

Caratterizzazione dell'impatto sulle prestazioni di sistemi di virtualizzazione in macchine per la manifattura

Tesi di laurea in:
PROGRAMMAZIONE AD OGGETTI

Relatore

Prof. Danilo Pianini

Candidato

Simone Magnani

Correlatori

Ing. Martina Baiardi

Ing. Angelo Filaseta

Sommario

La virtualizzazione è una tecnologia che permette di eseguire più sistemi operativi su una singola macchina. In particolare, le macchine virtuali sono repliche di una macchina reale dove le componenti hardware sono virtualizzate, eseguite da un programma chiamato hypervisor. La virtualizzazione ha un costo in termini di prestazioni causato dall'overhead dell'hypervisor e dalla condivisione delle risorse, e che varia in base alle tecniche e alle tecnologie di virtualizzazione utilizzate.

Questa tesi analizza l'impatto sulle prestazioni di sistemi di virtualizzazione su una macchina HP Pro Mini 400 G9, utilizzata per la manifattura. Sono confrontate le prestazioni di un sistema bare metal e lo stesso virtualizzato, utilizzando gli hypervisor VirtualBox, KVM e Xen, e i sistemi operativi Fedora CoreOS e Windows 10. Nel confronto sono utilizzati dispositivi emulati, la paravirtualizzazione, ed è inoltre effettuato il passthrough mediante SR-IOV della GPU integrata Intel UHD 770.

Sono presentati i risultati dei benchmark di CPU, memoria, disco, GPU e rete. Gli hypervisor VirtualBox, KVM e Xen mostrano prestazioni simili al sistema bare metal nei benchmark di CPU, memoria e rete. Nei benchmark della grafica, la GPU Virtio con accelerazione 3D per applicazioni OpenGL tramite VirGL ottiene prestazioni superiori rispetto al passthrough con SR-IOV, tuttavia quest'ultimo permette di eseguire anche applicazioni grafiche DirectX e di calcolo con OpenCL. Inoltre, SR-IOV non è ancora pienamente supportato per GPU Intel nel kernel upstream di Linux. Le prestazioni dei dischi virtuali basati su file risultano le più impattate dalla virtualizzazione, e caratterizzate da velocità instabili. In termini di funzionalità, VirtualBox risulta limitato dalla mancanza del supporto al PCI passthrough.

Indice

Sommario	iii
1 Introduzione	1
1.1 Contesto e motivazioni	2
1.2 Struttura della tesi	3
2 Stato dell'arte: Virtualizzazione	5
2.1 Classificazione	5
2.1.1 Virtualizzazione a livello di sistema operativo	6
2.1.2 Virtualizzazione a livello di hardware	7
2.2 Virtualizzazione di dispositivi I/O	9
2.2.1 PCI passthrough	10
2.2.2 SR-IOV	10
2.2.3 Virtualizzazione GPU	11
2.2.4 Virtualizzazione dei dispositivi di archiviazione	12
2.3 Tecnologie	13
2.3.1 VirtualBox	13
2.3.2 QEMU	14
2.3.3 KVM	15
2.3.4 Xen	16
2.3.5 libvirt	17
2.3.6 Virtual Machine Manager	18
2.4 Impatto sulle prestazioni: lavori precedenti	21
3 Caso di studio	23
3.1 Caratteristiche delle macchine utilizzate	23
3.2 Configurazione delle macchine virtuali	24

INDICE

3.2.1	VirtualBox	26
3.2.2	KVM	28
3.2.3	Xen	28
3.3	Virtualizzazione GPU	29
3.3.1	Fixed Passthrough	29
3.3.2	Mediated Passthrough tramite SR-IOV	34
4	Confronto delle prestazioni	37
4.1	Strumenti	37
4.1.1	Fedora CoreOS	39
4.1.2	Windows 10	40
4.2	Metodologia	40
4.3	Risultati	42
4.3.1	CPU	43
4.3.2	Memoria	47
4.3.3	Disco	48
4.3.4	GPU	54
4.3.5	Rete	62
4.4	Discussione	65
5	Conclusioni	67
	Bibliografia	71

Elenco delle figure

2.1	Tipi di virtualizzazione	6
2.2	Tipi di hypervisor	8
2.3	Modalità di virtualizzazione implementate in Xen	17
2.4	Finestra principale di virt-manager.	19
2.5	Visualizzazione delle impostazioni di una macchina virtuale in virt-manager.	19
2.6	Visualizzazione dello schermo di una macchina virtuale in virt-manager.	20
4.1	Benchmark della CPU con compress-7zip su Fedora CoreOS	44
4.2	Benchmark della CPU con compress-7zip su Windows 10	45
4.3	Benchmark PassMark PerformanceTest, CPU, Fedora CoreOS	46
4.4	Benchmark PassMark PerformanceTest, CPU, Windows 10	46
4.5	Benchmark PassMark PerformanceTest, memoria, Fedora CoreOS	47
4.6	Benchmark PassMark PerformanceTest, memoria, Windows 10	48
4.7	Benchmark fio, letture sequenziali con blocco di 1MiB, 8 richieste in coda, 1 thread, Fedora CoreOS	50
4.8	Benchmark fio, letture casuali con blocco di 4KiB, 1 richiesta in coda, 1 thread, Fedora CoreOS	51
4.9	Benchmark fio, scritture sequenziali con blocco di 1MiB, 8 richieste in coda, 1 thread, Fedora CoreOS	51
4.10	Benchmark fio, scritture casuali con blocco di 4KiB, 1 richiesta in coda, 1 thread, Fedora CoreOS	52
4.11	Benchmark diskspd, letture sequenziali con blocco di 1MiB, 8 richieste in coda, 1 thread, Windows 10	52
4.12	Benchmark diskspd, letture casuali con blocco di 4KiB, 1 richiesta in coda, 1 thread, Windows 10	53

ELENCO DELLE FIGURE

4.13	Benchmark diskspd, scritture sequenziali con blocco di 1MiB, 8 richieste in coda, 1 thread, Windows 10	53
4.14	Benchmark diskspd, scritture casuali con blocco di 4KiB, 1 richiesta in coda, 1 thread, Windows 10	54
4.15	Benchmark PassMark PerformanceTest, grafica 3D DirectX 9, Windows 10	57
4.16	Benchmark PassMark PerformanceTest, grafica 3D DirectX 10, Windows 10	58
4.17	Benchmark PassMark PerformanceTest, grafica 3D DirectX 11, Windows 10	58
4.18	Benchmark PassMark PerformanceTest, grafica 3D DirectX 12, Windows 10	59
4.19	Benchmark Unigine Heaven, grafica 3D OpenGL, Fedora CoreOS	59
4.20	Benchmark Unigine Heaven, grafica 3D OpenGL, Windows 10	60
4.21	Benchmark vkmark, grafica 3D Vulkan, Fedora CoreOS	60
4.22	Benchmark hashcat, computazione GPU OpenCL, Fedora CoreOS	61
4.23	Benchmark iperf3, Fedora CoreOS	63
4.24	Benchmark iperf3, Windows 10	64

Elenco dei listati

2.1	Risultato del comando <code>lspci</code> con un VF attivato mediante SR-IOV	11
3.1	Errore riscontrato utilizzando l'accelerazione 3D in VirtualBox. . . .	27
3.2	Dispositivi PCI della macchina PC-HP, suddivisi per gruppi IOMMU	30
3.3	Messaggi relativi al modulo <code>i915</code> nel log del sistema guest Linux, dopo aver effettuato il <i>fixed passthrough</i> della GPU Intel UHD 770 .	31
3.4	Messaggio di errore nel caricamento del ROM (Read-Only Memory) della GPU da parte del modulo <code>vfio-pci</code>	32
3.5	Comando utilizzato per attivare SR-IOV	34

ELENCO DEI LISTATI

Elenco delle tabelle

3.1	Caratteristiche della macchina PC-HP oggetto di studio	23
3.2	Caratteristiche della macchina secondaria PC-ACER, utilizzata come server per i benchmark di rete	24
3.3	Versioni dei sistemi operativi utilizzati	25
3.4	Versioni degli hypervisor utilizzati	25
3.5	Combinazioni di hypervisor e sistemi operativi utilizzati	25

ELENCO DELLE TABELLE

Capitolo 1

Introduzione

La virtualizzazione è una tecnologia che permette di eseguire contemporaneamente più sistemi operativi condividendo stesso hardware [1]. La sua origine risale agli anni '60, con il primo sistema di virtualizzazione CP-40 sviluppato da IBM. Questo sistema, originariamente definito come un sistema di partizione del tempo (*time-sharing system*), nacque con l'obiettivo di permettere, durante l'esecuzione di un programma, l'interazione dell'utente per modificare o eseguire un altro programma [2]. All'epoca, infatti, le macchine erano costose e venivano utilizzate per eseguire un singolo programma alla volta [3]. Oggi la virtualizzazione è utilizzata per diversi scopi, come ad esempio eseguire applicazioni destinate a un sistema operativo diverso¹, analizzare software malevolo in un ambiente isolato, migliorare la portabilità del software semplificando l'installazione delle dipendenze² oppure spartire le risorse in modo efficiente, in base al carico richiesto, tra più sistemi operativi utilizzati da più utenti o che ospitano servizi per scopi differenti³.

Le macchine virtuali sono repliche efficienti e isolate di una macchina reale [4], detta anche macchina bare metal, e sono eseguite da un programma chiamato

¹<https://web.archive.org/web/20250801042641/https://learn.microsoft.com/en-us/windows/wsl/>

²<http://archive.today/2025.08.02-103921/https://www.ibm.com/think/topics/containers>

³<https://web.archive.org/web/20250719035619/https://learn.microsoft.com/en-us/azure/virtual-machine-scale-sets/overview>

hypervisor. In una macchina virtuale tutte le componenti hardware, come CPU, memoria e dispositivi I/O (Input/Output) come disco, GPU, schede di rete, mouse, tastiera, sono virtualizzate. I sistemi operativi eseguiti all'interno delle macchine virtuali sono chiamati *guest* e il sistema operativo che esegue l'hypervisor è chiamato *host*. Per virtualizzare i dispositivi hardware esistono diverse tecniche: dalla più lenta emulazione software al *passthrough*, che permette alla macchina virtuale di accedere direttamente al dispositivo hardware fisico, con conseguenti prestazioni quasi alla pari di una macchina bare metal⁴.

1.1 Contesto e motivazioni

Questa tesi nasce nel contesto di un progetto volto ad automatizzare l'aggiornamento dei sistemi operativi, Windows e Linux, su una macchina utilizzata per la lavorazione industriale del legno. Per fare questo è utilizzata una macchina virtuale, a cui vengono assegnate il maggior numero di risorse possibili, con lo scopo di non intaccare l'utilizzo originale della macchina e nello stesso tempo rendendo gli aggiornamenti del sistema operativo trasparenti all'utente. In particolare, sulla macchina sono eseguite applicazioni che utilizzano le API (Application Programming Interface) grafiche DirectX e OpenGL, quindi è necessario verificare che quest'ultime possano essere eseguite all'interno della macchina virtuale.

Eseguire una macchina virtuale ha però un costo in termine di prestazioni, che dipende dalle tecniche di virtualizzazione utilizzate e dalla condivisione delle risorse tra un hypervisor e il sistema operativo virtualizzato. Inoltre, la configurazione di una macchina virtuale e in particolare di specifiche funzionalità può richiedere un hardware specifico. Ad esempio, per effettuare il *passthrough* di un dispositivo PCI (Peripheral Component Interconnect) con una CPU Intel è necessario che quest'ultima supporti la tecnologia Intel VT-d. Per finire, non tutti gli hypervisor supportano le stesse funzionalità. Si rende quindi necessario effettuare uno studio

⁴<https://web.archive.org/web/20250622112307/https://developer.ibm.com/tutorials/1-pci-passthrough/>.

sulle caratteristiche hardware della macchina, e un confronto di funzionalità e prestazioni tra diversi hypervisor. Come hypervisor, in questo lavoro sono stati scelti tre hypervisor open-source: VirtualBox, KVM e Xen.

Gli obiettivi posti da questa tesi sono:

- Configurare una macchina virtuale assegnando il maggior numero di risorse possibili, utilizzando gli hypervisor VirtualBox, KVM e Xen, sia con un sistema Windows che Linux;
- Verificare la possibilità di effettuare il passthrough della GPU, e la compatibilità con API grafiche OpenGL e DirectX;
- Confrontare le prestazioni di processore, memoria, disco, rete e GPU tra un sistema operativo bare metal e lo stesso virtualizzato.

1.2 Struttura della tesi

Questo documento è strutturato nel seguente modo:

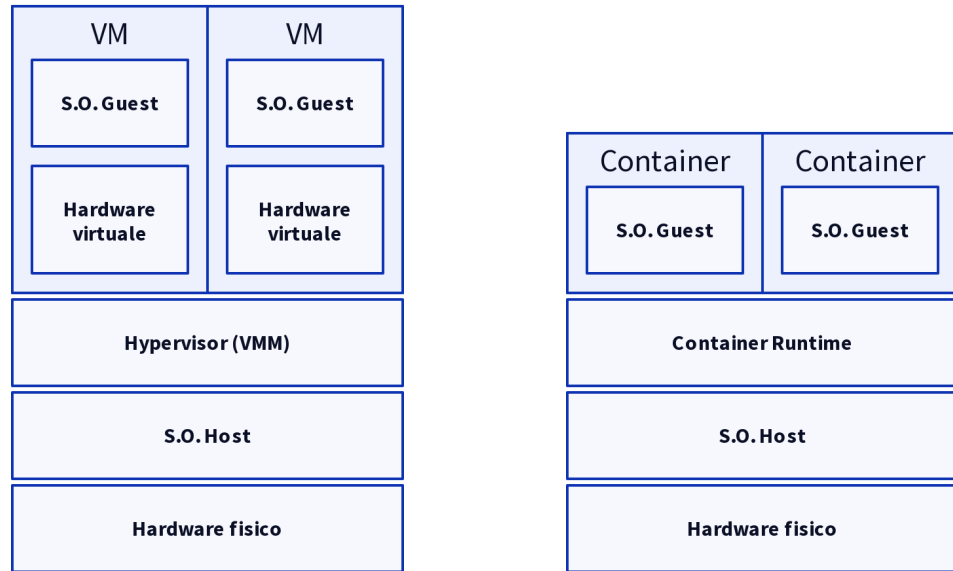
- Il capitolo 1 introduce il contesto e le motivazioni della tesi;
- Il capitolo 2 fornisce una panoramica sullo stato dell'arte della virtualizzazione, le tecnologie utilizzate in questo lavoro, e presenta alcuni studi precedenti sugli effetti della virtualizzazione sulle prestazioni;
- Il capitolo 3 descrive lo specifico caso di studio analizzato e come sono state configurate le macchine virtuali, con particolare attenzione al passthrough della GPU;
- Il capitolo 4 descrive la metodologia e gli strumenti utilizzati per effettuare i benchmark, e presenta i risultati;
- Il capitolo 5 conclude il documento, riassumendo i risultati e proponendo possibili sviluppi futuri.

Capitolo 2

Stato dell'arte: Virtualizzazione

2.1 Classificazione

La virtualizzazione è una tecnologia che permette di eseguire contemporaneamente più sistemi operativi condividendo lo stesso hardware [1]. Nella virtualizzazione si distinguono due categorie (Figura 2.1): la virtualizzazione a livello di sistema operativo, in cui è virtualizzato un sistema operativo che condivide lo stesso kernel del sistema host, e la virtualizzazione a livello di hardware, dove è invece l'intero hardware, inclusi CPU, memoria e dispositivi I/O.



