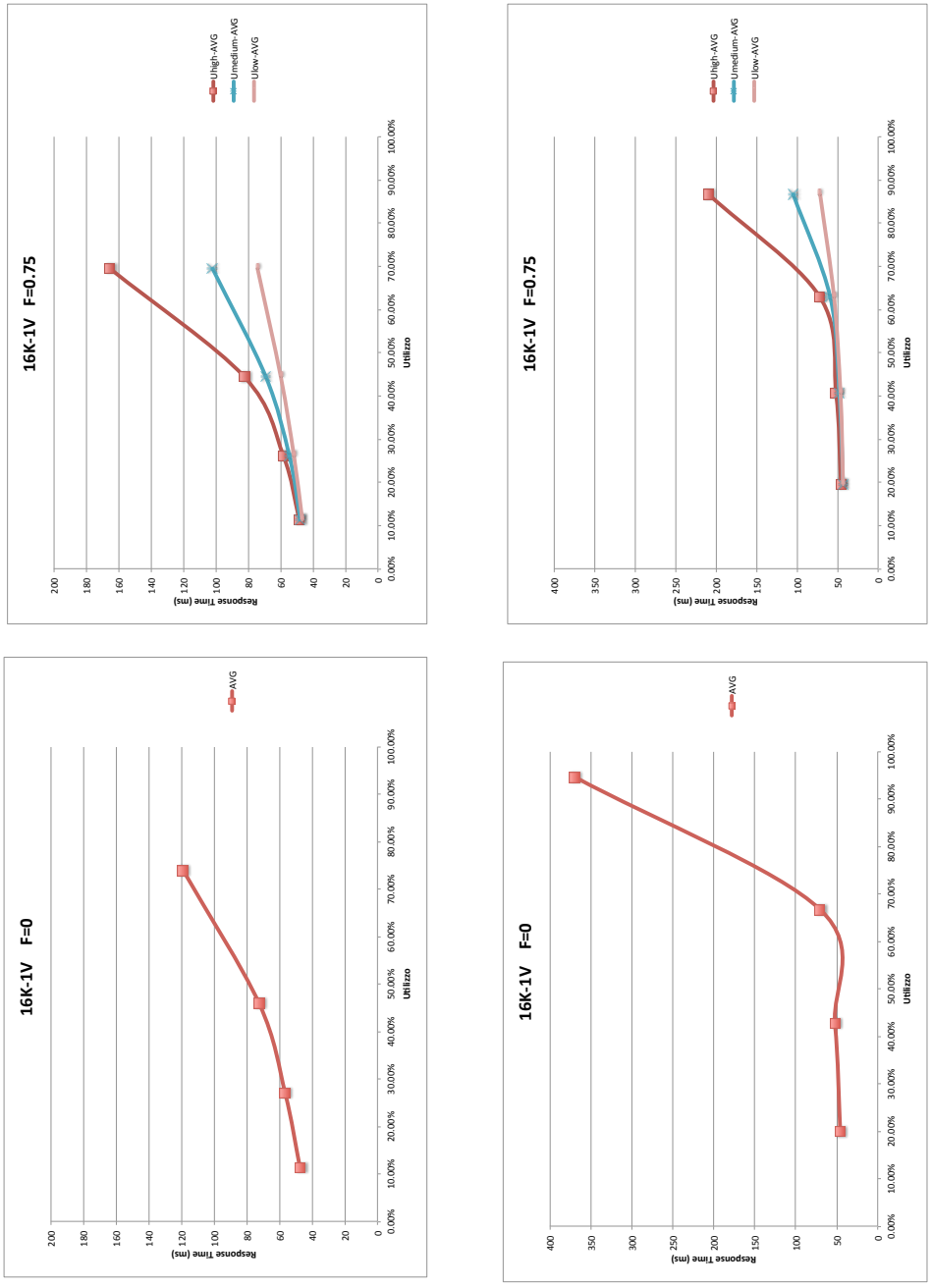


Figura 5.5: Xen per 16K - 1V con e senza hyperthreading: confronto tra $F = 0$ e $F = 0.75$

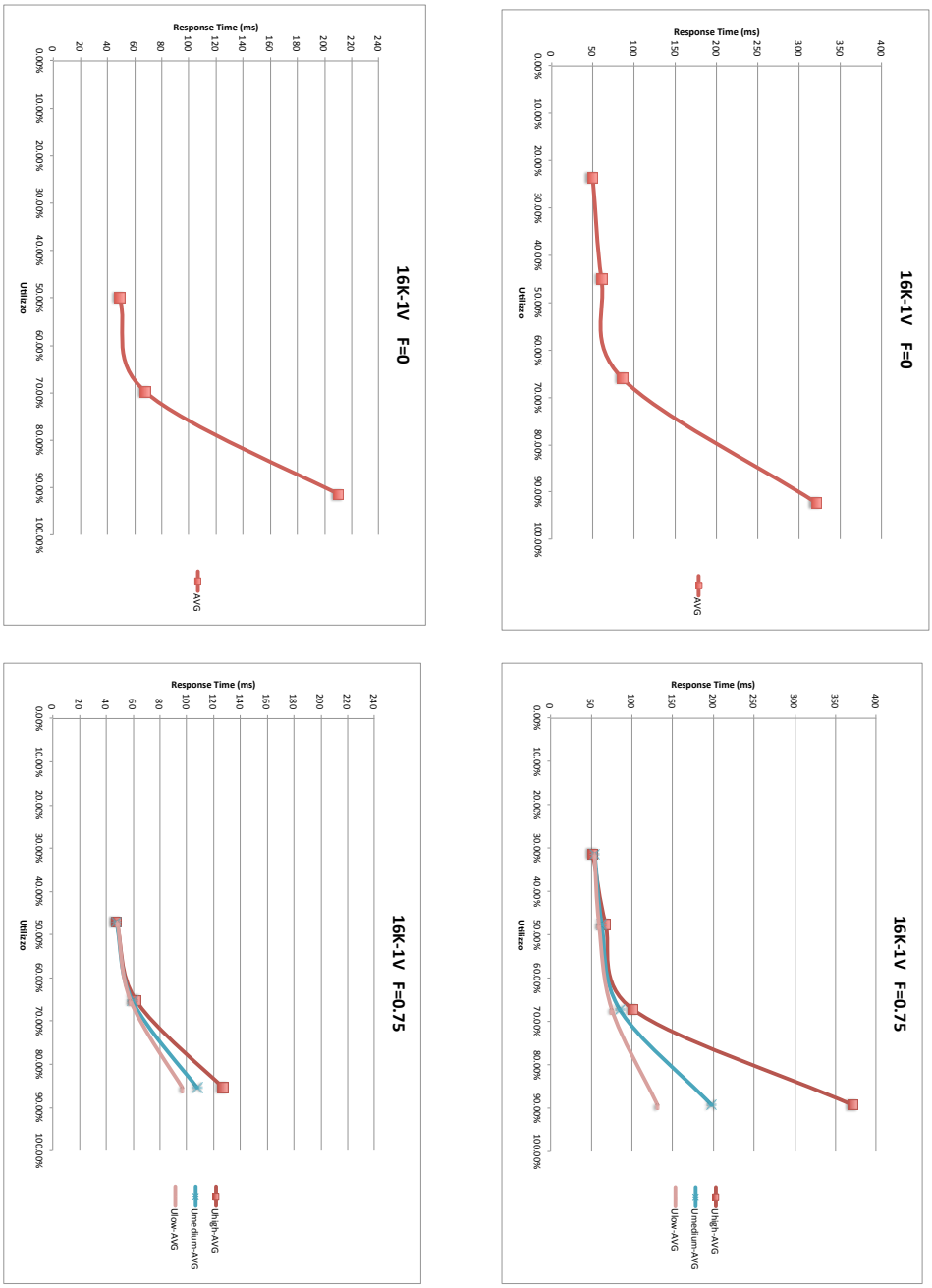


Figura 5.6: KVM per 16K - 1V con e senza hyperthreading: confronto tra $F = 0$ e $F = 0.75$