(a) A livello di hardware

(b) A livello di sistema operativo

Figura 2.1: Tipi di virtualizzazione

2.1.1 Virtualizzazione a livello di sistema operativo

Nella virtualizzazione a livello di sistema operativo è virtualizzato un sistema operativo. Gli ambienti virtualizzati sono chiamati *container*, e ogni container può contenere un sistema operativo diverso, ma tutti condividono lo stesso kernel del sistema [1]. Questo significa che su un sistema Linux possono essere eseguiti diversi container, ognuno con il proprio *filesystem*¹ e contenenti diverse distribuzioni di

¹<https://web.archive.org/web/20180810111453/https://thijs.ai/papers/scheepers-virtualization-containerization.pdf>.

Linux (ad esempio Arch, Debian, Ubuntu), ma non un sistema operativo Windows.

Esempi di tecnologie di virtualizzazione a livello di sistema operativo sono Docker e LXC (Linux Containers).

2.1.2 Virtualizzazione a livello di hardware

Nella virtualizzazione a livello di hardware, a differenza di quella a livello di sistema operativo, è virtualizzato l'hardware. L'ambiente virtualizzato è chiamato macchina virtuale o VM (Virtual Machine), definita da Popek e Goldberg come una “replica efficiente e isolata di una macchina reale” [4].

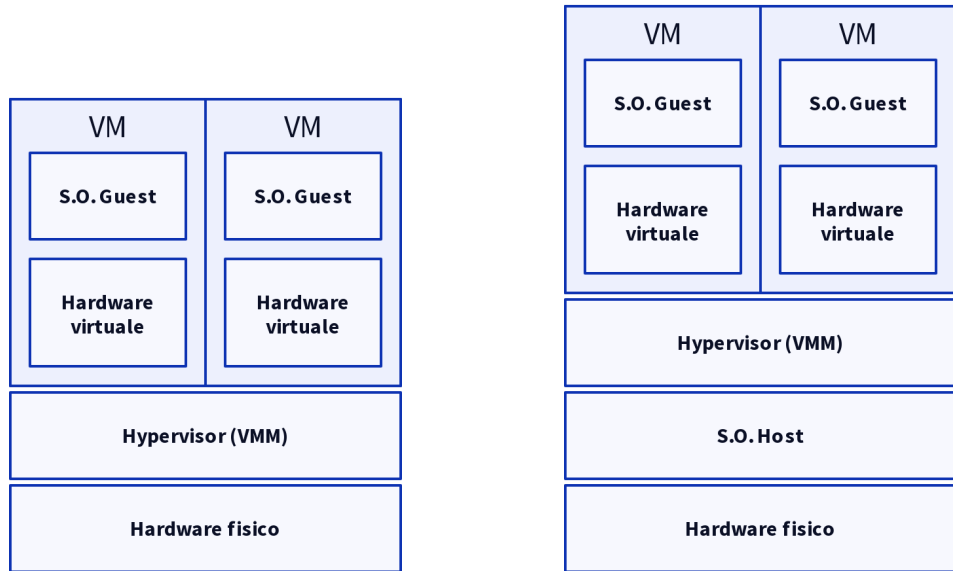
Le macchine virtuali sono eseguite da un software chiamato *hypervisor*, o *VMM* (*Virtual Machine Monitor*), definito da Katzan come “un programma di controllo che, insieme a una specifica funzionalità hardware, consente a due sistemi operativi di condividere un sistema di calcolo comune” [5].

Gli hypervisor si possono categorizzare in tipo-1 e tipo-2 (Figura 2.2). Gli hypervisor tipo-1 (o hypervisor *baremetal*) sono eseguiti direttamente dall'hardware, mentre quelli di tipo-2 (o hypervisor *hosted*) sono applicazioni eseguite su un sistema operativo, detto sistema (operativo) *host*². I sistemi operativi eseguiti nelle macchine virtuali sono chiamati sistemi (operativi) *guest*, o abbreviatamente *guest*. Alcuni esempi di hypervisor tipo-1 sono: VMware ESXi, Microsoft Hyper-V e Xen. Tra gli hypervisor tipo-2 rientrano invece: QEMU, VirtualBox, VMware Workstation.

Nella virtualizzazione a livello di hardware esistono tre tecniche principali di virtualizzazione: traduzione binaria (*binary translation*), paravirtualizzazione (*paravirtualization*) e virtualizzazione assistita dall'hardware (*hardware-assisted virtualization*) [1].

Nella virtualizzazione tramite traduzione binaria le istruzioni del kernel (privilegiate) del sistema guest sono interpretate dall'hypervisor a livello di software, mentre le istruzioni delle applicazioni sono eseguite direttamente dal processore. Il

²<https://web.archive.org/web/20240315201929/https://apps.dtic.mil/sti/citations/AD0772809>.



(a) Hypervisor tipo-1

(b) Hypervisor tipo-2

Figura 2.2: Tipi di hypervisor

sistema guest è completamente isolato dall'hypervisor e non richiede di sapere che è virtualizzato. Questa tecnica permette inoltre di eseguire sistemi di architetture diverse rispetto a quella del processore [1].

La paravirtualizzazione consiste nell'aggiunta di istruzioni virtuali dette *hypercall*, non presenti nell'architettura. Questo offre un vantaggio in termini di prestazioni ma richiede che il kernel del sistema guest sia modificato, in modo da poter comunicare con l'hypervisor tramite le *hypercall* [1].

La virtualizzazione assistita dall'hardware (*hardware-assisted virtualization*)

consiste nello sfruttare capacità dell'hardware per la virtualizzazione. Per architetture x86, AMD e Intel hanno sviluppato estensioni per i propri processori che aggiungono nuove istruzioni. La tecnologia di AMD è chiamata AMD-V (precedentemente SVM (Secure Virtual Machine)), mentre quella di Intel è chiamata VT-x o VMX (Virtual Machine Extensions). Queste estensioni hardware introducono un nuovo livello di privilegio, il ring -1, utilizzato dall'hypervisor, mentre il sistema operativo guest utilizza il ring 0 [1]. L'utilizzo dell'hardware permette di ottenere prestazioni migliori rispetto alla virtualizzazione tramite traduzione binaria.

2.2 Virtualizzazione di dispositivi I/O

In una macchina virtuale, oltre alla CPU e alla memoria, è necessario virtualizzare anche i dispositivi I/O come tastiera, mouse, dischi, schede di rete e GPU (Graphics Processing Unit). Per realizzare questo una tecnica comune è quella di emulare i dispositivi: questo può essere implementato direttamente all'interno dell'hypervisor o esternamente come processo separato³.

Le macchine virtuali non paravirtualizzate, dette anche virtualizzate completamente (*fully-virtualized*), hanno basse prestazioni I/O, ma queste possono essere migliorate tramite dispositivi virtuali progettati per la virtualizzazione, detti dispositivi paravirtualizzati⁴. Esempi di tecnologie che utilizzano questa tecnica sono i driver per la paravirtualizzazione di Xen⁵ e i driver Virtio (Virtual I/O Device)⁶, supportati da VirtualBox, QEMU e KVM.

Per ottenere prestazioni ancora migliori, simili a quelle di un sistema bare metal, esiste una tecnica che consente di assegnare il controllo completo di un dispo-

³<https://web.archive.org/web/20250622112307/https://developer.ibm.com/tutorials/l-pci-passthrough/>.

⁴<https://web.archive.org/web/20100215034356/https://www.kernel.org/doc/ols/2007/ols2007v1-pages-225-230.pdf>.

⁵<https://xenproject.org/projects/windows-pv-drivers/>.

⁶<https://web.archive.org/web/20250601030426/https://docs.oasis-open.org/virtio/virtio/v1.3/virtio-v1.3.html>

sitivo PCI fisico a una macchina virtuale, chiamata PCI passthrough, e tecnologie che espandono questa tecnica con la capacità di condividere lo stesso dispositivo tra più macchine virtuali, come SR-IOV (Single-Root Input/Output Virtualization).

2.2.1 PCI passthrough

Il PCI passthrough è una tecnica che permette l'accesso diretto di un dispositivo PCI fisico da parte di una macchina virtuale. Questo permette di ottenere prestazioni quasi alla pari del dispositivo utilizzato in una macchina bare metal, e non richiede driver paravirtualizzati.

Per effettuare il PCI passthrough è necessario che l'hardware sia in grado di tradurre gli indirizzi fisici del dispositivo PCI in indirizzi virtuali della macchina guest⁷. Il componente responsabile di questa funzione è chiamato IOMMU (Input-Output Memory Management Unit)⁸. Oltre a tradurre gli indirizzi, l'IOMMU permette di isolare i dispositivi, in modo che un dispositivo assegnato a una macchina virtuale non sia accessibile da un'altra o dal sistema host. Per architetture x86 esistono due specifiche di estensioni hardware che svolgono questo ruolo: AMD IOMMU⁹ per processori AMD e Intel VT-d¹⁰ per processori Intel.

2.2.2 SR-IOV

SR-IOV è uno standard realizzato dal PCI-SIG (PCI Special Interest Group) per la condivisione nativa di dispositivi I/O senza l'intermediazione di un hypervisor¹¹.

⁷<https://web.archive.org/web/20250622112307/https://developer.ibm.com/tutorials/l-pci-passthrough/>.

⁸<https://web.archive.org/web/20240518190130/https://kernel.org/doc/ols/2006/ols2006v1-pages-71-86.pdf>

⁹https://web.archive.org/web/20250502011102/https://www.amd.com/content/dam/amd/en/documents/processor-tech-docs/specifications/48882_IOMMU.pdf

¹⁰<https://web.archive.org/web/20250716104358/https://cdrdv2-public.intel.com/831418/vt-directe-d-io-spec.pdf>

¹¹<https://web.archive.org/web/20250119221522/https://www.intel.com/content/www/us/en/content-details/321211/pci-sig-sr-iov-primer-an-introduction-to-sr-iov-technology.html>.

In questo tipo di virtualizzazione il dispositivo fisico si presenta al sistema come due o più dispositivi PCI. Il dispositivo fisico è detto PF (Physical Function), mentre quelli virtuali sono chiamati VF (Virtual Function)¹². Il Listato 2.1 mostra un esempio di come viene presentato un dispositivo virtualizzato con SR-IOV, utilizzando il comando `lspci`, che elenca i dispositivi PCI su sistemi Linux.

```
1 0000:00:02.0 VGA compatible controller [0300]: Intel Corporation Alder Lake-S
   GT1 [UHD Graphics 770]
2 0000:00:02.1 VGA compatible controller [0300]: Intel Corporation Alder Lake-S
   GT1 [UHD Graphics 770]
```

Listato 2.1: Risultato del comando `lspci` con un VF attivato (riga 2) mediante SR-IOV, e il corrispondente PF (riga 1). I numeri che precedono il nome del tipo di dispositivo sono gli indici che identificano il dispositivo (*domain:bus:device.function*). Il VF possiede un indice *function* (.1) diverso rispetto al PF (.0).

2.2.3 Virtualizzazione GPU

Una GPU può essere virtualizzata con diverse tecniche: *emulazione*, *API remoting*, *fixed passthrough* e *mediated passthrough* [6]:

- Emulazione: è il caso in cui la GPU è emulata dal processore. Principalmente adatto per applicazioni grafiche 2D;
- API remoting: anche chiamato accelerazione 3D, è una tecnica con cui le chiamate alla API per la grafica (ad esempio OpenGL o DirectX) sono inviate al sistema host, che le esegue sulla GPU fisica (a cui ha accesso) e ne restituisce il risultato al sistema guest. Richiede l'installazione di un driver sul sistema guest che supporti la specifica API per poter reindirizzare

¹²<https://web.archive.org/web/20250717035040/https://www.kernel.org/doc/html/latest/PCI/pci-iov-howto.html>

le chiamate alle funzioni. Ne è un esempio VirGL¹³ che attualmente supporta OpenGL per guest Linux; il supporto per guest Windows è in fase di sviluppo¹⁴;

- Fixed passthrough: è un caso specifico di PCI passthrough (sottosezione 2.2.1) dove la macchina guest ha accesso diretto ed esclusivo alla GPU. Lo svantaggio è che la GPU non può essere condivisa né con l'host né con altri guest; per visualizzare lo schermo del sistema host è necessaria una seconda GPU, oltre che a un secondo schermo o a un commutatore (*switch KVM*, da non confondere con l'hypervisor KVM);
- Mediated passthrough: è un'estensione del fixed passthrough dove la GPU è condivisa tra più sistemi (ad esempio host e uno o più guest). Richiede una GPU che supporti una tecnologia per la condivisione, ad esempio NVIDIA vGPU o SR-IOV (sottosezione 2.2.2).

2.2.4 Virtualizzazione dei dispositivi di archiviazione

I dispositivi di archiviazione possono essere virtualizzati con diverse tecniche. La tecnica più comune è emulare emulando il controller del dispositivo. In questo caso, i dati del sistema guest possono essere memorizzati in un file immagine, salvato nel sistema host, oppure utilizzando una partizione di un disco fisico. A seconda dell'hypervisor utilizzato, esistono diversi formati che possono essere utilizzati per i file immagine. Una caratteristica comune di questi formati è la possibilità di essere creati occupando solo lo spazio necessario per i dati memorizzati (*dynamically allocated*), aumentando la dimensione del file quando viene utilizzato più spazio nel sistema guest, oppure di essere creati fin da subito con la dimensione massima

¹³<https://web.archive.org/web/20250521221235/https://docs.mesa3d.org/drivers/virgl.html>

¹⁴<https://github.com/virtio-win/kvm-guest-drivers-windows/pull/943>.

(*fixed-size*)¹⁵. Infine, un altro metodo è quello di effettuare il passthrough del dispositivo fisico¹⁶.

2.3 Tecnologie

2.3.1 VirtualBox

VirtualBox¹⁷ è un hypervisor tipo-2 open source, rilasciato per la prima volta nel 2007 e attualmente sviluppato da Oracle. Per architetture x86, come sistemi operativi host sono supportati i principali sistemi operativi (Windows, macOS e Linux) e Oracle Solaris, mentre come guest possono essere virtualizzati sistemi Windows, Linux e Oracle Solaris¹⁸. Per la virtualizzazione del disco, VirtualBox supporta la memorizzazione dei dati in file immagine di diversi formati, tra cui *vdi* (il formato di VirtualBox), *vmdk* (il formato di VMware) e *VHD* (il formato di Microsoft), oppure utilizzando partizioni di un disco fisico¹⁹.

In VirtualBox sono implementati diversi modelli di GPU virtuali²⁰:

- *VMSVGA* (VMware SVGA) per guest Linux;
- *VBoxVGA* per guest Windows precedenti a Windows 7 e Oracle Solaris;
- *VBoxSVGA* per guest Windows 7 o più recenti.

Sono inoltre incluse funzionalità aggiuntive come l'accelerazione 3D della grafica e la condivisione della *clipboard* tra host e guest tramite l'installazione nel

¹⁵<https://web.archive.org/web/20250712235209/https://docs.oracle.com/en/virtualization/virtualbox/7.1/user/storage.html>.