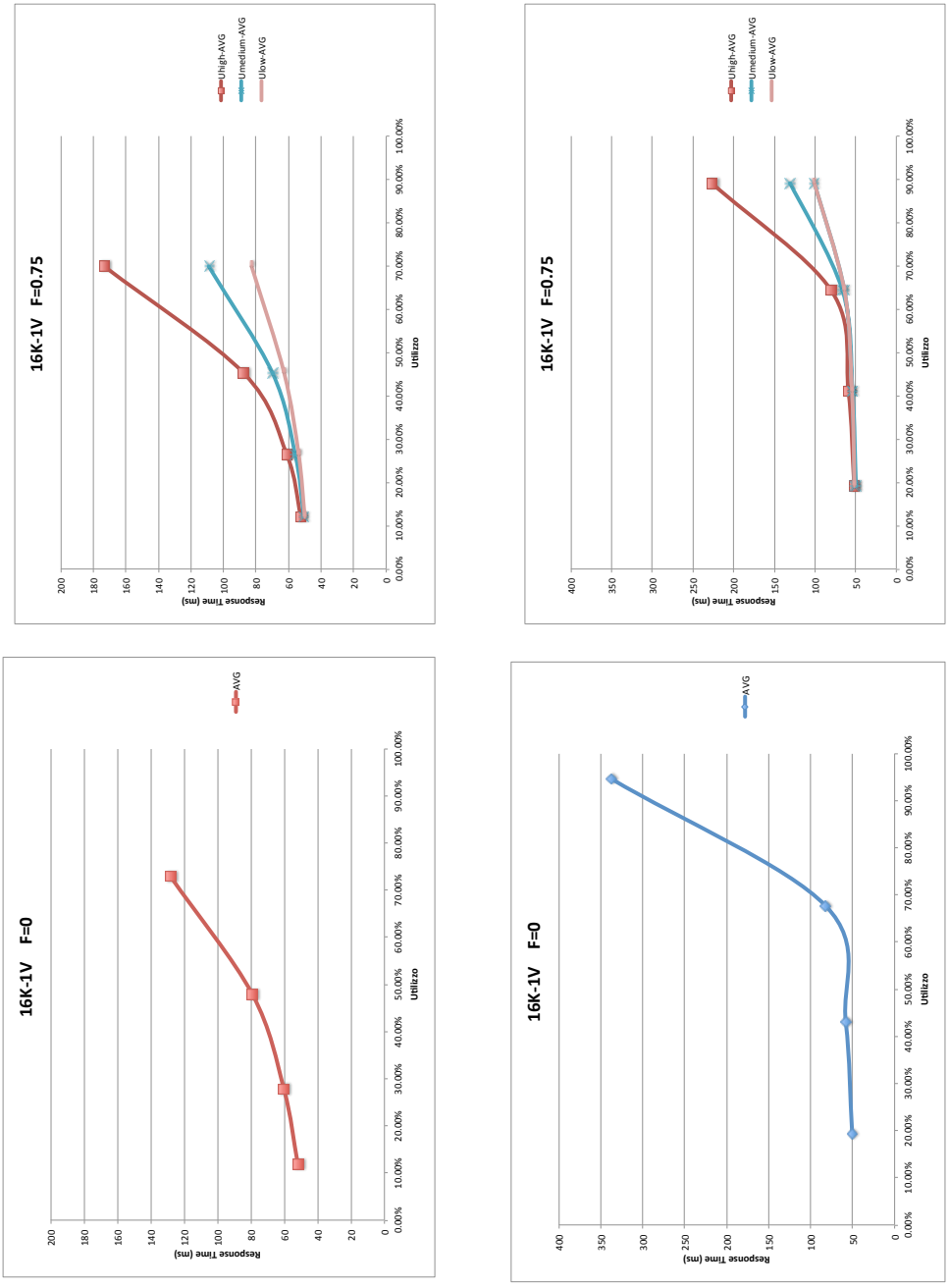


Figura 5.7: Hyper-V per 16K - 1V con e senza hyperthreading: confronto tra $F = 0$ e $F = 0.75$

5.3 Conclusioni

I grafici relativi ai casi con 8 e 4 guest sono riportati nell'appendice A. Le conclusioni alle quali siamo giunti sono le seguenti:

- Xen e Hyper-V mostrano in generale un comportamento simile. Infatti le figure 5.5 e 5.7 lo dimostrano.
- Si può dire che utilizzando l'hyperthreading, i tempi di risposta si mantengono minori rispetto al caso senza hyperthreading ma, per utilizzi bassi (diciamo dal 10% al 50%) si nota che il caso senza hyperthreading mostra un andamento "piatto", tipico dei modelli $m/m/n$, mentre il caso con hyperthreading mostra da subito un incremento al crescere dell'utilizzo.
- I grafici relativi al confronto tra $F = 0$ e $F = 0.75$ all'interno della stessa tecnologia (figure 5.5, 5.6, 5.7) mettono in luce il comportamento dello scheduler degli hypervisor che, a fronte di una modifica degli utilizzi delle virtual machine, assegna ad esse una priorità dinamica. Infatti, i guest meno utilizzati mostrano in generale un response time minore rispetto a quelle più utilizzate. Il fatto che utilizzi diversi assegnati a macchine virtuali identiche risulti in tempi di risposta differenti è in contrasto con quanto viene affermato nella teoria delle code per modelli $m/m/n$ multiclasse e in particolare quelli i tipo processor sharing. Secondo questi modelli i tempi di risposta sono gli stessi per le diverse classi di workload, indipendentemente dalle risorse utilizzate. Gli hypervisor cercano invece di isolare l'impatto prestazionale delle virtual machine. Pertanto la QoS di un guest poco utilizzato viene almeno parzialmente protetta anche quando altre macchine virtuali consumano gran parte delle risorse. Se da un lato questo fattore rappresenta un vantaggio, non esistono però modelli che consentono di fare previsioni e che possono essere quindi utilizzati durante operazioni di dimensionamento dei server o di scelta delle strategie per la server consolidation.
- I grafici relativi al confronto tra tecnologie (figure 5.3 e 5.4) mettono in risalto il comportamento dei diversi hypervisor che, per quanto simili tra Xen ed Hyper-V, si differenziano nei confronti di KVM. Questo hypervisor di tipo hosted nel kernel Linux mostra in generale un response time maggiore e soprattutto un utilizzo maggiore dell'host. Infatti quando viene impostato un utilizzo totale pari al 10%, si ottiene come primo valore utile per il campionamento un utilizzo del 30%. Andando ad indagare il motivo di questo comportamento inatteso, abbiamo scoperto che il kernel Linux

2.6.32 mette in esecuzione un processo che occupa circa il 20% della CPU totale ed esso prende il nome di Kernel Shared Memory (KSM). Questo processo si occupa di duplicare le pagine di memoria in una singola pagina; ciò dovrebbe risultare utile soprattutto nel momento in cui, in un ambiente virtualizzato, si hanno più guest su cui è installato lo stesso sistema operativo; ciò significa avere molte pagine di memoria identiche. In realtà se un workload è di tipo CPU-intensive, questo risulta in un utilizzo della CPU aggiuntivo che aumenta i tempi di risposta dell'host. Tale aspetto ci ha portato a capire il motivo per il quale anche negli esperimenti relativi al throughput massimo (Capitolo 4) KVM mostrasse in generale un throughput più basso rispetto alle altre tecnologie.

Capitolo 6

Conclusioni e sviluppi futuri

Attraverso questo lavoro di tesi è stato possibile evidenziare come il benchmarking dei sistemi virtualizzati sia estremamente importante nell'ambito della valutazione delle performance. In primo luogo, rilevando il thorughtput massimo in funzione del numero di virtual machine e di virtual cpu, abbiamo quantificato l'overhead introdotto dalle varie tecnologie di virtualizzazione. All'aumentare del numero di vm, KVM mostra un calo prestazionale più elevato rispetto a Xen ed Hyper-V, mentre VMware meglio si avvicina alle prestazioni di un sistema nativo.

Successivamente, osservando i tempi di risposta con un carico a modello aperto, abbiamo illustrato la mancanza di modelli matematici che consentano di fare previsioni sul comportamento degli hypervisor. Infatti, differenziando i tempi medi di interarrivo, e quindi gli utilizzi delle diverse virtual machine, si è osservato che i tempi medi di risposta sono eterogenei. Questo è in conflitto con i tradizionali modelli a coda di tipo processor sharing. In altri termini il carico presente su ciascuna virtual machine introduce delle priorità dinamiche all'interno del meccanismo di scheduling.

Come lavoro futuro possibile, utilizzando i dati ottenuti, si potranno eseguire delle analisi statistico-probabilistiche dei risultati che conducano alla validazione di un modello che consenta di fare previsioni circa il comportamento degli hypervisor.