¹⁶[https://web.archive.org/web/20250819121413/https://pve.proxmox.com/wiki/Passthrough_Physical_Disk_to_Virtual_Machine_\(VM\)](https://web.archive.org/web/20250819121413/https://pve.proxmox.com/wiki/Passthrough_Physical_Disk_to_Virtual_Machine_(VM))

¹⁷<https://www.virtualbox.org/>

¹⁸<https://web.archive.org/web/20250210163108/https://www.virtualbox.org/manual/topics/Introduction.html#hostsupport>

¹⁹<https://web.archive.org/web/20250712235209/https://docs.oracle.com/en/virtualization/virtualbox/7.1/user/storage.html>.

²⁰<https://web.archive.org/web/20250122182935/https://www.virtualbox.org/manual/topics/BasicConcepts.html>

sistema guest dei Oracle VirtualBox Guest Additions²¹. Attualmente il PCI passthrough non è supportato; il precedente supporto sperimentale per host Linux tramite *VirtualBox Extension Pack* è stato rimosso a partire dalla versione 6.1.0²².

2.3.2 QEMU

QEMU è un emulatore open source, presentato per la prima volta nel 2005, che può essere utilizzato su sistemi operativi Linux, macOS e Windows²³.

QEMU possiede due modalità di funzionamento²⁴:

- **System Emulation:** permette di virtualizzare un intero sistema (svolgendo il ruolo di hypervisor tipo-2), oppure singoli dispositivi hardware, affiancando un altro hypervisor come KVM o Xen²⁵; oppure
- **User Mode Emulation:** permette di eseguire singole applicazioni destinate a un'architettura diversa rispetto a quella della CPU su cui viene eseguita (ad esempio un'applicazione per CPU ARM su una CPU x86).

Per architetture x86, QEMU permette di scegliere tra due diversi chipset di scheda madre da emulare: i440fx e Q35. Il chipset Q35 è il più recente tra i due e supporta funzionalità aggiuntive rispetto a i440fx, come la possibilità di emulare un bus PCIe o un IOMMU²⁶. È possibile configurare la CPU virtuale in diversi modalità: replicando lo stesso modello della CPU fisica, replicando un modello preimpostato o specificando manualmente il numero di *socket*, *core* e *thread* (processori logici). È possibile inoltre abilitare o disabilitare singole caratteristiche (*flag*) della CPU da mostrare al sistema virtualizzato, come ad esempio *vmx*, che

²¹<https://web.archive.org/web/20250714024326/https://www.virtualbox.org/manual/topics/guestadditions.html>

²²<https://web.archive.org/web/20250211183824/https://www.virtualbox.org/wiki/Changelog-6.1>

²³<https://www.qemu.org/download/>

²⁴https://web.archive.org/web/20060423091026/http://usenix.org/publications/library/proceedings/usenix05/tech/freenix/full_papers/bellard/bellard.pdf

²⁵<https://web.archive.org/web/20250708094231/https://www.qemu.org/docs/master/about/index.html>

²⁶<https://web.archive.org/web/20250427123144/https://wiki.qemu.org/images/4/4e/Q35.pdf>

indica il supporto alla tecnologia Intel VT-x²⁷. Tra i dispositivi che QEMU permette di emulare rientrano GPU, schede audio, schede di rete, dischi e controller USB²⁸. Per la virtualizzazione del disco, QEMU supporta diversi formati di file immagine, tra cui `qcow2` (il formato di QEMU), `vdi` (il formato di VirtualBox) e `vmdk` (il formato di VMware)²⁹. In particolare, utilizzando `qcow2` è possibile sfruttare funzionalità come la compressione e la cifratura. In QEMU sono implementati diversi modelli di GPU virtuali, tra cui in particolare il modello paravirtualizzato Virtio³⁰, che permette l'accelerazione 3D su guest Linux per applicazioni OpenGL (per guest Windows è in fase di sviluppo un driver per applicazioni OpenGL e DirectX 10³¹).

2.3.3 KVM

KVM (Kernel-based Virtual Machine) è un hypervisor presentato per la prima volta nel 2007³². Permette la virtualizzazione assistita dall'hardware su sistemi host Linux che supportano le estensioni hardware di virtualizzazione (Intel VT-x o AMD-V). KVM è diviso in due componenti: uno lato kernel, incluso in Linux dalla versione 2.6.20, e uno lato utente, integrato in QEMU a partire dalla versione 1.3³³. QEMU può essere inoltre utilizzato per la virtualizzazione di dispositivi I/O, ma per ottenere prestazioni migliori, KVM supporta anche dispositivi paravirtualizzati Virtio³⁴. Per utilizzare Virtio su guest Windows è necessario installare i driver Virtio³⁵ mentre Linux include già il supporto dalla versione del kernel 2.6.25 in

²⁷<https://web.archive.org/web/20241218124403/https://qemu.org/docs/master/system/i386/cpu.html>

²⁸<https://web.archive.org/web/20250619122324/https://qemu.readthedocs.io/en/v9.1.2/system/qemu-manpage.html>

²⁹<https://web.archive.org/web/20250716093807/https://www.qemu.org/docs/master/system/images.html>

³⁰<https://web.archive.org/web/20250619122402/https://qemu.readthedocs.io/en/v9.1.2/system/devices/virtio-gpu.html>

³¹<https://github.com/virtio-win/kvm-guest-drivers-windows/pull/943>.

³²<https://web.archive.org/web/20100215034356/https://www.kernel.org/doc/ols/2007/ols2007v1-pages-225-230.pdf>.

³³https://web.archive.org/web/20250727043459/https://www.linux-kvm.org/page/Main_Page

³⁴<https://web.archive.org/web/20250709041953/https://www.linux-kvm.org/page/Virtio>

³⁵https://www.linux-kvm.org/page/WindowsGuestDrivers/Download_Drivers

avanti³⁶. KVM supporta inoltre il PCI passthrough tramite Intel VT-d o AMD IOMMU³⁷.

2.3.4 Xen

Xen è un hypervisor tipo-1 open source, rilasciato per la prima volta nel 2003³⁸. In Xen le macchine virtuali sono chiamate domini (*domain*). Xen richiede un dominio che controlla l'hypervisor, chiamato *Dom0* o *Control Domain*. Gli altri domini sono invece chiamati *DomU* o *Guest Domain*. Il *Dom0* rispetto ai *DomU* ha accesso privilegiato all'hardware³⁹. Xen supporta sia la paravirtualizzazione (PV) che la virtualizzazione completa assistita dall'hardware (HVM). Le specifiche di Xen contengono un'ulteriore distinzione nella modalità HVM: HVM, HVM con driver PV e PVHVM. I guest HVM senza driver PV sono macchine virtuali completamente virtualizzate assistite dall'hardware. I guest HVM con driver PV sono guest HVM su cui sono installati driver per la paravirtualizzazione e hanno migliori prestazioni di disco e rete⁴⁰. I guest PVHVM sono guest HVM ma il cui sistema operativo supporta PVHVM (Linux a partire dalla versione 2.6.36 e FreeBSD dalla versione 10) e hanno migliori prestazioni rispetto ai guest HVM con driver PV. Oltre alle modalità PV e HVM, Xen prevede anche una terza modalità denominata PVH. La modalità PVH necessita di un sistema operativo modificato appositamente e non utilizza QEMU per virtualizzare dispositivi⁴¹. In Figura 2.3 è mostrato uno schema riassuntivo delle modalità di virtualizzazione di Xen. Per la virtualizzazione del disco e la memorizzazione dei dati, Xen supporta diversi formati di file, tra cui *qcow2*; in alternativa, è possibile utilizzare partizioni di un

³⁶<https://web.archive.org/web/20250624135506/https://wiki.libvirt.org/Virtio.html>

³⁷https://web.archive.org/web/20250716074415/https://www.linux-kvm.org/page/How_to_assign_devices_with_VT-d_in_KVM

³⁸<https://web.archive.org/web/20240831111538/https://lwn.net/Articles/52033/>

³⁹<https://web.archive.org/web/20250616204232/https://wiki.xenproject.org/wiki/Dom0>

⁴⁰https://web.archive.org/web/20250719131320/https://wiki.xenproject.org/wiki/PV_on_HVM

⁴¹https://web.archive.org/web/20250111002808/https://wiki.xenproject.org/wiki/Xen_Project_Software_Overview.

⁴³<https://web.archive.org/web/20250630200321/https://wiki.xenproject.org/wiki/Paravirtualization>

⁴⁶<https://libvirt.org/drivers.html>

gli stessi comandi, e un formato unificato dei file di configurazione delle macchine virtuali⁴⁷.

2.3.6 Virtual Machine Manager

Virtual Machine Manager, abbreviato `virt-manager`⁴⁸, è un'applicazione open-source per la creazione e la configurazione delle macchine virtuali tramite un'interfaccia grafica. All'interno, `virt-manager` utilizza `libvirt` per controllare gli hypervisor. Per creare una macchina virtuale, chiamata *domain* all'interno di `virt-manager`, è possibile utilizzare l'interfaccia grafica o un'applicazione a linea di comando, *virt-install*. L'interfaccia principale (Figura 2.4) permette di visualizzare le macchine virtuali, distinte per hypervisor. Aprendo una macchina virtuale viene aperta una seconda schermata, dove è possibile visualizzare lo schermo (Figura 2.6) e la configurazione della macchina virtuale (Figura 2.5). Per la visualizzazione dello schermo e l'interazione, `virt-manager` supporta SPICE⁴⁹ e VNC⁵⁰, due protocolli per il controllo remoto di un computer. In particolare, utilizzando un server SPICE e una GPU Virtio è possibile sfruttare l'accelerazione 3D (*API remoting*) per applicazioni OpenGL tramite il driver `VirGL` su guest Linux.

⁴⁷<https://libvirt.org/formatdomain.html>

⁴⁸<https://virt-manager.org/>

⁴⁹<https://spice-space.org/>

⁵⁰<https://web.archive.org/web/20250628044537/https://web.mit.edu/cdsdev/src/howitworks.html>

2.3. TECNOLOGIE

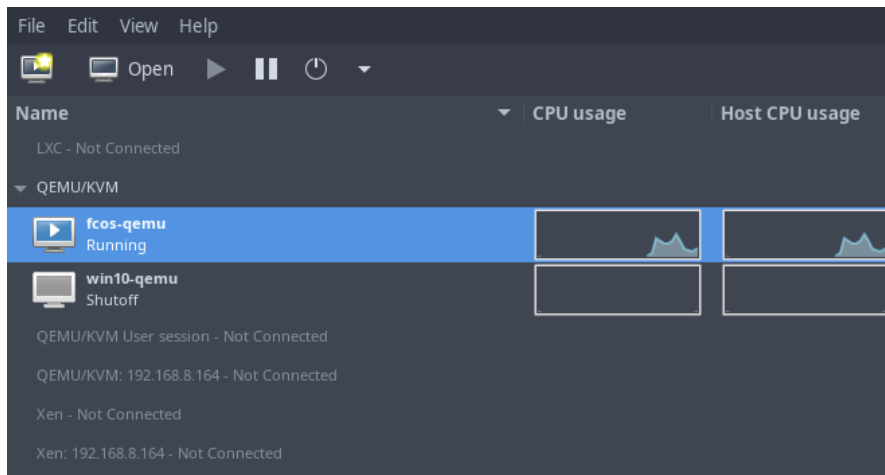


Figura 2.4: Finestra principale di virt-manager.

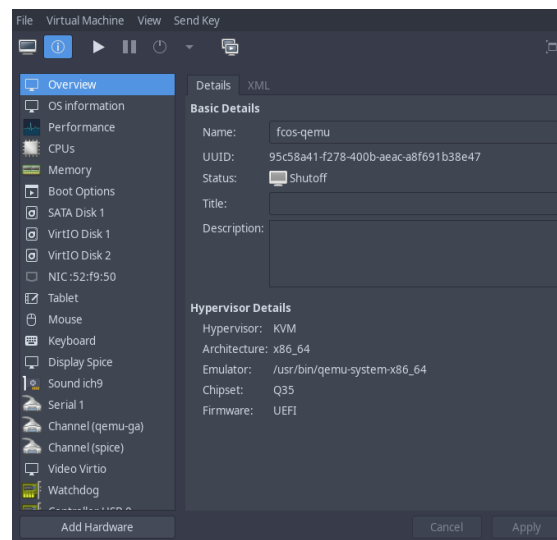


Figura 2.5: Visualizzazione delle impostazioni di una macchina virtuale in virt-manager.

2.3. TECNOLOGIE

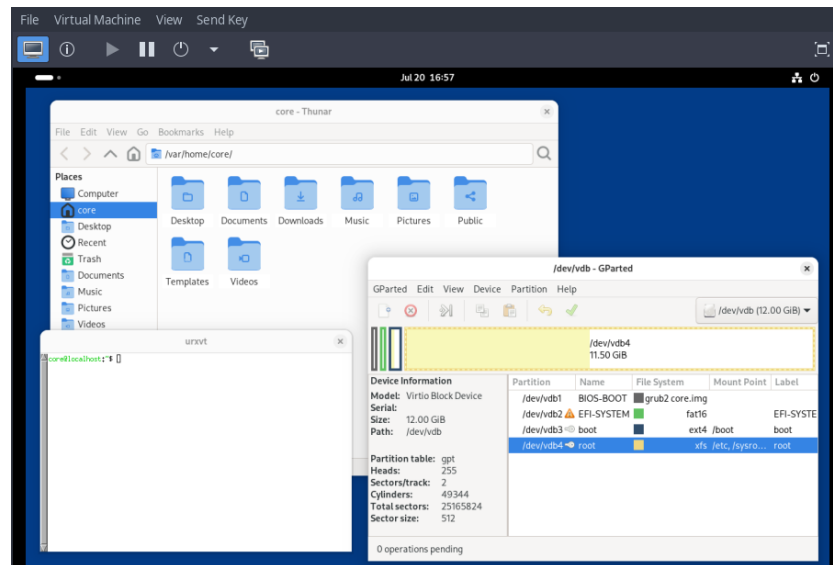


Figura 2.6: Visualizzazione dello schermo di una macchina virtuale in virt-manager.

2.4 Impatto sulle prestazioni: lavori precedenti

Il confronto delle prestazioni tra hypervisor e bare metal è stato già oggetto di studio in altri lavori. Sono riportati alcuni studi rilevanti che hanno utilizzato gli stessi hypervisor di questo elaborato, compresi casi di passthrough della GPU:

- Graziano [7] (2011) ha effettuato un confronto delle prestazioni tra Xen e KVM, effettuando benchmark di CPU, memoria e disco. Sono stati messe a confronto macchine virtuali con e senza paravirtualizzazione: Xen PV, Xen HVM, KVM con Virtio, KVM senza Virtio. Le macchine virtuali Xen PV hanno ottenuto prestazioni migliori rispetto a KVM in operazioni di disco, mentre KVM ha superato Xen in benchmark incentrati sulla CPU.
- Shea e Liu [8] (2013) hanno confrontato le prestazioni del GPU passthrough con hypervisor KVM e Xen rispetto al bare metal. I benchmark sono stati effettuati con applicazioni OpenGL (Doom3 e Unigine Sanctuary) e OpenCL (Luxmark). Nelle applicazioni OpenGL, le prestazioni con il GPU passthrough sono risultate inferiori rispetto al bare metal, mentre nell'applicazione OpenCL le prestazioni sono state identiche a quelle del bare metal.
- Walters et al. [9] (2014) hanno confrontato le prestazioni di GPU passthrough di GPU NVIDIA con hypervisor KVM, Xen e VMware ESXi. I benchmark sono stati effettuati mediante computazioni scientifiche utilizzando API CUDA e OpenCL. In particolare, dai risultati ottenuti è emerso che il GPU passthrough con KVM raggiunge prestazioni del 98-100% rispetto al sistema bare metal, mentre Xen e VMware ESXi raggiungono il 96-99% delle prestazioni rispetto al bare metal.
- Younge et al. [10] (2014) hanno confrontato le prestazioni di GPU passthrough con hypervisor Xen di due GPU dedicate NVIDIA Tesla. I benchmark sono stati effettuati effettuando computazioni matematiche utilizzando API

CUDA. I risultati hanno mostrato prestazioni simili a quelle di un sistema bare metal, e che il passthrough di GPU è adeguato per applicazioni scientifiche.