Appendice A

Grafici response time

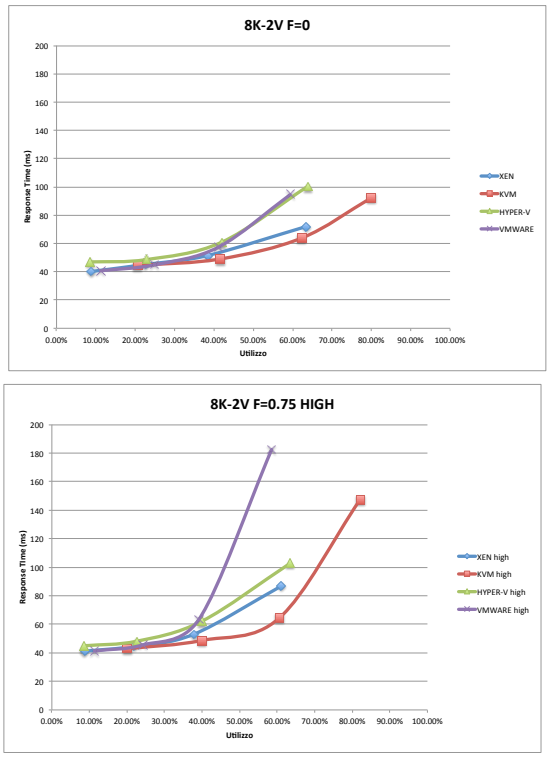
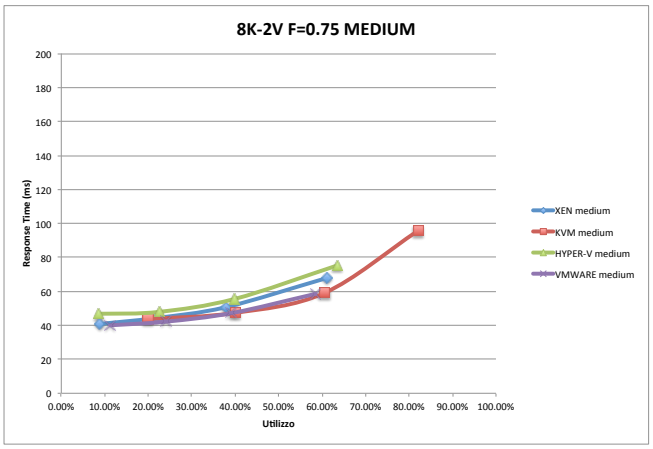


Figura A.1: Confronto tra tecnologie per 8K - 2V con hyperthreading



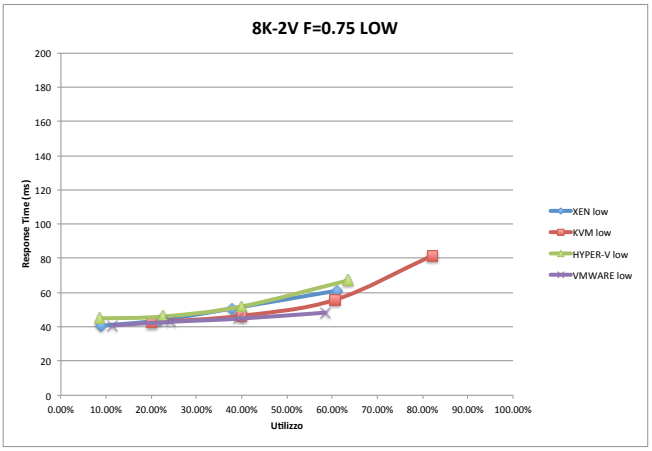


Figura A.2: Confronto tra tecnologie per 8K - 2V con hyperthreading

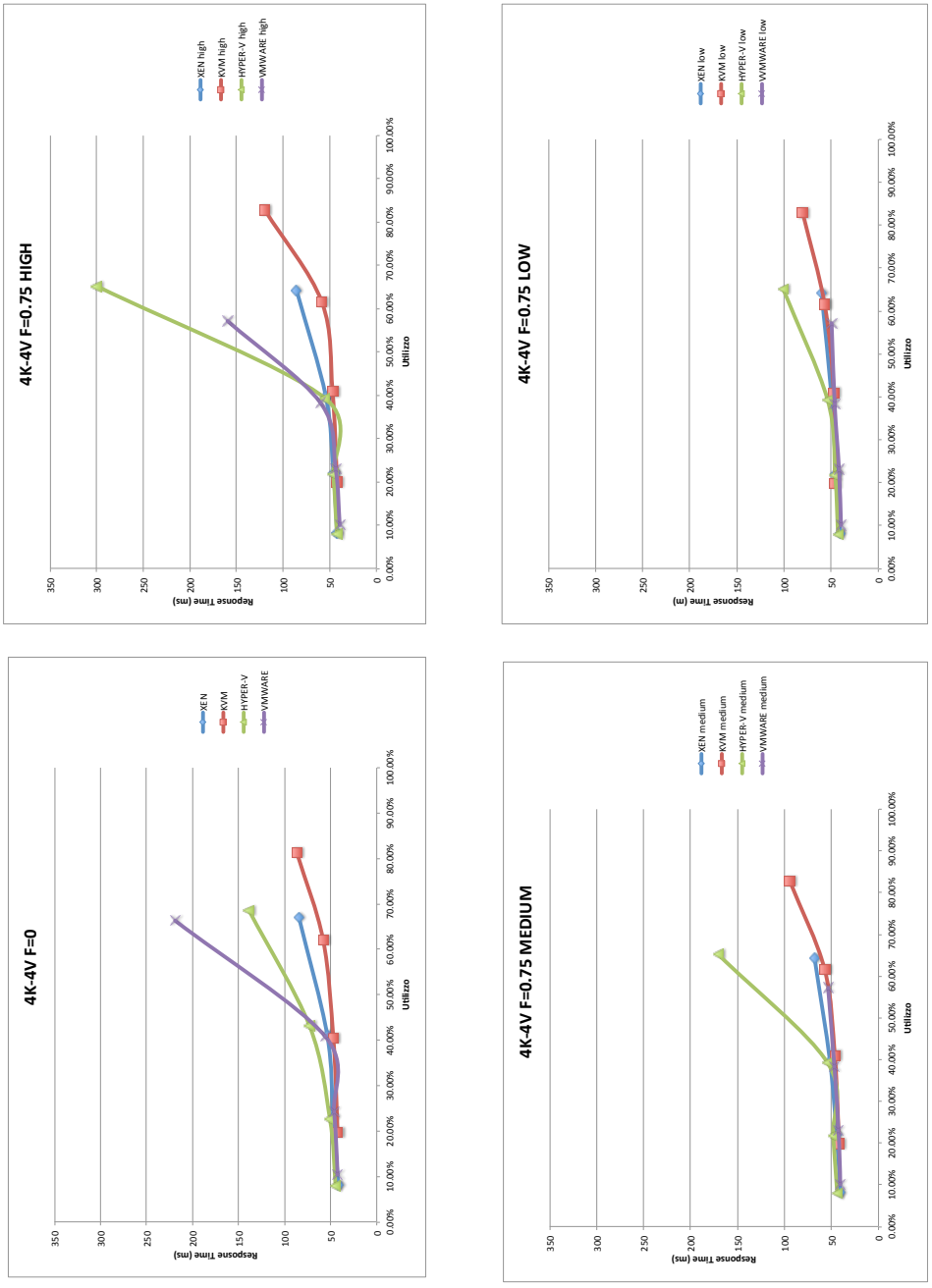


Figura A.3: Confronto tra tecnologie per 4K - 4V con hyperthreading

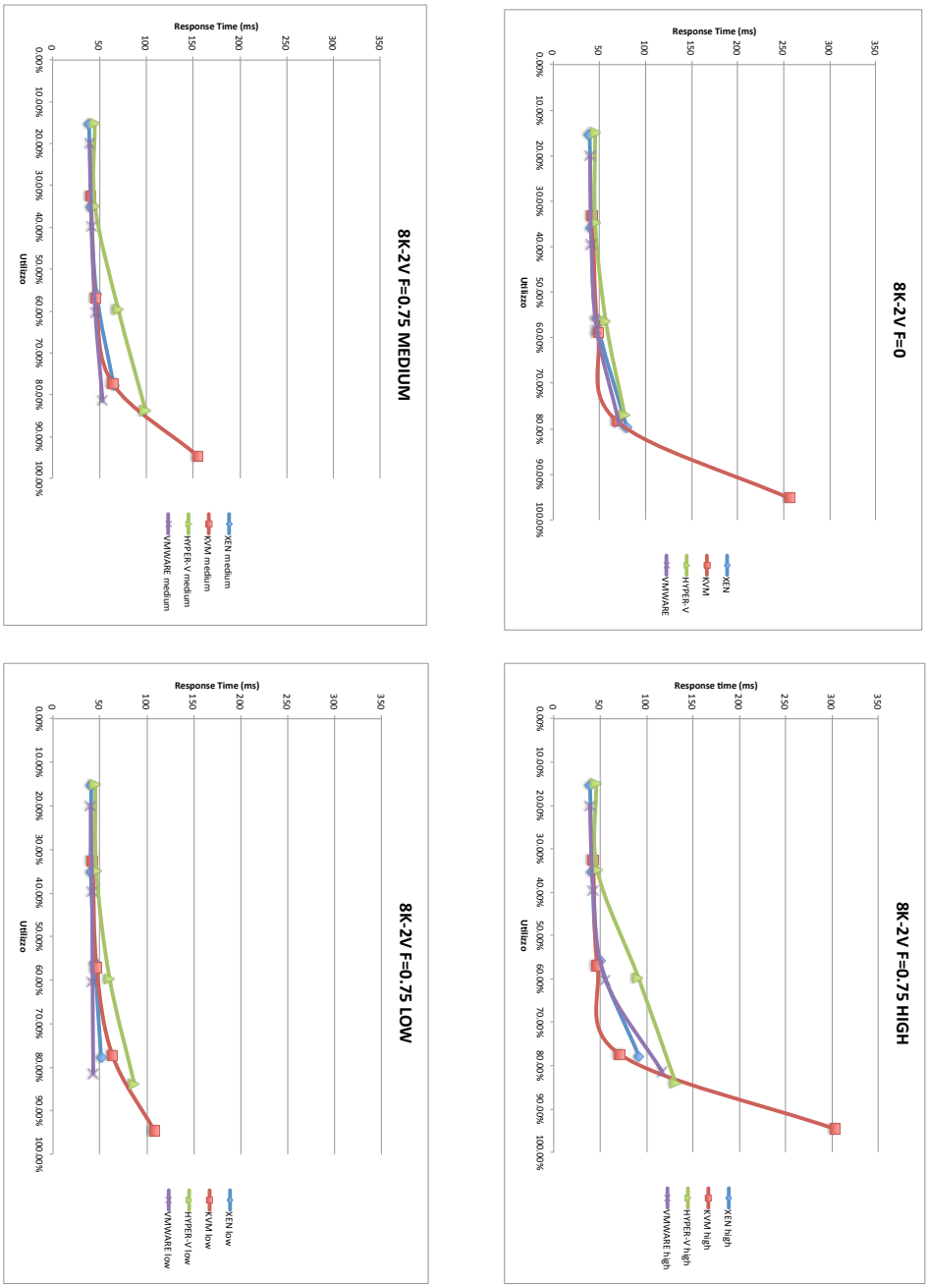


Figura A.4: Confronto tra tecnologie per 8K - 2V senza hyperthreading

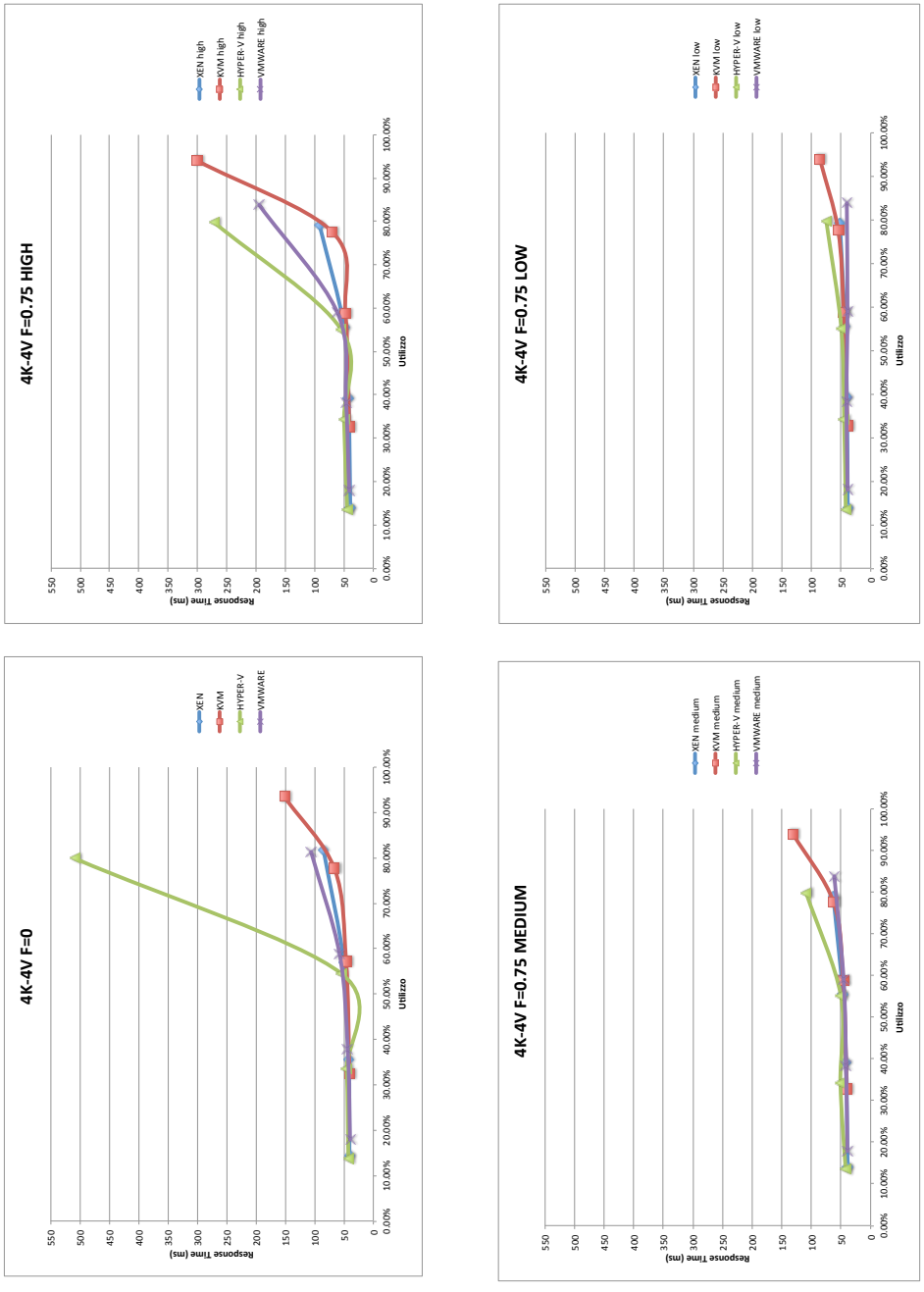