- Algarni et al. [11] (2018) hanno confrontato le prestazioni tra gli hypervisor KVM, Xen e Proxmox VE. È risultato che KVM è più adatto per applicazioni con carichi di lavoro di CPU e memoria, mentre Xen è più adatto per applicazioni con carichi di lavoro di disco.
- Wen et al. [12] (2023) hanno confrontato le prestazioni di container e macchine virtuali rispetto a un sistema bare metal. Come hypervisor è stato utilizzato KVM. Sono stati osservati cali di prestazioni tra lo 0% e il 5% rispetto al bare metal, in benchmark di CPU, memoria e rete, sia per container che per macchine virtuali. Significativo invece il calo di prestazioni nel benchmark di disco, dove KVM ha mostrato un calo di prestazioni del 35% rispetto al bare metal.

Capitolo 3

Caso di studio

L'obiettivo di questo elaborato è studiare la differenza in termini di prestazioni tra un sistema bare-metal e la sua controparte virtualizzata, considerando di voler fornire il maggior numero di risorse a un singolo sistema virtualizzato. In particolare, oggetto di questo studio è una macchina HP Pro Mini 400 G9 utilizzata in sistemi per la lavorazione industriale del legno.

3.1 Caratteristiche delle macchine utilizzate

Le caratteristiche della macchina oggetto di studio, denominata PC-HP, sono riportate in Tabella 3.1.

Modello	HP Pro Mini 400 G9
CPU	12th Gen Intel(R) Core(TM) i5-12500T
Caratteristiche CPU	4.4 GHz max, 6 core fisici, 12 core logici
RAM	16 GB 3200 MHz
SSD	SSSTC CL4-8D256-HP NVMe SSD M.2 256 GB
iGPU	Intel UHD Graphics 770 (Alder Lake-S 601 GT1)

Tabella 3.1: Caratteristiche della macchina PC-HP oggetto di studio

3.2. CONFIGURAZIONE DELLE MACCHINE VIRTUALI

Per la verifica delle prestazioni di rete è utilizzata una seconda macchina, denominata PC-ACER, le cui specifiche sono riportate in Tabella 3.2. Le due macchine sono collegate attraverso un router GL-AXT1800 e mediante cavi Ethernet 10/100/1000 Mbps (Megabit per second).

Modello	Acer Aspire E5-571G
CPU	Intel(R) Core(TM) i3-4030U
Caratteristiche CPU	1.9 GHz max, 2 core fisici, 4 core logici
RAM	4 GB 1600 MHz
HDD	WD5000LPVX-22V0TT0 500 GB
iGPU	Intel(R) HD Graphics 4400 (Intel Haswell-ULT)

Tabella 3.2: Caratteristiche della macchina secondaria PC-ACER, utilizzata come server per i benchmark di rete

Entrambe le macchine supportano sia la tecnologia Intel VT-x (per la virtualizzazione assistita dall'hardware) che Intel VT-d (per il PCI passthrough). La GPU integrata Intel UHD 770 della macchina PC-HP supporta la tecnologia SR-IOV¹, che può essere utilizzata per il *mediated passthrough* della GPU.

3.2 Configurazione delle macchine virtuali

In questo progetto sono valutati tre hypervisor open source e gratuitamente reperibili: VirtualBox, KVM e Xen.

Come sistema operativo host, o Dom0 nel caso di Xen, è utilizzata una distribuzione Linux Fedora Xfce, mentre per i sistemi operativi guest, o DomU, sono utilizzati Windows 10 e Fedora CoreOS, una distribuzione server immutabile di Linux. La scelta dei sistemi operativi guest è dettata dall'effettivo utilizzo di tali sistemi nell'azienda, mentre Fedora Xfce è scelto come sistema operativo host

¹<https://web.archive.org/web/20250117160020/https://www.intel.com/content/www/us/en/support/articles/000093216/graphics/processor-graphics.html>.

3.2. CONFIGURAZIONE DELLE MACCHINE VIRTUALI

data l'ampia disponibilità di pacchetti per la virtualizzazione tramite repository ufficiale, senza richiedere ad esempio di compilare Xen manualmente.

In Tabella 3.3 sono riportate le versioni dei sistemi operativi utilizzati, mentre in Tabella 3.4 sono riportate le versioni degli hypervisor. La Tabella 3.5 riassume tutte le combinazioni di hypervisor e sistemi operativi utilizzati.

Sistema Operativo	Versione
Fedora Xfce	Fedora Xfce 41
Fedora CoreOS	41.20241122.3.0
Windows 10	Win10 22H2 x64v1

Tabella 3.3: Versioni dei sistemi operativi utilizzati

Hypervisor	Versione	Versione pacchetto
VirtualBox	7.1.6	7.1.6_167084_fedora40-1
QEMU (KVM)	9.1.2	2:9.1.2-2.fc41
Xen	4.19.1	4.19.1-3.fc41

Tabella 3.4: Versioni degli hypervisor utilizzati

Bare metal / Hypervisor	S.O. Host / Dom0	S.O. Guest / DomU
Bare metal	Fedora CoreOS	///
Bare metal	Windows 10	///
VirtualBox	Fedora Xfce	Fedora CoreOS
VirtualBox	Fedora Xfce	Windows 10
KVM	Fedora Xfce	Fedora CoreOS
KVM	Fedora Xfce	Windows 10
Xen	Fedora Xfce	Fedora CoreOS
Xen	Fedora Xfce	Windows 10

Tabella 3.5: Combinazioni di hypervisor e sistemi operativi utilizzati

In una macchina virtuale è possibile configurare varie caratteristiche tra cui il numero di processori, la quantità di memoria disponibile, assegnare dischi virtuali o altri dispositivi come GPU e schede di rete.

Il numero di processori virtuali (vCPU) da assegnare alla macchina virtuale è scelto considerando il carico di lavoro richiesto e le caratteristiche del processore fisico (pCPU). Aumentando il numero di processori virtuali oltre a una certa soglia si potrebbero avere effetti di sovra-allocazione con conseguenti perdite di prestazioni. Rahman et al. [13] hanno verificato sperimentalmente che questa soglia combacia con il numero di processori logici fisici. In tutte le macchine virtuali il numero di processori virtuali è quindi impostato al numero di processori logici fisici. Alle macchine virtuali sono assegnate 13312 MiB (Mebibyte) (circa 14 GB (Gigabyte)) di memoria. Questo numero è scelto in modo che l'host e l'hypervisor abbiano abbastanza memoria a loro disposizione, per evitare di incorrere nell'uso dello swap.

I file di configurazione delle macchine virtuali e gli script per l'installazione di script e creazione delle macchine virtuali sono reperibili all'indirizzo: <https://github.com/smagnaniuni/vmm-bench-2025>. All'interno sono inoltre presenti i file di configurazione delle macchine virtuali. Si sottolinea che prima di utilizzare questi file è necessario creare i dischi virtuali, e in alcuni casi anche effettuare delle modifiche al file. Ad esempio, VirtualBox include nel file di configurazione gli identificatori unici (UUID) del disco, che vanno quindi sostituiti con quelli effettivi.

3.2.1 VirtualBox

Seppur sia possibile utilizzare virt-manager per comandare VirtualBox, quest'ultimo include già al suo interno un'interfaccia grafica ufficiale per configurare e utilizzare le macchine virtuali. Per questo motivo sono utilizzate le applicazioni incluse in VirtualBox: la configurazione delle macchine virtuali è effettuata tramite l'applicazione a linea di comando (**VBoxManage**) mentre per l'interazione e la visualizzazione dello schermo è utilizzata l'interfaccia grafica.

L'impostazione del firmware è lasciata al valore predefinito utilizzato da VirtualBox: UEFI (Unified Extensible Firmware Interface) per Fedora CoreOS e BIOS

(Basic Input/Output System) per Windows 10. Come CPU sono impostati 12 processori virtuali, corrispondenti al numero di processori logici del processore fisico. VirtualBox non permette di configurare la topologia della CPU virtuale. Alla macchina virtuale sono assegnati 13312 MiB di memoria. Come disco per il sistema operativo è utilizzato il modello con bus SATA (Serial AT Attachment), mentre per i benchmark sono aggiunti altri due dischi emulati, uno di tipo SATA e uno Virtio (paravirtualizzato). Per utilizzare il disco con bus tipo Virtio su sistema operativo Windows 10 è necessario installare i driver guest Virtio per la paravirtualizzazione, mentre per Fedora CoreOS questo è già supportato. Per quest'ultimo è utilizzato un file di formato `vmdk` creato automaticamente dall'archivio di installazione di Fedora CoreOS per hypervisor VirtualBox. Per Windows 10 è utilizzato un file di formato `vdi`.

Come precedentemente menzionato in sottosezione 2.3.1 VirtualBox supporta l'accelerazione 3D, tuttavia attivando quest'ultima la macchina virtuale non si avvia, riportando l'errore in Listato 3.1.

```
1 VMSVGA: unsupported SVGA3D_DEVCAP
```

Listato 3.1: Errore riscontrato utilizzando l'accelerazione 3D in VirtualBox.

Il problema risulta essere una regressione di VirtualBox per sistemi host Linux, introdotta a partire dalla versione 7.1.0². Con VirtualBox è quindi possibile utilizzare solamente la virtualizzazione tramite emulazione: sono utilizzati i modelli `VMSVGA` per Fedora CoreOS e `VBoxSVGA` per Windows 10.

Come scheda di rete sono utilizzati i modelli predefiniti: `virtio` per Fedora CoreOS e `Intel 82540EM` per Windows 10.

²<https://web.archive.org/web/20250712235532/https://forums.virtualbox.org/viewtopic.php?t=1123>
73

3.2.2 KVM

Per l'interazione con le macchine virtuali con hypervisor KVM è utilizzato virt-manager, mentre per la creazione è utilizzata l'applicazione a linea di comando virt-install. Le macchine virtuali sono configurate per la virtualizzazione completa (HVM). Per entrambi i sistemi operativi, come firmware è utilizzato UEFI (implementazione OVMF³) e come chipset Q35. Come CPU è replicato lo stesso modello e la stessa topologia (stesso numero di processori fisici e logici) della CPU fisica. Alla macchina virtuale sono assegnati 13312 MiB di memoria. Come disco per il sistema operativo è utilizzato il modello con bus SATA. Per i benchmark sono aggiunti altri due dischi emulati, uno di tipo SATA e uno Virtio (paravirtualizzato). Per utilizzare il disco con bus tipo Virtio su sistema operativo Windows 10 è necessario installare i driver guest Virtio per la paravirtualizzazione, mentre per Fedora CoreOS questo è già supportato. Come disco virtuale del sistema operativo è utilizzato un file di formato qcow2. Come GPU è utilizzato il modello Virtio sia su Fedora CoreOS che su Windows 10; con guest Fedora CoreOS questo permette inoltre di utilizzare l'accelerazione 3D tramite driver VirGL e server SPICE. È inoltre utilizzato il passthrough mediante SR-IOV. Per quest'ultimo il procedimento è lo stesso utilizzato per Xen ed è descritto in sottosezione 3.3.2. Come scheda di rete sono utilizzati i modelli predefiniti: virtio per Fedora CoreOS e e1000e per Windows 10.

3.2.3 Xen

Per l'interazione con le macchine virtuali con hypervisor Xen è utilizzato virt-manager, mentre per la creazione è utilizzata l'applicazione a linea di comando virt-install. Le macchine virtuali sono configurate nella modalità HVM. Sia per Fedora CoreOS che per Windows 10 è utilizzato il firmware BIOS predefinito di Xen (hvmloader). Come CPU è replicato lo stesso modello e la stessa topologia

³<https://github.com/tianocore/tianocore.github.io/wiki/OVMF>

(stesso numero di processori fisici e logici) della CPU fisica. Alla macchina virtuale sono assegnati 13312 MiB di memoria. Come disco per il sistema operativo è utilizzato il modello con bus IDE (Integrated Drive Electronics). Per i benchmark sono aggiunti altri due dischi, uno IDE e uno Xen (paravirtualizzato). Per utilizzare il disco Xen su sistema operativo Windows 10 è necessario installare i driver per la paravirtualizzazione⁴, mentre su Fedora CoreOS questo è già supportato. Utilizzando il disco paravirtualizzato, secondo la classificazione di Xen (sottosezione 2.3.4) la macchina virtuale Windows 10 è quindi classificata come HVM con driver PV mentre quella Fedora CoreOS come PVHVM. Come disco virtuale del sistema operativo è utilizzato un file di formato qcow2. Come GPU è utilizzato il modello Xen (GPU emulata) il passthrough tramite SR-IOV. Per quest'ultimo il procedimento è lo stesso utilizzato per KVM ed è descritto in sottosezione 3.3.2. Come scheda di rete è utilizzato il modello predefinito: e1000 sia per Fedora CoreOS che per Windows 10.

3.3 Virtualizzazione GPU

Per la virtualizzazione della GPU con KVM e Xen, oltre all'emulazione, sono utilizzate due ulteriori tecniche: il *fixed passthrough* e il *mediated passthrough* tramite SR-IOV.

3.3.1 Fixed Passthrough

Per le GPU Intel, o IGD (Intel Graphics Device), si distinguono due modalità di passthrough:⁵⁶

⁴<https://xenproject.org/projects/windows-pv-drivers/>.

⁵https://web.archive.org/web/20250607214334/https://github.com/intel/gvt-linux/wiki/GVTd_Setup_Guide.

⁶<https://archive.today/2025.07.18-154249/https://gitlab.com/qemu-project/qemu/-/blob/0e3aff9ec34059512d597eacfcf4d1b5d4570c50/docs/igd-assign.txt>.

3.3. VIRTUALIZZAZIONE GPU

- UPT (Universal Pass-Through), in cui la GPU viene utilizzata come GPU secondaria per accelerazione hardware ma senza avere uscita a schermo, richiedendo quindi un'altra GPU; oppure
- Legacy, in cui la GPU viene utilizzata come GPU primaria, con possibilità di avere uscita a schermo.

Per lo scopo richiesto, cioè utilizzare la macchina virtuale sulla stessa macchina fisica su cui è eseguita, e data la presenza di una sola GPU (vedere Tabella 3.1), il *fixed passthrough* in modalità UPT è da escludere, in quanto non sarebbe visualizzato nulla a schermo.

Verificando la suddivisione dei dispositivi PCI in gruppi IOMMU⁷ la GPU Intel UHD 770 risulta essere l'unico dispositivo in un gruppo (Listato 3.2). Questo significa che è possibile effettuare il passthrough della singola GPU senza effettuare il passthrough di altri dispositivi. Nel caso in cui la GPU si trovasse in un gruppo IOMMU assieme a un altro dispositivo, ad esempio un controller USB, sarebbe necessario effettuare il passthrough di entrambi. I dispositivi in un gruppo IOMMU sono infatti isolati dai dispositivi in altri gruppi⁸.

```
1 IOMMU Group 0:
2   00:02.0 VGA compatible controller [0300]: Intel Corporation Alder Lake-S GT1 [
   UHD Graphics 770] [8086:4690] (rev 0c)
3 IOMMU Group 1:
4   00:00.0 Host bridge [0600]: Intel Corporation 12th Gen Core Processor Host
   Bridge [8086:4650] (rev 05)
5 IOMMU Group 2:
6   00:14.0 USB controller [0c03]: Intel Corporation Alder Lake-S PCH USB 3.2 Gen 2
   x2 XHCI Controller [8086:7ae0] (rev 11)
7   00:14.2 RAM memory [0500]: Intel Corporation Alder Lake-S PCH Shared SRAM
   [8086:7aa7] (rev 11)
8 IOMMU Group 3:
9   00:16.0 Communication controller [0780]: Intel Corporation Alder Lake-S PCH HECI
   Controller #1 [8086:7ae8] (rev 11)
```

⁷https://web.archive.org/web/20250716175539/https://wiki.archlinux.org/title/PCI_passthrough_via_OVMF#Ensuring_that_the_groups_are_valid

⁸<https://web.archive.org/web/20250603203543/https://www.kernel.org/doc/html/latest/driver-api/vfio.html>.

3.3. VIRTUALIZZAZIONE GPU

```
10 | 00:16.3 Serial controller [0700]: Intel Corporation Alder Lake-S Keyboard and
    | Text (KT) Redirection [8086:7aeb] (rev 11)
11 | IOMMU Group 4:
12 | 00:17.0 SATA controller [0106]: Intel Corporation Alder Lake-S PCH SATA
    | Controller [AHCI Mode] [8086:7ae2] (rev 11)
13 | IOMMU Group 5:
14 | 00:1a.0 PCI bridge [0604]: Intel Corporation Alder Lake-S PCH PCI Express Root
    | Port #25 [8086:7ac8] (rev 11)
15 | IOMMU Group 6:
16 | 00:1f.0 ISA bridge [0601]: Intel Corporation Device [8086:7a83] (rev 11)
17 | 00:1f.3 Audio device [0403]: Intel Corporation Alder Lake-S HD Audio Controller
    | [8086:7ad0] (rev 11)
18 | 00:1f.4 SMBus [0c05]: Intel Corporation Alder Lake-S PCH SMBus Controller
    | [8086:7aa3] (rev 11)
19 | 00:1f.5 Serial bus controller [0c80]: Intel Corporation Alder Lake-S PCH SPI
    | Controller [8086:7aa4] (rev 11)
20 | 00:1f.6 Ethernet controller [0200]: Intel Corporation Ethernet Connection (17)
    | I219-LM [8086:1a1c] (rev 11)
21 | IOMMU Group 7:
22 | 01:00.0 Non-Volatile memory controller [0108]: Solid State Storage Technology
    | Corporation CL4-8D512 NVMe SSD M.2 (DRAM-less) [1e95:1007] (rev 03)
```

Listato 3.2: Dispositivi PCI della macchina PC-HP, suddivisi per gruppi IOMMU

Dalle verifiche effettuate non risulta possibile effettuare il passthrough “Legacy” della GPU integrata Intel né con sistema guest Linux né Windows. Utilizzando il comando `lspci` tramite collegamento SSH al sistema guest Linux, la GPU è riconosciuta e il modulo Linux per GPU Intel i915 è caricato, tuttavia non si ha segnale a schermo. Il log di sistema (`journalctl -b`) riporta in particolare che non è stato possibile trovare le tabelle del VBIOS (Video BIOS). Il VBIOS è il BIOS di una GPU, ed è necessario per la sua inizializzazione⁹.

```
1 | [core@localhost ~]$ sudo dmesg | grep i915
2 | [ 5.919883] i915 0000:07:00.0: [drm] Found ALDERLAKE_S (device ID 4690) display
    | version 12.00
3 | [ 5.920873] i915 0000:07:00.0: [drm] VT-d active for gfx access
4 | [ 5.920878] i915 0000:07:00.0: vgaarb: deactivate vga console
5 | [ 5.920941] i915 0000:07:00.0: [drm] Using Transparent Hugepages
```

⁹https://web.archive.org/web/20131124214626/https://www.x.org/wiki/Events/XDC2013/XDC2013MartinPeresNvidiaPM/xdc2013-nvidia_pm.pdf.

3.3. VIRTUALIZZAZIONE GPU

```
6 [ 5.921613] i915 0000:07:00.0: ROM [??? 0x00000000 flags 0x20000000]: can't
  assign; bogus alignment
7 [ 5.921619] i915 0000:07:00.0: [drm] Failed to find VBIOS tables (VBT)
8 [ 5.922322] i915 0000:07:00.0: vgaarb: VGA decodes changed: olddecodes=io+mem,
  decodes=io+mem:owns=none
9 [ 5.942207] i915 0000:07:00.0: [drm] Finished loading DMC firmware i915/
  adls_dmc_ver2_01.bin (v2.1)
10 [ 7.852427] i915 0000:07:00.0: [drm] [ENCODER:240:DDI A/PHY A] failed to
  retrieve link info, disabling eDP
11 [ 7.933357] i915 0000:07:00.0: [drm] GT0: GuC firmware i915/tgl_guc_70.bin
  version 70.29.2
12 [ 7.933362] i915 0000:07:00.0: [drm] GT0: HuC firmware i915/tgl_huc.bin version
  7.9.3
13 [ 7.941447] i915 0000:07:00.0: [drm] GT0: HuC: authenticated for all workloads
14 [ 7.941449] i915 0000:07:00.0: [drm] GT0: GUC: submission disabled
15 [ 7.941450] i915 0000:07:00.0: [drm] GT0: GUC: SLPC disabled
16 [ 7.942411] i915 0000:07:00.0: [drm] Protected Xe Path (PXP) protected content
  support initialized
17 [ 7.944162] [drm] Initialized i915 1.6.0 for 0000:07:00.0 on minor 0
18 [ 7.953892] i915 0000:07:00.0: [drm] Cannot find any crtc or sizes
19 [ 7.973715] i915 0000:07:00.0: [drm] Cannot find any crtc or sizes
20 [ 7.983766] i915 0000:07:00.0: [drm] Cannot find any crtc or sizes
```

Listato 3.3: Messaggi relativi al modulo i915 nel log del sistema guest Linux, dopo aver effettuato il *fixed passthrough* della GPU Intel UHD 770

Nel log del sistema host Linux il modulo `vfio-pci`¹⁰, utilizzato per effettuare il PCI passthrough, segnala che il VBIOS della GPU estratto non è valido (Listato 3.4). All'avvio della macchina virtuale, libvirt disconnette la GPU dal sistema host e la connette al guest, caricando il modulo `vfio-pci`. Per poter inizializzare la GPU, il modulo `vfio-pci` tenta automaticamente di estrarre il VBIOS, tuttavia questo può anche essere fornito manualmente se già estratto.

```
1 vfio-pci 0000:00:02.0: Invalid PCI ROM header signature: expecting 0xaa55, got 0
  xffff
```

Listato 3.4: Messaggio di errore nel caricamento del ROM della GPU da parte del modulo `vfio-pci`

¹⁰<https://web.archive.org/web/20250603203543/https://www.kernel.org/doc/html/latest/driver-api/vfio.html>.

Dalle verifiche effettuate non risulta possibile estrarre il VBIOS sia su sistema Linux mediante comandi a terminale¹¹ che su sistema Windows mediante l'applicazione GPU-Z¹². Una risposta¹³ su **StackExchange** riporta la necessità di effettuare l'avvio del sistema operativo (host) in modalità **Legacy** per estrarre il VBIOS, tuttavia la macchina non supporta questa modalità¹⁴.

La necessità di utilizzare la modalità d'avvio **Legacy** è confermata con la macchina secondaria PC-ACER, che consente di utilizzarla, e anch'essa contenente una GPU integrata Intel. Utilizzando la modalità d'avvio **Legacy** è possibile estrarre il VBIOS, mentre in modalità UEFI si ripresentano gli stessi errori riscontrati con la macchina PC-HP. Cambiare la modalità d'avvio richiede la reinstallazione del sistema operativo; per evitare ciò è possibile cambiare la modalità d'avvio in **Legacy**, avviare la macchina con un'immagine di installazione "live" di Linux¹⁵, estrarre il VBIOS e riavviare la macchina in modalità UEFI. Utilizzando il VBIOS estratto manualmente è possibile effettuare il passthrough "Legacy", ma solo se il sistema guest utilizza la modalità d'avvio Legacy/BIOS (implementazione **SeaBIOS**). Sebbene il passthrough "Legacy" sia supportato anche con l'implementazione del firmware UEFI OVMF¹⁶, nelle verifiche effettuate non è visualizzato nulla a schermo. Infine, il passthrough "Legacy" è utilizzato con successo solo con il chipset **i440fx**, mentre con **Q35** lo schermo lampeggia ripetutamente.

¹¹<https://web.archive.org/web/20240528051053/https://www.intel.com/content/www/us/en/docs/graphics-for-linux/developer-reference/1-0/dump-video-bios.html>.

¹²<https://www.techpowerup.com/gpuz/>

¹³<https://web.archive.org/web/20250119182932/https://unix.stackexchange.com/questions/729711/how-to-dump-video-bios-of-mesa-intel-uhd-graphics-620-kbl-gt2>.

¹⁴<https://web.archive.org/web/20250119233516/https://www.intel.com/content/www/us/en/content-details/630266/removal-of-legacy-boot-support-for-intel-platforms-technical-advisory.html>.

¹⁵Un'immagine "live" permette di avviare un sistema operativo direttamente da un dispositivo rimovibile, senza la necessità di installarlo.

¹⁶<https://web.archive.org/web/20250702222059/https://github.com/intel/gvt-linux/issues/23>.

3.3.2 Mediated Passthrough tramite SR-IOV

La GPU integrata Intel UHD 770 della macchina PC-HP supporta la tecnologia SR-IOV¹⁷, tuttavia il driver grafico per GPU Intel i915 nel kernel Linux *upstream* (originale) non ha abilitato il supporto a SR-IOV¹⁸, mentre lo è in un kernel *out-of-tree* (cioè in un *fork*)¹⁹. Il modulo nel kernel *out-of-tree* di Intel²⁰ supporta solo alcune distribuzioni Linux, principalmente Enterprise, quindi è utilizzato il pacchetto `i915-sriov-dkms`²¹ dell'utente `strongtz` su Github, che ne supporta un maggior numero. Entrambi utilizzano DKMS (Dynamic Kernel Module Support) per modificare il singolo modulo senza modificare l'intero kernel²².

Per utilizzare SR-IOV è necessario impostare il numero massimo di istanze virtuali della GPU tramite il parametro `max_vfs` nella configurazione del *bootloader*. Per attivare un VF della GPU è necessario utilizzare inviare a terminale il comando in Listato 3.5.

```
1 echo 1 > /sys/bus/pci/devices/0000:00:02.0/sriov_numvfs
```

Listato 3.5: Comando utilizzato per attivare SR-IOV, dove 1 rappresenta il numero di VF da attivare e `0000:00:02.0` è l'identificatore completo del PF.

Per poter utilizzare la GPU nei sistemi guest la procedura varia a seconda del sistema operativo. Nei sistemi guest Fedora CoreOS è necessario installare lo stesso modulo utilizzato per attivare SR-IOV nel sistema host. Per i sistemi Windows 10 invece è necessario installare i driver grafici ufficiali Intel, forniti da HP per la specifica macchina, e impostare nella configurazione libvirt delle

¹⁷<https://web.archive.org/web/20250117160020/https://www.intel.com/content/www/us/en/support/articles/000093216/graphics/processor-graphics.html>.

¹⁸<https://web.archive.org/web/20250630103602/https://github.com/intel/linux-intel-lts/issues/33>

¹⁹<https://web.archive.org/web/20250119174303/https://dgpu-docs.intel.com/driver/kernel-driver-types.html>.

²⁰<https://github.com/intel-gpu/intel-gpu-i915-backports/tree/backport/main>

²¹<https://github.com/strongtz/i915-sriov-dkms>

²²<https://web.archive.org/web/20250119181530/https://www.kernel.org/doc/ols/2004/ols2004v1-pages-187-202.pdf>.

macchine virtuali il parametro `vendor_id` a `GenuineIntel`²³. Senza quest'ultimo parametro la GPU viene riconosciuta ma il driver restituisce il codice di errore 43, che indica un problema nel funzionamento del dispositivo²⁴.

Anche se la GPU è in grado di essere utilizzata dal sistema guest per applicazioni 3D o computazione, i server SPICE e VNC utilizzati con virt-manager non riescono a sfruttarla per visualizzare lo schermo. Per sistema Linux è quindi utilizzato un altro server VNC, TigerVNC, che non richiede ulteriori GPU per visualizzare il desktop. TigerVNC risulta adeguato su sistema Linux ma non su sistema Windows: l'aggiornamento dello schermo risulta molto lento e in alcuni casi è assente (ad esempio quando si scorre una finestra). Per la visualizzazione dello schermo su Windows sono utilizzati due diversi tipi di server per il controllo remoto a seconda dell'API grafica necessaria: con applicazioni DirectX è utilizzato il server VNC di virt-manager, mentre con applicazioni OpenGL è utilizzato il server RDP (Remote Desktop) di Windows. Con server VNC è necessario aggiungere una GPU emulata; la scelta è ricaduta sul modello Virtio con hypervisor KVM e modello Xen con hypervisor Xen (identificate all'interno di Windows come `Red Hat VirtIO GPU DOD controller` e `Microsoft Basic Display Adapter` rispettivamente con GPU VirtIO e Xen). Per il server RDP questo non è necessario in quanto include già una GPU emulata interna (identificata come `Microsoft Remote Display Adapter`). Le applicazioni DirectX si avviano con server VNC ma non con server RDP, e viceversa per quelle OpenGL. Dato che le GPU sono identificate diversamente in Windows nei due casi, si sospetta che questo sia dovuto dall'incompatibilità con le API grafiche a seconda del driver utilizzato.

Per connettersi ai server TigerVNC e RDP è utilizzato come client il software Remmina²⁵.

²³https://web.archive.org/web/20250205192240/https://wiki.archlinux.org/title/QEMU/Guest_graphics_acceleration

²⁴<https://web.archive.org/web/20250526115004/https://support.microsoft.com/en-us/windows/fix-graphics-device-problems-with-error-code-43-6f6ae1ec-0bbe-a848-142e-0c6190502842>

²⁵<https://remmina.org/>

Capitolo 4

Confronto delle prestazioni

4.1 Strumenti

Per effettuare i benchmark sono utilizzati sia strumenti sintetici, applicazioni che creano artificialmente carichi di lavoro, che reali, cioè basati su reali applicazioni, come ad esempio un programma di archiviazione dati¹.

Per effettuare i benchmark sono utilizzati i seguenti strumenti:

- **iperf3**:² software per la valutazione della velocità di banda di rete, sia con protocollo TCP che UDP;
- **Phoronix Test Suite (PTS)**:³ software per l'installazione e l'esecuzione automatica di strumenti di benchmark. All'interno sono inoltre presenti altre funzionalità a supporto, come la generazione di documenti riassuntivi e grafici dei risultati.