Figura A.5: Confronto tra tecnologie per 4K - 4V senza hyperthreading

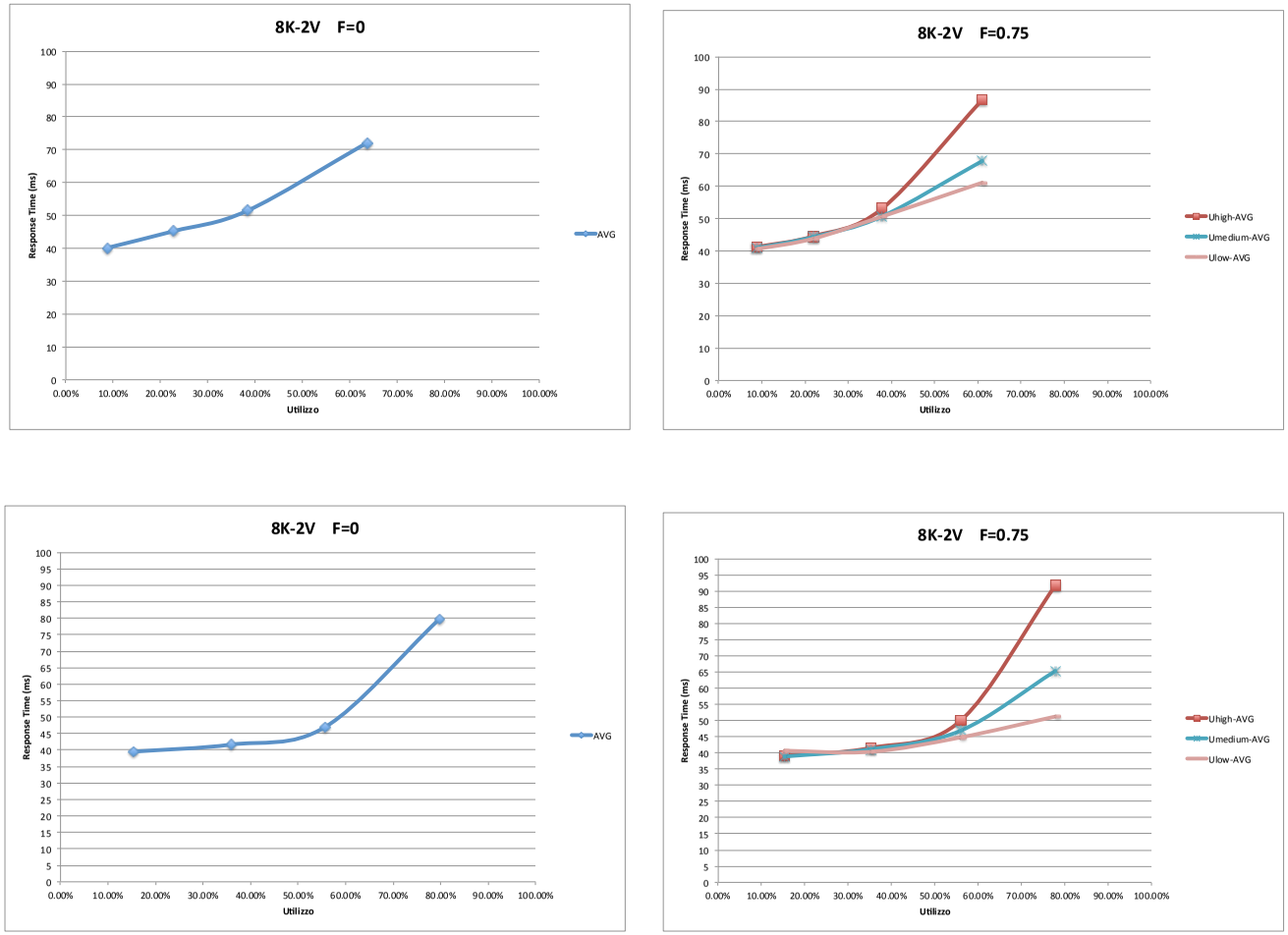
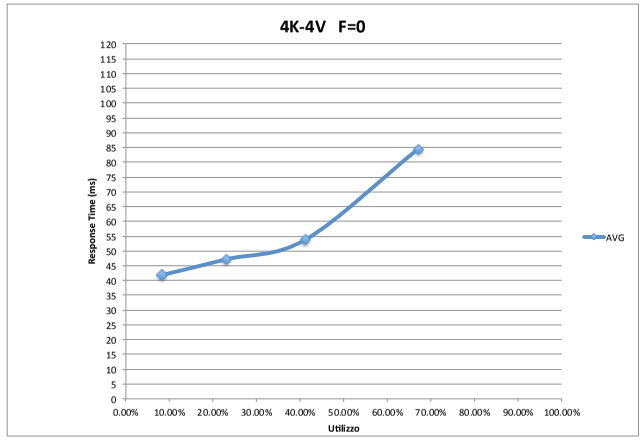
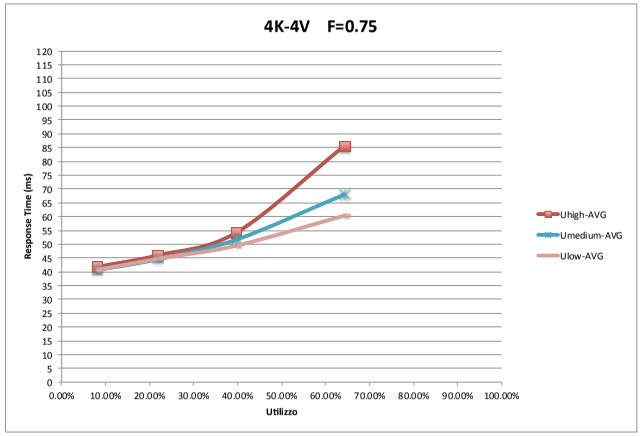
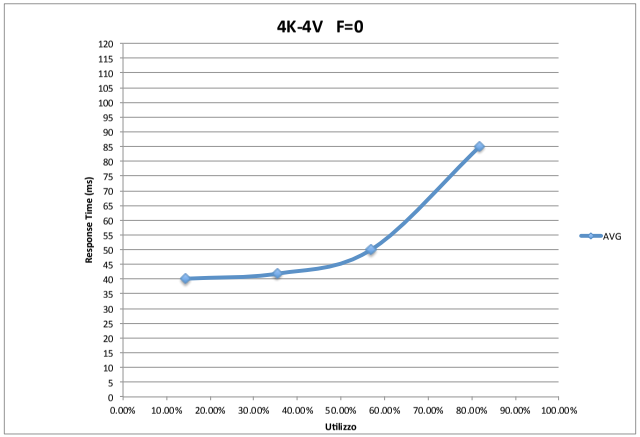


Figura A.6: Xen per 8K - 2V con e senza hyperthreading: confronto tra $F = 0$ e $F = 0.75$







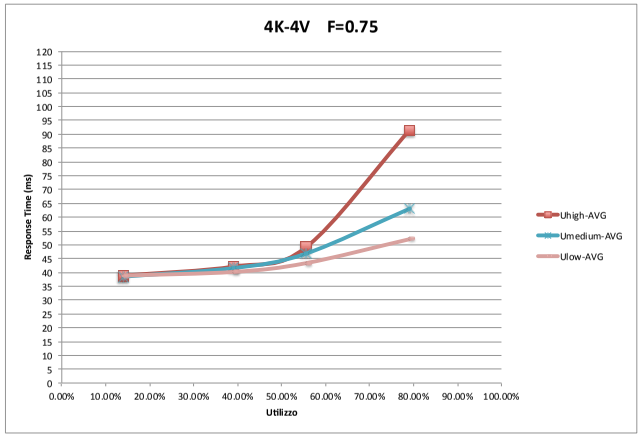


Figura A.7: Xen per 4K - 4V con e senza hyperthreading: confronto tra $F = 0$ e $F = 0.75$