Con PTS sono utilizzati i seguenti benchmark:

¹<https://web.archive.org/web/20250613011348/https://www.intel.com/content/www/us/en/gaming/resources/read-cpu-benchmarks.html>

²<https://software.es.net/iperf/>

³<https://www.phoronix-test-suite.com/>

- **pts/compress-7zip**: utilizza **7zip**⁴, un software per la compressione e la decompressione di dati, per valutare le prestazioni della CPU;
 - **pts/hashcat**: utilizza **hashcat**⁵, un software di recupero password. Da un hash iniziale, **hashcat** tenta di recuperare la password con cui è stato calcolato (*password cracking*). Supporta diversi algoritmi crittografici di hash, tra cui MD5 e SHA1, e diverse API per la computazione, tra le quali OpenCL e CUDA. Contiene una funzionalità di benchmark che genera un hash casuale e misura la velocità di cracking⁶;
 - **pts/unigine-heaven**: utilizza **Unigine Heaven**⁷, un software per la valutazione delle prestazioni della GPU mediante il rendering di una scena 3D animata. Oltre a OpenGL supporta anche l'utilizzo di API DirectX su sistemi Windows;
 - **pts/vkmark**: utilizza **vkmark**, un software per la valutazione delle prestazioni della GPU effettuando il rendering di una scena 3D animata utilizzando API Vulkan.
- **PassMark PerformanceTest**:⁸ software di benchmark disponibile per diversi sistemi operativi, tra cui Linux e Windows. Nella versione per Linux permette di valutare le prestazioni di CPU e memoria, mentre nella versione per Windows permette di valutare anche quelle di disco e GPU. Le prestazioni della CPU sono valutate effettuando calcoli di matematica intera e floating point, ricerca di numeri primi, compressione dei dati, e utilizzo di estensioni hardware SIMD e per la crittografia. Le prestazioni della memoria sono valutate mediante operazioni su database, operazioni di lettura e scrittura, memoria disponibile e latenza. Per la GPU sono presenti benchmark per

⁴<https://www.7-zip.org/>

⁵<https://hashcat.net/hashcat/>

⁶https://web.archive.org/web/20250627024035/https://hashcat.net/wiki/doku.php?id=frequently_asked_questions

⁷<https://benchmark.unigine.com/heaven>

⁸<https://www.passmark.com/products/performance-test/index.php>

API DirectX dalla versione 9 alla 12. Al termine dei test, PerformanceTest calcola un punteggio sintetico per confrontare le prestazioni tra diverse configurazioni;

- **Fio (Flexible I/O tester):**⁹ software di benchmark che permette di valutare le prestazioni del disco su sistemi Linux. Permette di effettuare test di lettura e scrittura, con accessi sequenziali o casuali, e di configurare diversi parametri come la dimensione dei blocchi, il numero di operazioni da eseguire (profondità della coda) e il numero di thread da utilizzare. Permette inoltre di disattivare l'utilizzo della cache del disco;
- **DiskSpd:**¹⁰ software di benchmark che permette di valutare le prestazioni del disco su sistemi Windows. Come **fio**, permette di effettuare test di lettura e scrittura, con accessi sequenziali o casuali, e di configurare diversi parametri come la dimensione dei blocchi, il numero di operazioni da eseguire (profondità della coda), il numero di thread da utilizzare, e la disattivazione della cache del disco.

4.1.1 Fedora CoreOS

Gli strumenti utilizzati su sistema operativo Fedora CoreOS sono:

- **iperf3** versione 3.17.1, per il benchmark della rete;
- **phoronix-test-suite** versione 10.8.4, con:
 - **pts/compress-7zip** versione 1.11.0, per il benchmark della CPU;
 - **pts/hashcat** versione 1.1.0, per il benchmark della GPU;
 - **pts/unigine-heaven** versione 1.6.6, per il benchmark della GPU;
 - **pts/vkmark** versione 1.3.2, per il benchmark della GPU;

⁹<https://github.com/axboe/fio>

¹⁰<https://github.com/microsoft/diskspd>

- PassMark PerformanceTest Linux versione 11.0, per i benchmark di CPU e memoria;
- fio versione 3.37, per il benchmark del disco.

4.1.2 Windows 10

Gli strumenti utilizzati su sistema operativo Windows 10 sono:

- iperf3 versione 3.18, per il benchmark della rete;
- phoronix-test-suite versione 10.8.4, con:
 - pts/compress-7zip versione 1.11.0, per il benchmark della CPU;
 - pts/unigine-heaven versione 1.6.6, per il benchmark della GPU;
- PassMark PerformanceTest Windows versione 11.1, per i benchmark di CPU, memoria e GPU;
- DiskSpd versione 2.2, per il benchmark del disco.

4.2 Metodologia

I benchmark sono effettuati sulla macchina PC-HP (Tabella 3.1). Per i benchmark della rete con `iperf3` è inoltre utilizzata la macchina secondaria PC-ACER (Tabella 3.2) come server. Le macchine PC-HP e PC-ACER sono collegate a un router GL-AXT1800 con cavi Ethernet 10/100/1000 Mbps.

Per l'installazione dei sistemi operativi guest è utilizzato un disco virtuale separato per ogni hypervisor, per evitare possibili interferenze da driver installati per un altro hypervisor.

Su sistema operativo Windows 10 bare metal sono installati i seguenti driver forniti da HP:

4.2. METODOLOGIA

- Intel NIC Driver¹¹ versione 12.19.2.56 Rev.P per la scheda di rete;
- Intel Video Driver and Control Panel¹² versione 32.0.101.5973 Rev.A per la GPU.

Con hypervisor KVM e Xen, il driver della GPU è installato anche su guest Windows 10 per sfruttare il passthrough mediante SR-IOV.

Per ogni tipo di configurazione e per entrambi i sistemi operativi, Fedora CoreOS e Windows 10, sono disattivati gli aggiornamenti automatici. Su Windows 10 è inoltre disattivato Windows Defender. Durante l'esecuzione dei benchmark, la macchina (PC-HP) non è utilizzata per effettuare altre operazioni. Durante i benchmark della rete, anche la macchina secondaria (PC-ACER) non è utilizzata per altre operazioni. Le prestazioni del disco nelle macchine virtuali sono misurate su dischi virtuali basati su file di 10 GiB, distinti da quello del sistema operativo. I file dei dischi sono pre-allocati alla creazione per evitare errori dovuti al ridimensionamento dinamico [14]. Come formati sono utilizzati `vdi` per VirtualBox e `qcow2` per KVM e Xen. I dischi sono formattati con il filesystem predefinito del sistema operativo: su Fedora CoreOS i dischi sono formattati con il filesystem `xfs`, mentre su Windows 10 con `NTFS`. Negli hypervisor KVM e VirtualBox, per il disco Virtio è disattivata la *cache* nella configurazione della macchina virtuale. Anche se la cache è disabilitata mediante parametri negli strumenti di benchmark, questo risulta ignorato, e le misurazioni raggiungono valori molto superiori alle caratteristiche del disco fisico: in lettura sequenziale risulta una velocità di circa 16 GB/s rispetto ai 3.7 GB/s massimi riportati dal produttore¹³.

In `PassMark PerformanceTest` è possibile impostare il numero di iterazioni da eseguire per ogni test, tuttavia non sono restituiti i singoli valori ma solo quello

¹¹<https://support.hp.com/ie-en/drivers/swdetails/hp-pro-mini-400-g9-desktop-pc/2101054031/swItemId/ob-324953-1>

¹²<https://support.hp.com/ie-en/drivers/swdetails/hp-pro-mini-400-g9-desktop-pc/2101054031/swItemId/vc-339280-1>

¹³<https://web.archive.org/web/20250130165702/https://www.sstc.com/products-detail/CL4-M2-2280-pcie-ssd/>

finale. Sono quindi eseguite singole iterazione più volte, registrando i risultati ottenuti a ogni esecuzione.

Con `fio` sono effettuati benchmark di lettura e scrittura, sia con accessi sequenziali che casuali, in particolare:

- lettura e scrittura sequenziale con blocchi di 1 MiB, profondità della coda di 8, e un thread;
- lettura e scrittura casuale con blocchi di 4 KiB, profondità della coda di 1, e un thread;

I benchmark sono effettuati con una durata di 5 secondi, con un tempo di warmup di 5 secondi, ed è disattivata la cache del disco mediante parametro `direct=1`.

Con `DiskSpd` sono effettuati benchmark di lettura e scrittura, sia con accessi sequenziali che casuali. Per le dimensioni dei blocchi, profondità di coda e numero di thread, sono utilizzati gli stessi valori del benchmark con `fio`. I benchmark sono effettuati con una durata di 10 secondi, con un tempo di warmup di 5 secondi, ed è disattivata la cache del disco mediante parametro `-Sh`.

Per i benchmark della GPU con `Unigine Heaven` e `vkmark` è utilizzata la risoluzione minima di 800x600 pixel, per avere margini di confronto maggiori tra le GPU utilizzate. I benchmark della GPU in `PassMark PerformanceTest` non sono configurabili.

I file di configurazione delle macchine virtuali, e gli script per l'installazione dei pacchetti e la creazione delle macchine virtuali, sono reperibili all'indirizzo: <https://github.com/smagnaniuni/vmm-bench-2025>.

4.3 Risultati

In questa sezione sono presentati i risultati dei benchmark eseguiti sulla macchina PC-HP, su sistemi operativi Fedora CoreOS e Windows 10. I risultati sono suddivisi per tipologia di componente: CPU, memoria, disco, grafica e rete.

Per CPU, memoria e rete sono confrontate quattro configurazioni corrispondenti ai tre hypervisor (VirtualBox, KVM e Xen) e al sistema bare metal.

Configurazioni sopra riportate ma non presenti nei benchmark indicano l'impossibilità a eseguire il benchmark per mancato supporto alle API grafiche.

Nei grafici sono riportati gli intervalli di confidenza al 95%, calcolati utilizzando la distribuzione t di Student, dato il numero di campioni inferiore a 30 [15, p. 50].

4.3.1 CPU

Per i benchmark della CPU sono confrontate le seguenti configurazioni:

- **Xen:** Hypervisor Xen.
File: `fcos-xen.xml` (Fedora CoreOS), `win10-xen.xml` (Windows 10);
- **VirtualBox:** Hypervisor VirtualBox.
File: `fcos-virtualbox.vbox` (Fedora CoreOS), `win10-virtualbox.vbox` (Windows 10);
- **KVM:** Hypervisor KVM.
File: `fcos-qemu.xml` (Fedora CoreOS), `win10-qemu.xml` (Windows 10);
- **Bare Metal:** Sistema bare metal.

Nel benchmark con `compress-7zip` su Fedora CoreOS (Figura 4.1) con l'hypervisor Xen è riscontrato un calo nella velocità di compressione di circa 2% inferiore rispetto al sistema bare metal, seguito da KVM (4%) e VirtualBox (11%). Nei risultati della decompressione non sono riscontrate differenze statisticamente significative tra tutte le configurazioni. Risultati simili sono ottenuti su Windows 10 (Figura 4.2), con una velocità di compressione con Xen circa 1% inferiore rispetto al sistema bare metal, seguito da KVM (4%) e VirtualBox (8%). Anche su Windows 10 non sono riscontrate differenze statisticamente significative nei risultati della decompressione.

4.3. RISULTATI

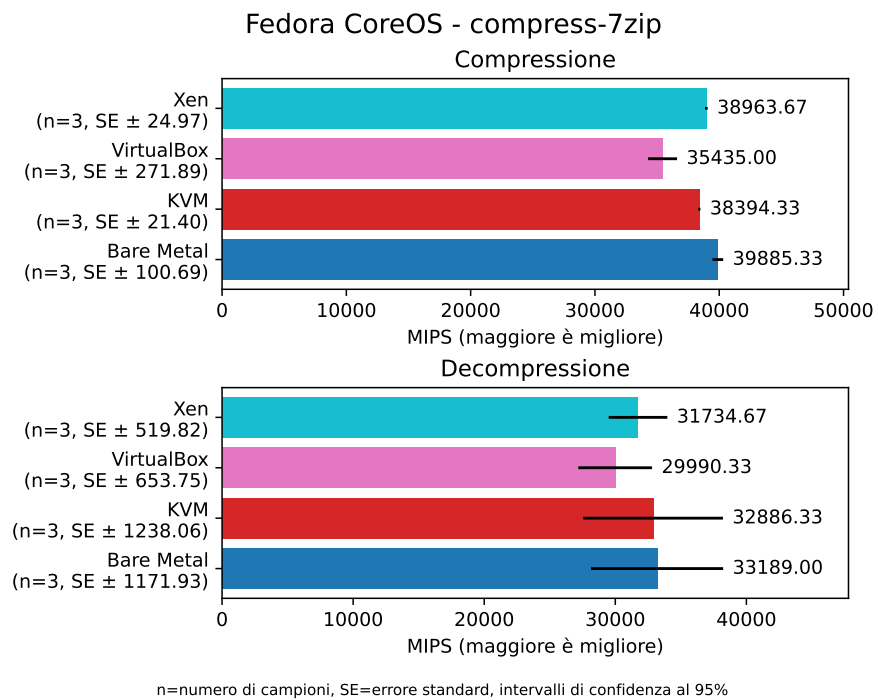


Figura 4.1: Benchmark della CPU con compress-7zip su Fedora CoreOS

4.3. RISULTATI

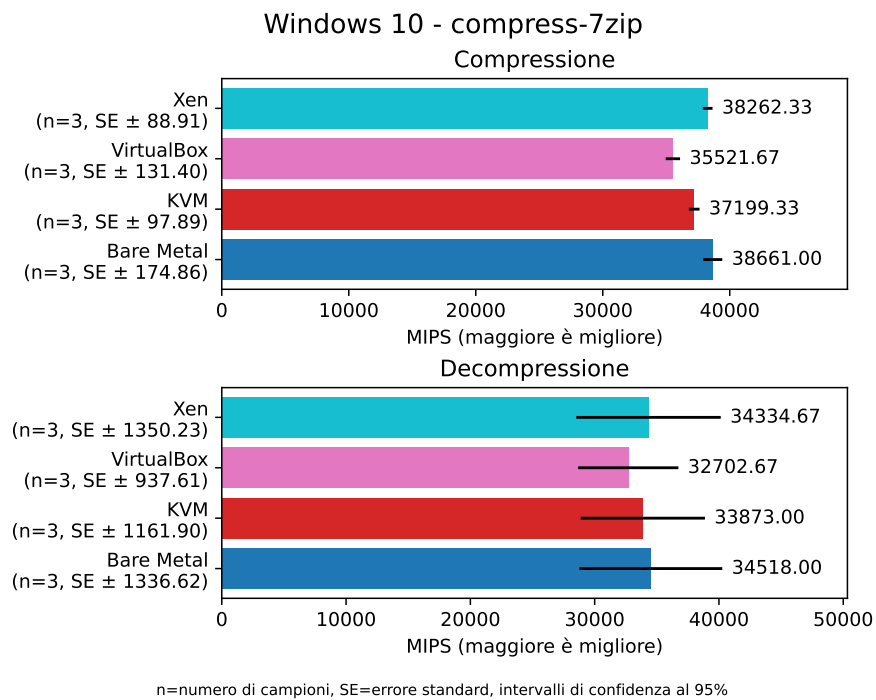


Figura 4.2: Benchmark della CPU con compress-7zip su Windows 10

4.3. RISULTATI

Nel benchmark con `PassMark PerformanceTest` su Fedora CoreOS (Figura 4.3) l'hypervisor KVM raggiunge prestazioni alla pari del sistema bare metal. Non sono osservate differenze statisticamente significative tra le prestazioni di Xen e KVM e tra Xen e il bare metal. Con VirtualBox si verifica una perdita di prestazioni del 2% rispetto al sistema bare metal. Su Windows 10 (Figura 4.4) KVM ottiene il punteggio migliore tra gli hypervisor, ma con una perdita del 4% rispetto al sistema bare metal, seguito da VirtualBox con una perdita del 5%, e Xen con una perdita del 17%.

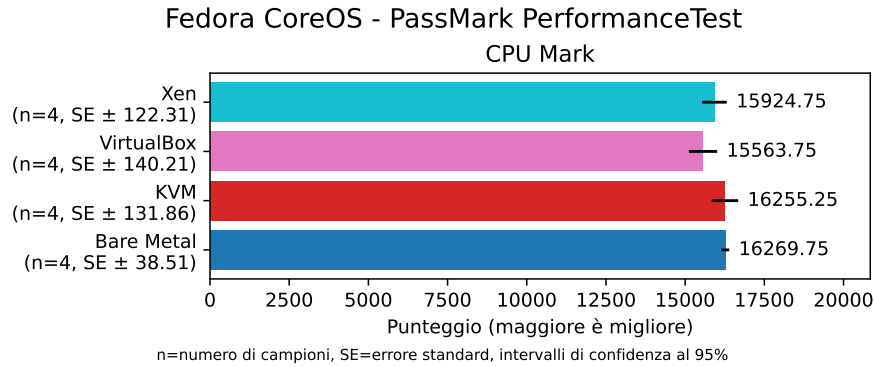


Figura 4.3: Benchmark PassMark PerformanceTest, CPU, Fedora CoreOS

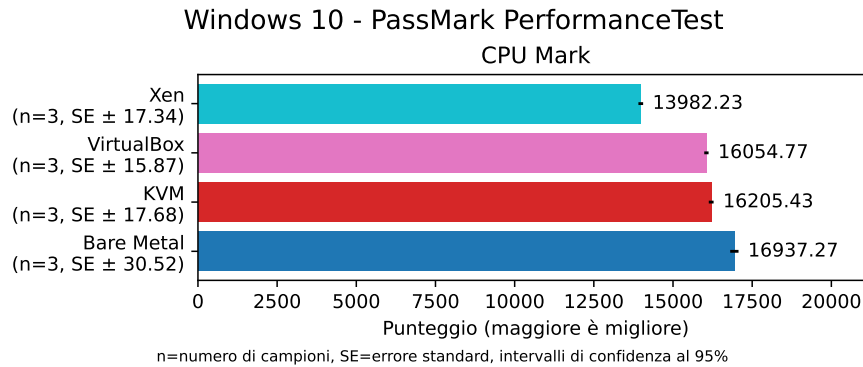


Figura 4.4: Benchmark PassMark PerformanceTest, CPU, Windows 10

4.3.2 Memoria

Per i benchmark della memoria sono confrontate le seguenti configurazioni:

- **Xen:** Hypervisor Xen.
File: `fcos-xen.xml` (Fedora CoreOS), `win10-xen.xml` (Windows 10);
- **VirtualBox:** Hypervisor VirtualBox.
File: `fcos-virtualbox.vbox` (Fedora CoreOS), `win10-virtualbox.vbox` (Windows 10);
- **KVM:** Hypervisor KVM.
File: `fcos-qemu.xml` (Fedora CoreOS), `win10-qemu.xml` (Windows 10);
- **Bare Metal:** Sistema bare metal.

Su Fedora CoreOS nel benchmark della memoria con PassMark PerformanceTest (Figura 4.5) l'hypervisor Xen raggiunge il punteggio migliore, superiore del 7% rispetto al sistema bare metal. Con VirtualBox e KVM sono registrate perdite rispettivamente del 3% e 10% rispetto al sistema bare metal. Su Windows 10 ((Figura 4.6)) Xen ottiene un punteggio inferiore del 4% rispetto al sistema bare metal, seguito da VirtualBox (12%) e KVM (13%).

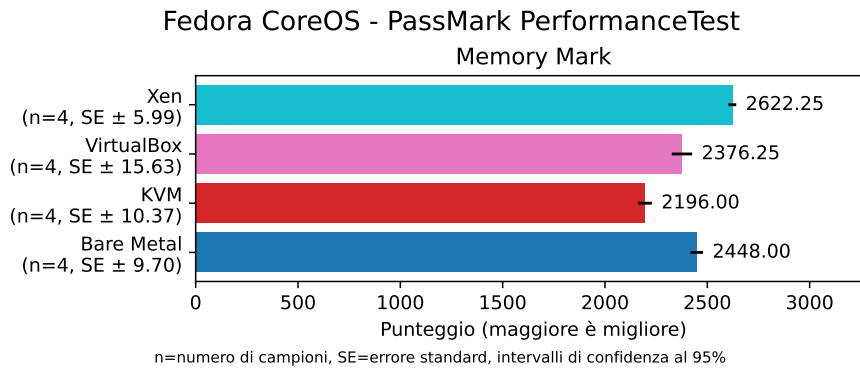


Figura 4.5: Benchmark PassMark PerformanceTest, memoria, Fedora CoreOS

4.3. RISULTATI

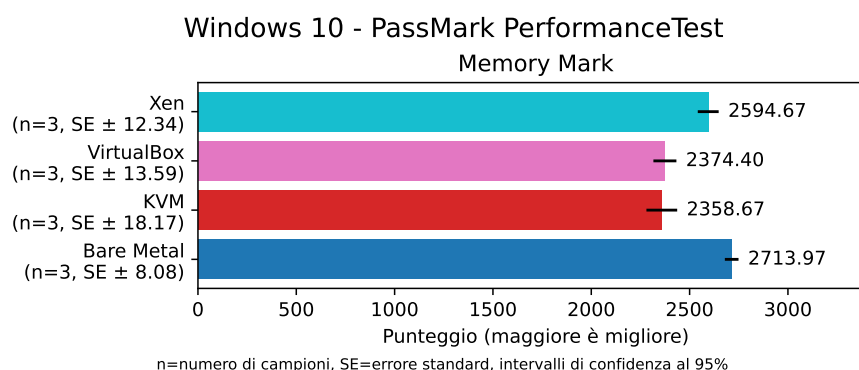


Figura 4.6: Benchmark PassMark PerformanceTest, memoria, Windows 10

4.3.3 Disco

Per i benchmark del disco sono confrontate le seguenti configurazioni:

- Xen, Bus Xen: Hypervisor Xen, disco virtuale Xen.
File: `fcos-xen.xml` (Fedora CoreOS), `win10-xen.xml` (Windows 10);
- Xen, Bus Ide: Hypervisor Xen, disco virtuale IDE.
File: `fcos-xen.xml` (Fedora CoreOS), `win10-xen.xml` (Windows 10);
- VirtualBox, Bus Virtio: Hypervisor VirtualBox, disco virtuale Virtio.
File: `fcos-virtualbox.vbox` (Fedora CoreOS), `win10-virtualbox.vbox` (Windows 10);
- VirtualBox, Bus Sata: Hypervisor VirtualBox, disco virtuale SATA.
File: `fcos-virtualbox.vbox` (Fedora CoreOS), `win10-virtualbox.vbox` (Windows 10);
- KVM, Bus Virtio: Hypervisor KVM, disco virtuale Virtio.
File: `fcos-qemu.xml` (Fedora CoreOS), `win10-qemu.xml` (Windows 10);
- KVM, Bus Sata: Hypervisor KVM, disco virtuale SATA.
File: `fcos-qemu.xml` (Fedora CoreOS), `win10-qemu.xml` (Windows 10);

- **Bare Metal:** Sistema bare metal.

Nel benchmark di letture sequenziali con `fio` su Fedora CoreOS (Figura 4.7) con KVM sono registrate, con dischi SATA e Virtio, prestazioni inferiori rispettivamente del 15% e 17% rispetto al bare metal. Con VirtualBox, i dischi SATA e Virtio ottengono una velocità inferiore del 48% e 54% rispetto al bare metal, ma con alta variabilità. Il disco IDE con Xen ottiene prestazioni inferiori del 72% rispetto al bare metal, mentre il disco paravirtualizzato Xen ottiene prestazioni inferiori del 57% rispetto al bare metal. Nelle letture casuali (Figura 4.8) il disco Virtio con KVM ottiene prestazioni inferiori del 49% rispetto al bare metal. Il disco IDE con Xen ottiene il calo maggiore, il 77% rispetto al bare metal. Nelle scritture sequenziali (Figura 4.9) il disco Virtio con KVM ottiene prestazioni inferiori del 79% rispetto al bare metal. Con VirtualBox si registrano i cali maggiori, del 88% e 86% rispettivamente con dischi SATA e Virtio. Nelle scritture casuali (Figura 4.10) VirtualBox ottiene le prestazioni migliori rispetto agli altri hypervisor, ma con un calo del 77% e 72% rispetto al bare metal, rispettivamente con dischi SATA e Virtio. Il calo maggiore è registrato con Xen (92%).

Nel benchmark di letture sequenziali con `DiskSpd` su Windows 10 (Figura 4.11) con KVM si registrano prestazioni inferiori del 25% e 22% rispetto al bare metal, rispettivamente con dischi SATA e Virtio. Con VirtualBox, i dischi SATA e Virtio ottengono prestazioni inferiori, con cali del 67% e 50% con dischi SATA e Virtio. Con Xen sono registrate le prestazioni peggiori, con un calo del 72% e 69% con dischi IDE e Xen. Nelle letture casuali (Figura 4.12) il risultato migliore è ottenuto con KVM e disco Virtio, ma con un calo del 89% rispetto al bare metal. Il calo maggiore è registrato con Xen e disco IDE (96%). Nelle scritture sequenziali (Figura 4.13) il risultato migliore è ottenuto con KVM e disco Virtio, ma con un calo del 82% rispetto al bare metal. In questo caso il calo maggiore è registrato con VirtualBox, con un calo del 92% e 94% rispettivamente con dischi SATA e Virtio. Nelle scritture casuali (Figura 4.14) il risultato migliore è ottenuto con KVM e disco Virtio, ma con un calo del 78% rispetto al bare metal. Il calo maggiore è

4.3. RISULTATI

registrato con Xen e disco IDE (93%).

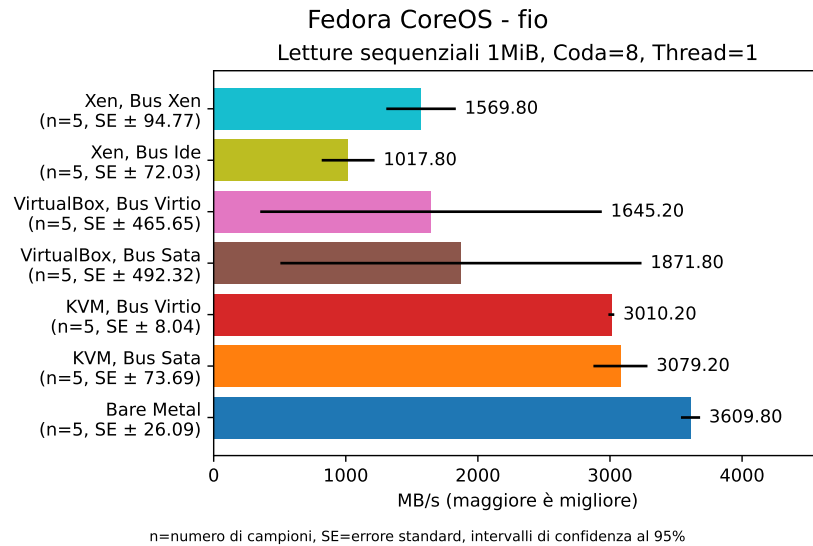


Figura 4.7: Benchmark fio, letture sequenziali con blocco di 1MiB, 8 richieste in coda, 1 thread, Fedora CoreOS

4.3. RISULTATI

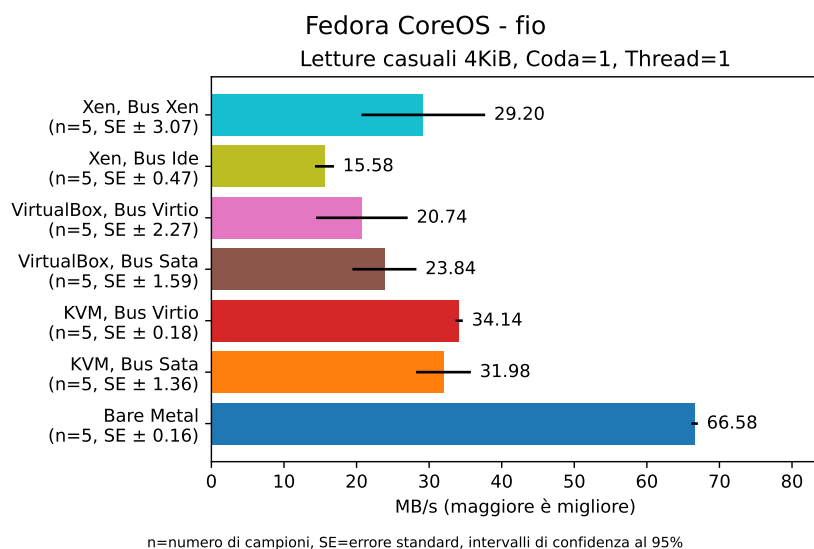


Figura 4.8: Benchmark fio, letture casuali con blocco di 4KiB, 1 richiesta in coda, 1 thread, Fedora CoreOS

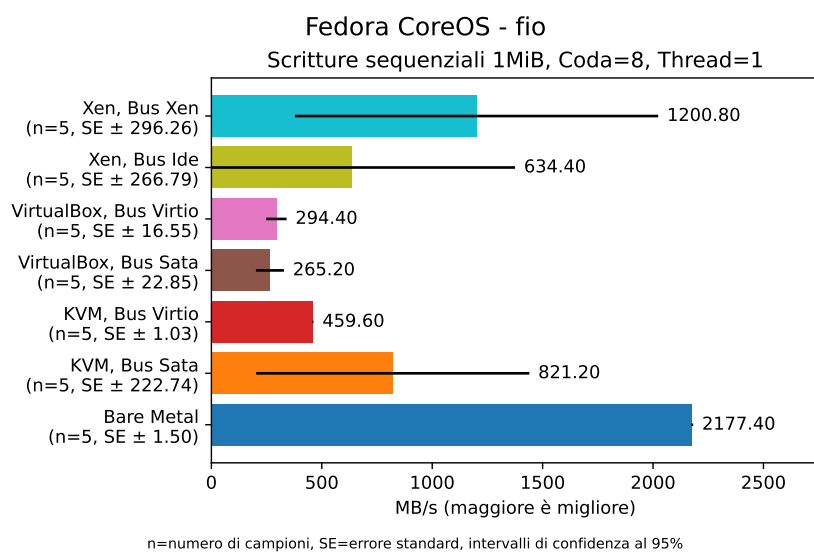


Figura 4.9: Benchmark fio, scritture sequenziali con blocco di 1MiB, 8 richieste in coda, 1 thread, Fedora CoreOS