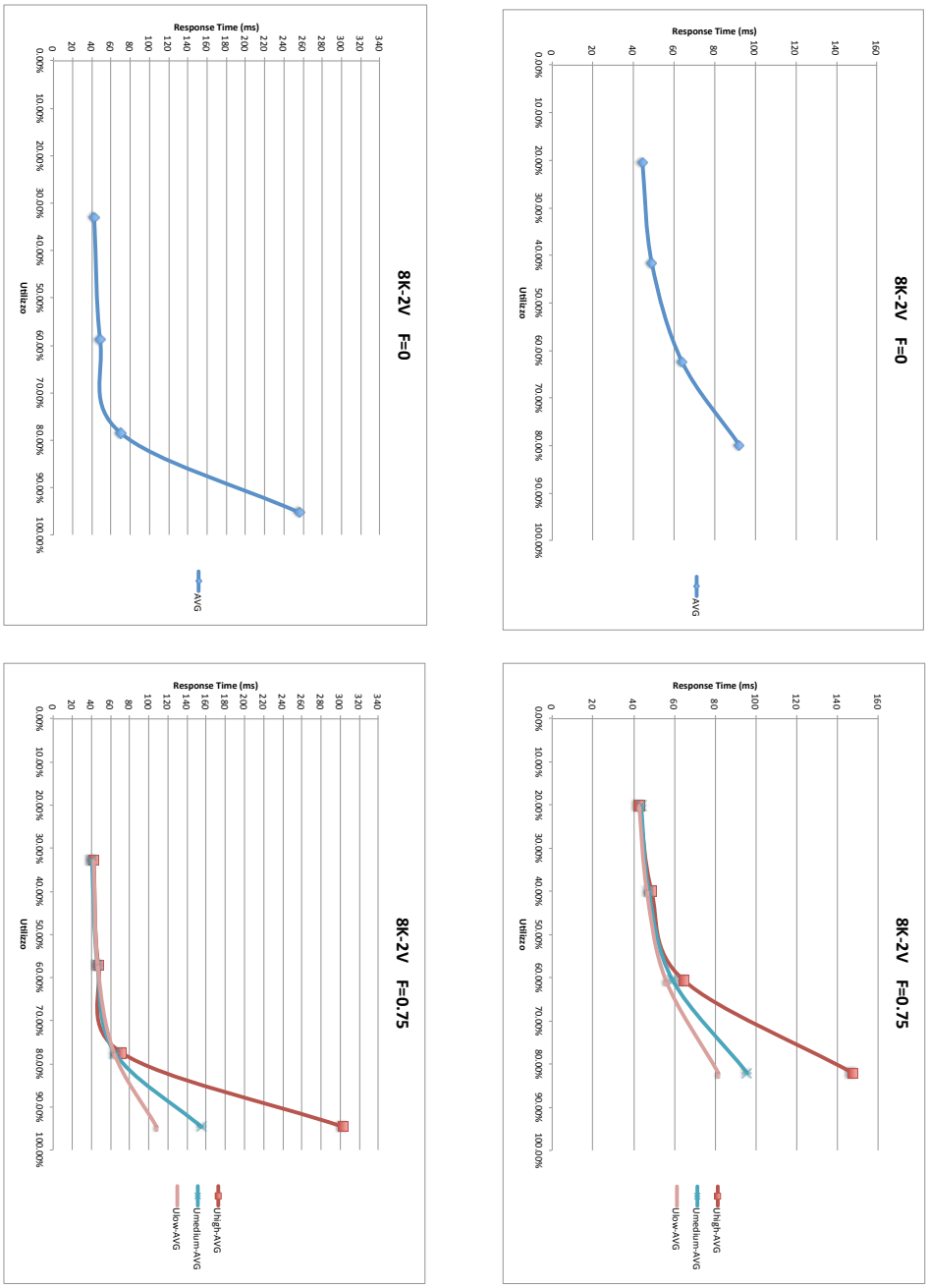
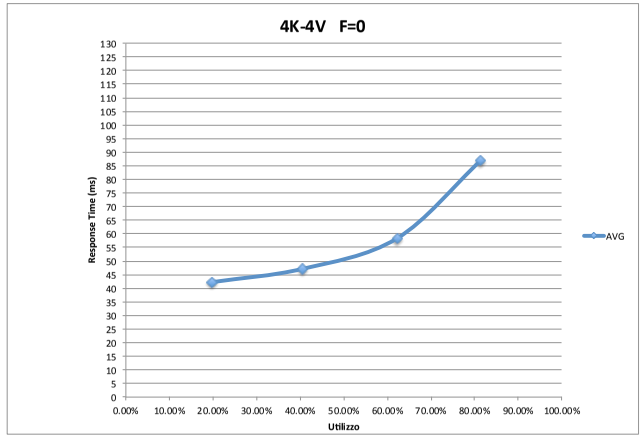
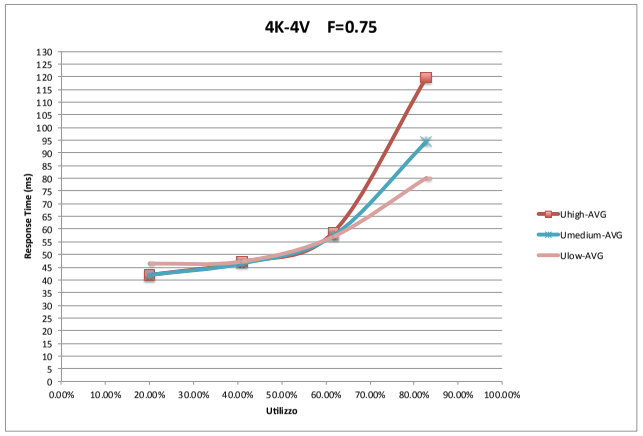
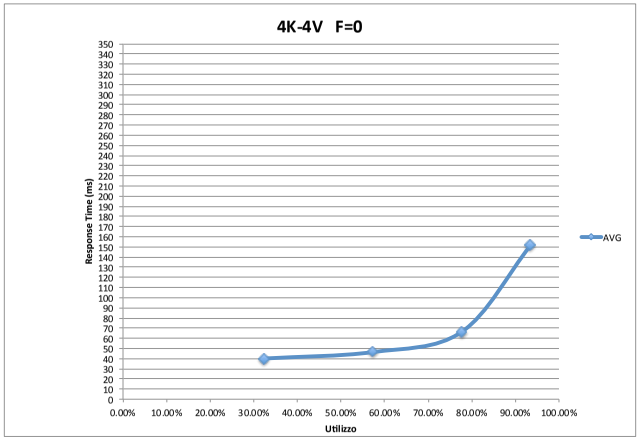


Figura A.8: KVM per 8K - 2V con e senza hyperthreading: confronto tra $F = 0$ e $F = 0.75$







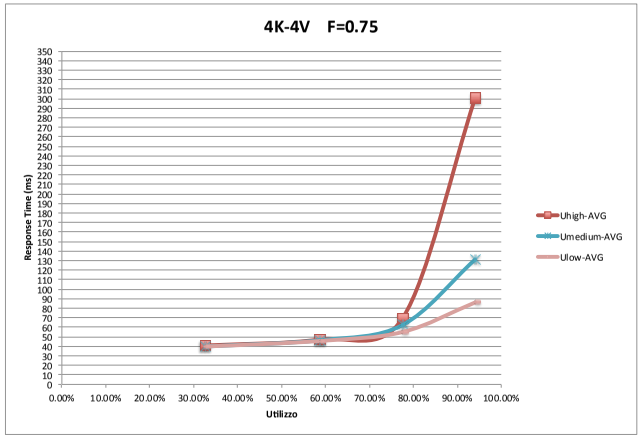


Figura A.9: KVM per 4K - 4V con e senza hyperthreading: confronto tra $F = 0$ e $F = 0.75$

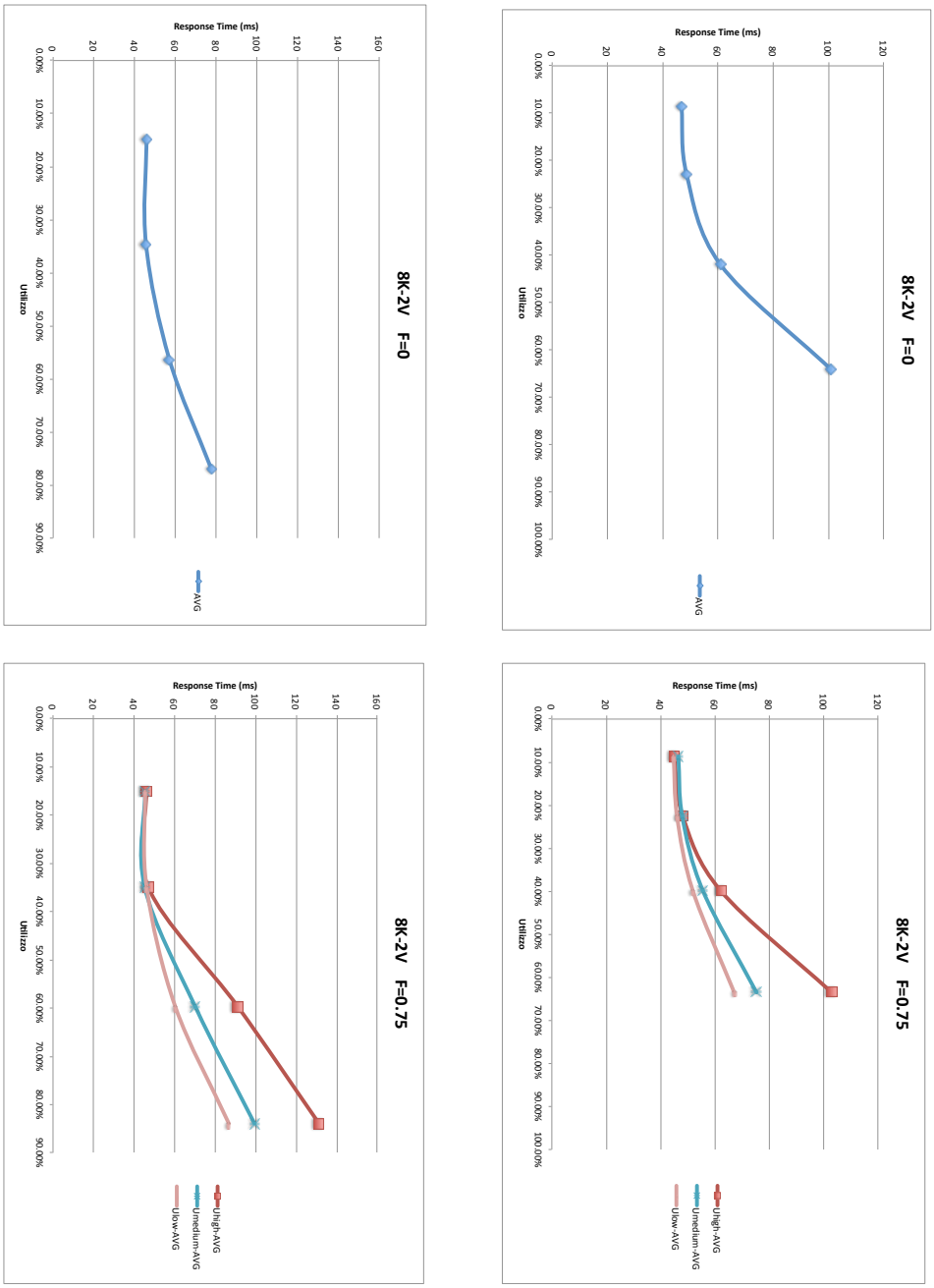


Figura A.10: Hyper-V per 8K - 2V con e senza hyperthreading: confronto tra $F = 0$ e $F = 0.75$

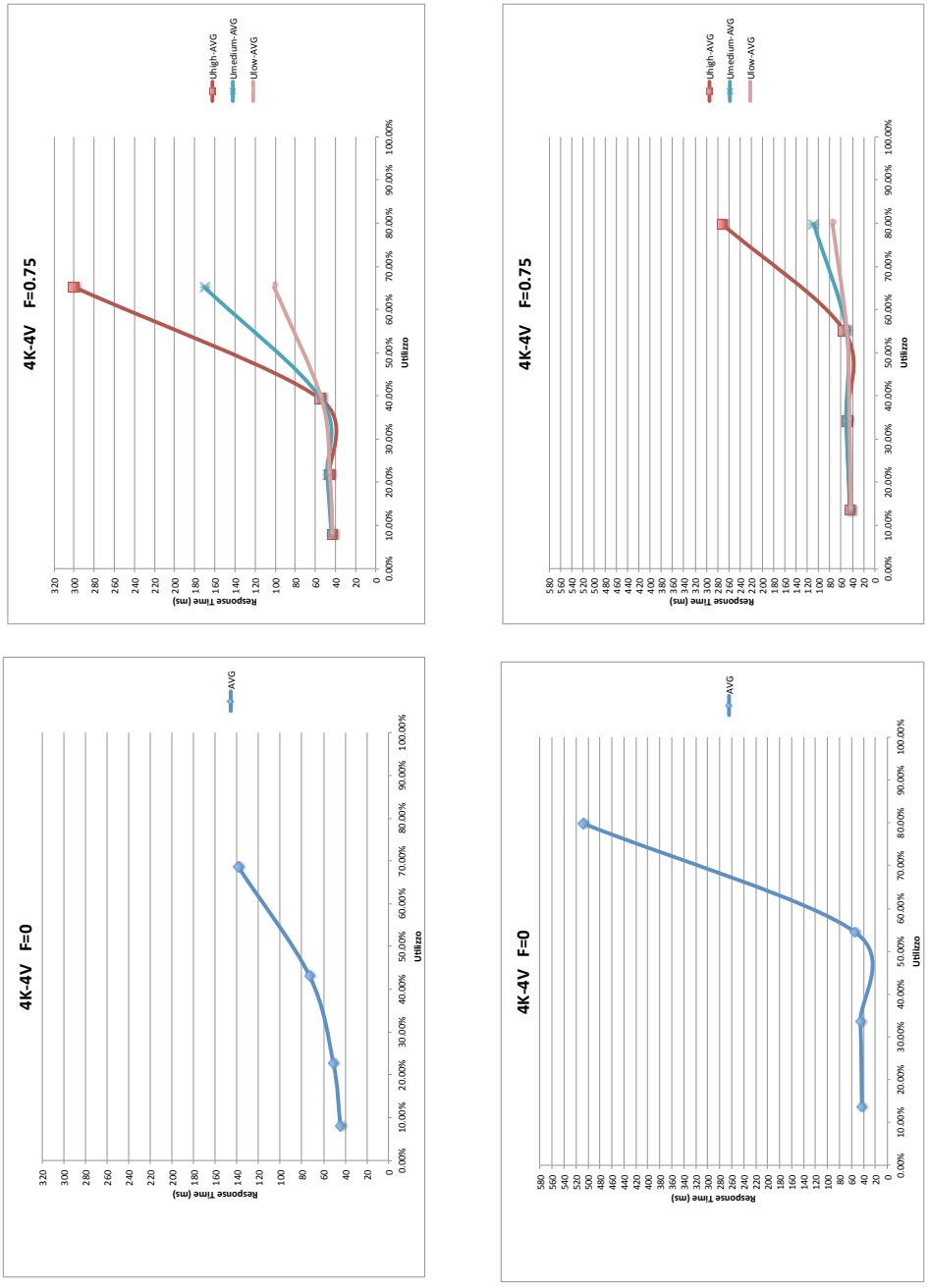


Figura A.11: Hyper-V per 4K - 4V con e senza hyperthreading: confronto tra $F = 0$ e $F = 0.75$

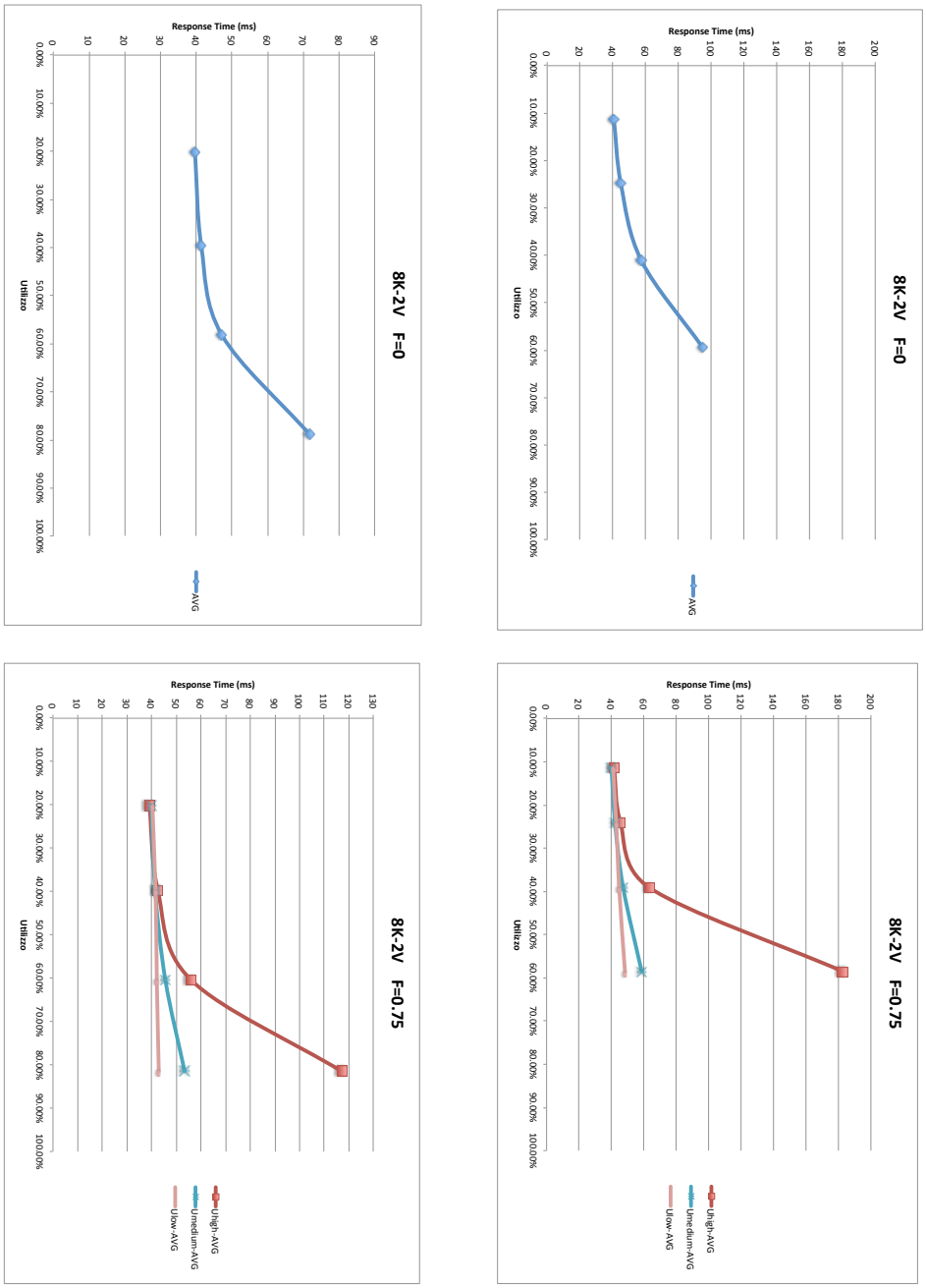


Figura A.12: VMware per 8K - 2V con e senza hyperthreading: confronto tra $F = 0$ e $F = 0.75$

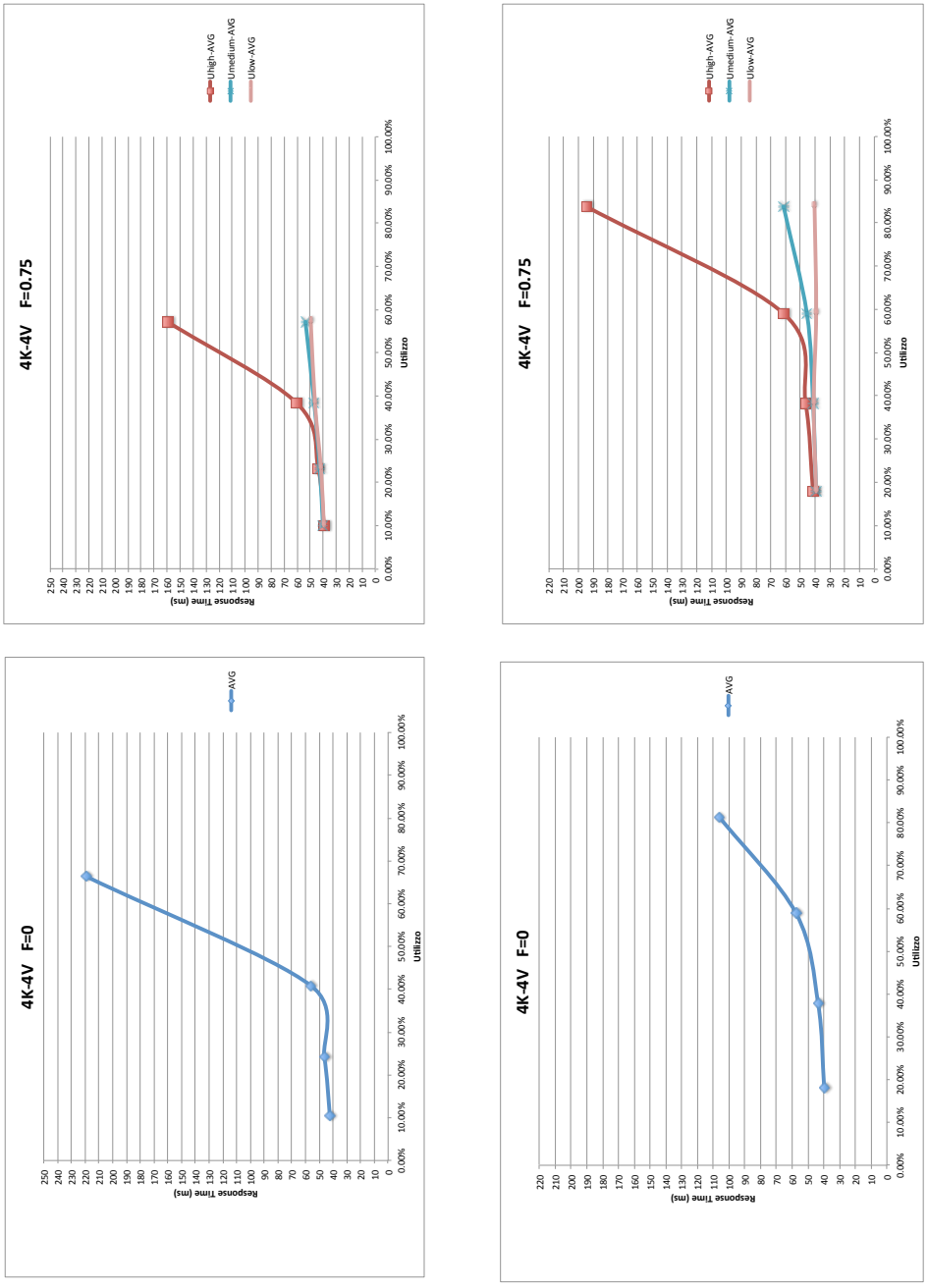


Figura A.13: VMware per 4K - 4V con e senza hyperthreading: confronto tra $F = 0$ e $F = 0.75$

Bibliografia

[1] M. Boari and S. Balboni, *Tecniche di virtualizzazione: teoria e pratica*.

[2] J. E. Smith and R. Nair, *The Architecture of Virtual Machines*. Los Alamitos, CA, USA: IEEE Computer Society, 2005.

[3] N. Kiyanclar, *A Survey of Virtualization Techniques Focusing on Secure On-Demand Cluster Computing*, vol. abs/cs/0511010. 2005.

[4] J. Sahoo, S. Mohapatra, and R. Lath, "Virtualization: A survey on concepts, taxonomy and associated security issues," *Computer and Network Technology, International Conference on*, vol. 0, pp. 222–226, 2010.

[5] R. Rose, *Survey of System Virtualization Techniques*. 2004.

[6] J. P. van Hoogeveen, *Consolidating Servers and Applications With Solaris Containers*.

[7] S. Brandon, A. Chatterjee, H. Gammelmark, V. Mekala, L. Rosca, and A. Sanyal, *Workload Partition Management in IBM AIX Version 6.1*.

[8] M. Fenn, M. A. Murphy, J. Martin, and S. Goasguen, *An Evaluation of KVM for Use in Cloud Computing*.

[9] K. Adams and O. Agesen, *A Comparison of Software and Hardware Techniques for x86 Virtualization*. San Jose, California, USA: ACM Press, Oct. 2006.

[10] M. Chapman, D. J. Magenheimer, and P. Ranganathan, *MagiXen: Combining Binary Translation and Virtualization*. 2007.

[11] M. Rosenblum and T. Garfinkel, *Virtual Machine Monitors: Current Technology and Future Trends*, vol. 38. Los Alamitos, CA, USA: IEEE Computer Society, 2005.

[12] VMware, *Performance Evaluation of Intel EPT Hardware Assist*.

[13] M. Cloutier, *VMware Virtualization and Software Development.*

[14] P. Barham, B. Dragovic, K. Fraser, S. Hand, T. Harris, A. Ho, R. Neugebauer, I. Pratt, and A. Warfield, *Xen and the art of virtualization*, vol. 37. New York, NY, USA: ACM, October 2003.

[15] `http://www.microsoft.com/hyper-v-server/en/us/default.aspx`.

[16] A. Kivity, Y. Kamay, D. Laor, U. Lublin, and A. Liguori, *Kvm: The Linux Virtual Machine Monitor.*

[17] A. Hoetzel, A. Benhaim, N. Griffiths, C. Holliday, and N. Pistoor, *Benchmarking in Focus.*

[18] `http://www.spec.org/`.

[19] `http://www.tpc.org/`.

[20] P. Padala, X. Zhu, Z. Wang, S. Singhal, and K. G. Shin, *Performance Evaluation of Virtualization Technologies for Server Consolidation.*

[21] T. Deshane, Z. Shepherd, J. Matthews, M. Ben-Yehuda, A. Shah, and B. Rao, *Quantitative comparison of Xen and KVM.* Berkeley, CA, USA: USENIX association, June 2008.

[22] E. Hofmann, *Evoluzione e prospettive nell'High Performance Computing.*

[23] M. F. Mergen, V. Uhlig, O. Krieger, and J. Xenidis, *Virtualization for high-performance computing*, vol. 40. New York, NY, USA: ACM, April 2006.

[24] A. Tikotekar, G. Vallèe, T. Naughton, H. Ong, C. Engelmann, and S. L. Scott, *An Analysis of HPC Benchmarks in Virtual Machine Environments.* Berlin, Heidelberg: Springer-Verlag, 2009.

[25] A. Gavrilovska, S. Kumar, H. Raj, K. Schwan, V. Gupta, R. Nathuji, R. Niranjan, A. Ranadive, and P. Saraiya, *High-Performance Hypervisor Architectures: Virtualization in HPC Systems.*

[26] T. Deshane, Z. Shepherd, J. N. Matthews, M. Ben-Yehuda, A. Shah, and B. Rao, *Quantitative Comparison of Xen and KVM.*

[27] J. P. Walters, V. Chaudhary, M. Cha, S. G. Jr., and S. Gallo, *A Comparison of Virtualization Technologies for HPC.* Washington, DC, USA: IEEE Computer Society, 2008.

[28] `http://www.vmware.com/products/vmmark/`.

[29] E. D. Lazowska, J. Zahorjan, G. S. Graham, and K. C. Sevcik, *Quantitative System Performance: Computer System Analysis Using Queueing Network Models*. Prentice-Hall, Inc., 1984.

[30] D. T. Marr, F. Binns, D. L. Hill, G. Hinton, D. A. Koufaty, A. J. Miller, and M. Upton, *Hyper-Threading Technology Architecture and Microarchitecture*, vol. 6. Feb. 2002.

[31] M. R. Hines and K. Gopalan, *Post-copy based live virtual machine migration using adaptive pre-paging and dynamic self-ballooning*. VEE '09, New York, NY, USA: ACM, 2009.