4.3. RISULTATI

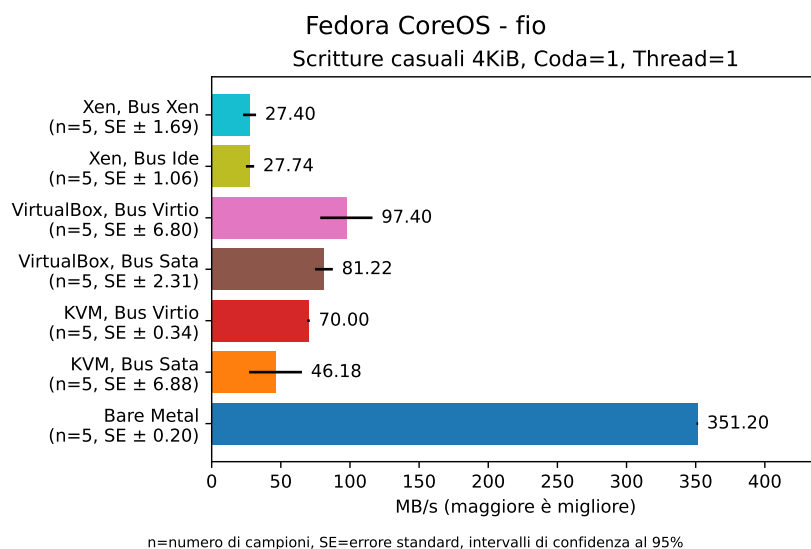


Figura 4.10: Benchmark fio, scritture casuali con blocco di 4KiB, 1 richiesta in coda, 1 thread, Fedora CoreOS

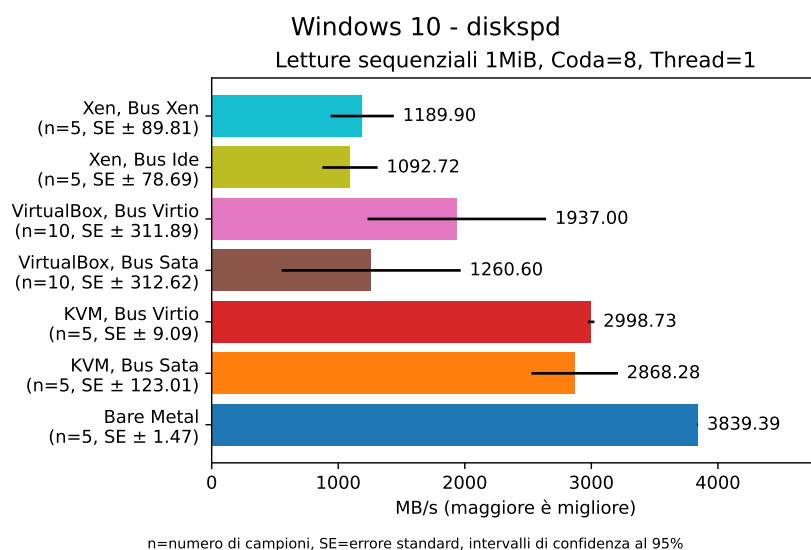


Figura 4.11: Benchmark diskspd, letture sequenziali con blocco di 1MiB, 8 richieste in coda, 1 thread, Windows 10

4.3. RISULTATI

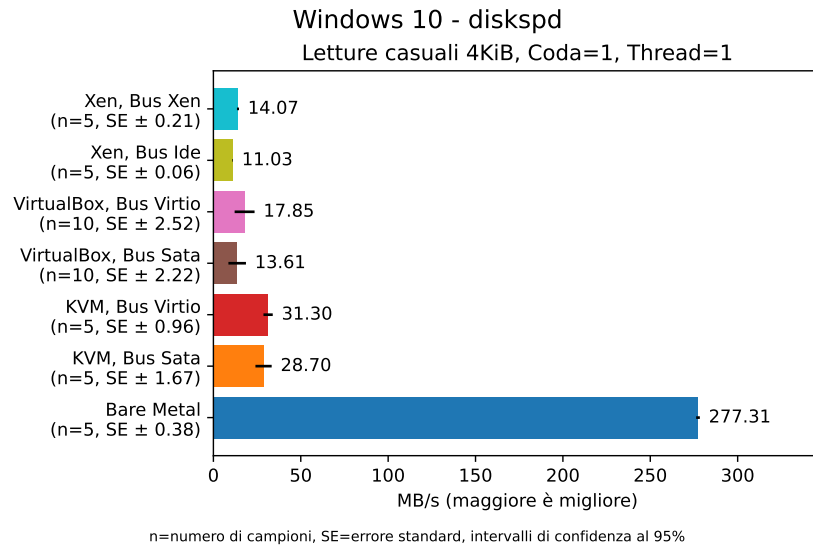


Figura 4.12: Benchmark diskspd, letture casuali con blocco di 4KiB, 1 richiesta in coda, 1 thread, Windows 10

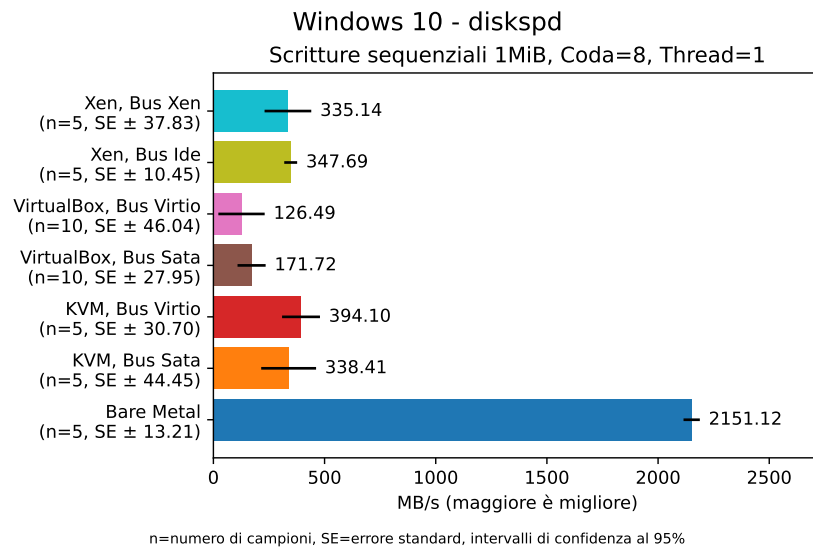


Figura 4.13: Benchmark diskspd, scritture sequenziali con blocco di 1MiB, 8 richieste in coda, 1 thread, Windows 10

4.3. RISULTATI

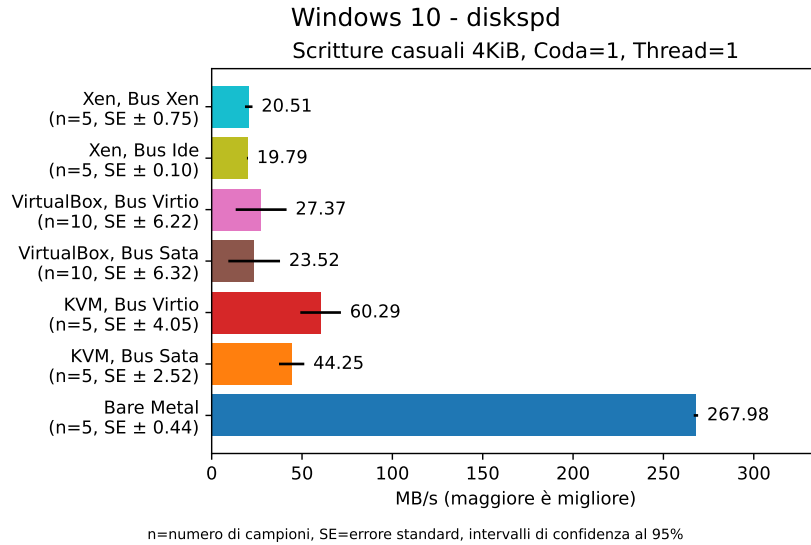


Figura 4.14: Benchmark diskspd, scritture casuali con blocco di 4KiB, 1 richiesta in coda, 1 thread, Windows 10

4.3.4 GPU

Per i benchmark della GPU sono confrontate le seguenti configurazioni:

- **Xen, GPU emulata Xen:** Hypervisor Xen, GPU emulata Xen senza accelerazione 3D.
File: `fcos-xen.xml` (Fedora CoreOS), `win10-xen.xml` (Windows 10);
- **Xen, GPU passthrough SR-IOV (TigerVNC):** Hypervisor Xen, controllo remoto con server TigerVNC (GPU emulata internamente per output video), GPU passthrough con SR-IOV per accelerazione hardware.
File: `fcos-xen-sriov.xml` (Fedora CoreOS);
- **Xen, GPU passthrough SR-IOV (VNC):** Hypervisor Xen, controllo remoto con server VNC di virt-manager e GPU emulata Xen per output video, GPU passthrough con SR-IOV per accelerazione hardware.
File: `win10-xen-sriov.xml` (Windows 10);

- Xen, GPU passthrough SR-IOV (RDP): Hypervisor Xen, controllo remoto con server RDP (GPU emulata internamente per output video), GPU passthrough con SR-IOV per accelerazione hardware.
File: win10-xen-sriov.xml (Windows 10);
- VirtualBox, GPU emulata VBoxSVGA: Hypervisor VirtualBox, GPU emulata VBoxSVGA senza accelerazione 3D.
File: win10-virtualbox.vbox (Windows 10);
- VirtualBox, GPU emulata VMSVGA: Hypervisor VirtualBox, GPU emulata VMSVGA senza accelerazione 3D.
File: fcos-virtualbox.vbox (Fedora CoreOS);
- KVM, GPU passthrough SR-IOV (TigerVNC): Hypervisor KVM, controllo remoto con server TigerVNC (GPU emulata internamente per output video), GPU passthrough con SR-IOV per accelerazione hardware.
File: fcos-qemu-sriov.xml (Fedora CoreOS);
- KVM, GPU passthrough SR-IOV (VNC): Hypervisor KVM, controllo remoto con server VNC di virt-manager e GPU paravirtualizzata Virtio per output video, GPU passthrough con SR-IOV per accelerazione hardware.
File: win10-qemu-sriov.xml (Windows 10);
- KVM, GPU passthrough SR-IOV (RDP): Hypervisor KVM, controllo remoto con server RDP (GPU emulata internamente per output video), GPU passthrough con SR-IOV per accelerazione hardware.
File: win10-qemu-sriov.xml (Windows 10);
- KVM, GPU paravirtualizzata Virtio: Hypervisor KVM, GPU paravirtualizzata Virtio senza accelerazione 3D.
File: win10-qemu.xml (Windows 10);

- **KVM, GPU paravirtualizzata Virtio VirGL:** Hypervisor KVM, GPU paravirtualizzata Virtio con accelerazione 3D tramite VirGL (applicazioni OpenGL).
File: `fcos-qemu.xml` (Fedora CoreOS);
- **Bare Metal:** Sistema bare metal (GPU Intel UHD 770).

Su Windows 10 con **PassMark PerformanceTest** sono effettuati benchmark per la GPU utilizzando le API grafiche DirectX 9, 10, 11 e 12. In tutti i casi, le prestazioni con passthrough della GPU con SR-IOV sono superiori rispetto a GPU emulate e paravirtualizzate, ma inferiori rispetto al sistema bare metal. Nel benchmark con API DirectX 9 (Figura 4.15), con hypervisor KVM e Xen sono registrati rispettivamente cali del 22% e 20% con GPU passthrough tramite SR-IOV rispetto al bare metal, mentre con GPU emulate sono registrati cali tra l'89% e il 90% rispetto al bare metal. Nel benchmark con API DirectX 10 (Figura 4.16), il passthrough tramite SR-IOV mostra un calo maggiore rispetto al bare metal, rispettivamente del 50% e 53% per KVM e Xen, mentre con GPU emulate le perdite di prestazioni sono tra l'81% e l'87%. Nel benchmark con API DirectX 11 (Figura 4.17), il passthrough tramite SR-IOV mostra un calo del 49% e 52% rispetto al bare metal, rispettivamente per KVM e Xen, mentre con GPU emulate le perdite di prestazioni sono maggiori del 92%. Nel benchmark con API DirectX 12 (Figura 4.18), il passthrough tramite SR-IOV mostra un calo significativo rispetto al bare metal, del 86% sia per KVM che per Xen. Non è possibile effettuare il benchmark sia con GPU emulate (Xen e VBoxSVGA) che paravirtualizzata (Virtio) per mancato supporto a DirectX 12.

Nel benchmark **Unigine Heaven** su Fedora CoreOS (Figura 4.19) la GPU Virtio con accelerazione 3D ottiene prestazioni del 17% inferiori rispetto al bare metal, prestazioni migliori rispetto a quelle con passthrough mediante SR-IOV che invece mostra un calo del 37% e del 40% rispettivamente per KVM e Xen. Su Windows 10 (Figura 4.20) è possibile effettuare il benchmark solamente con passthrough

4.3. RISULTATI

mediante SR-IOV, che mostra un calo tra l'11% e il 12% per KVM e Xen rispetto al bare metal.

Nel benchmark `vkmark` su Fedora CoreOS (Figura 4.21) la GPU emulata Xen con hypervisor Xen raggiunge prestazioni del 71% inferiori rispetto al bare metal, seguita dal passthrough mediante SR-IOV con KVM e Xen, rispettivamente con cali del 73% e 77%, e dalla GPU emulata VMSVGA con hypervisor VirtualBox con un calo del 83%. Con la GPU paravirtualizzata Virtio e hypervisor KVM è registrato il calo maggiore, il 95% rispetto al bare metal.

Nel benchmark di computazione con `hashcat` (Figura 4.22) la configurazione con passthrough della GPU mediante SR-IOV raggiunge prestazioni alla pari del sistema bare metal, sia con KVM che con Xen. Non è possibile effettuare il benchmark con GPU emulate e paravirtualizzate per mancato supporto all'API OpenCL.

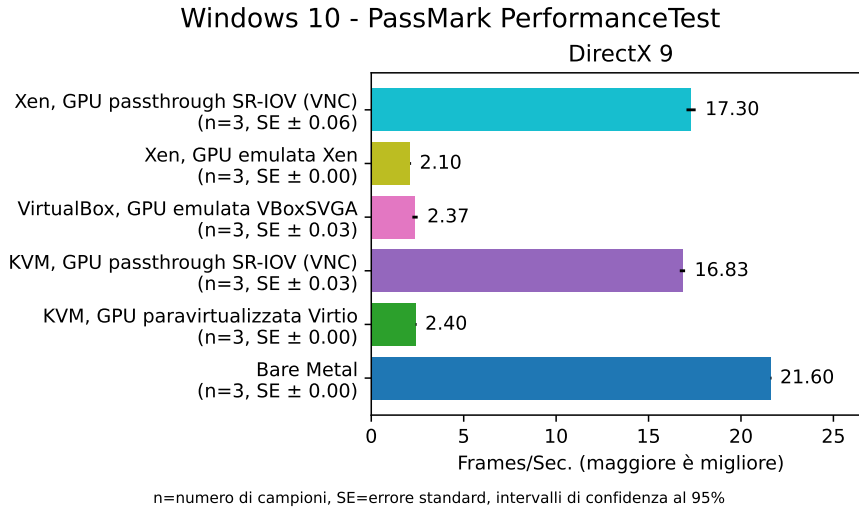


Figura 4.15: Benchmark PassMark PerformanceTest, grafica 3D DirectX 9, Windows 10

4.3. RISULTATI

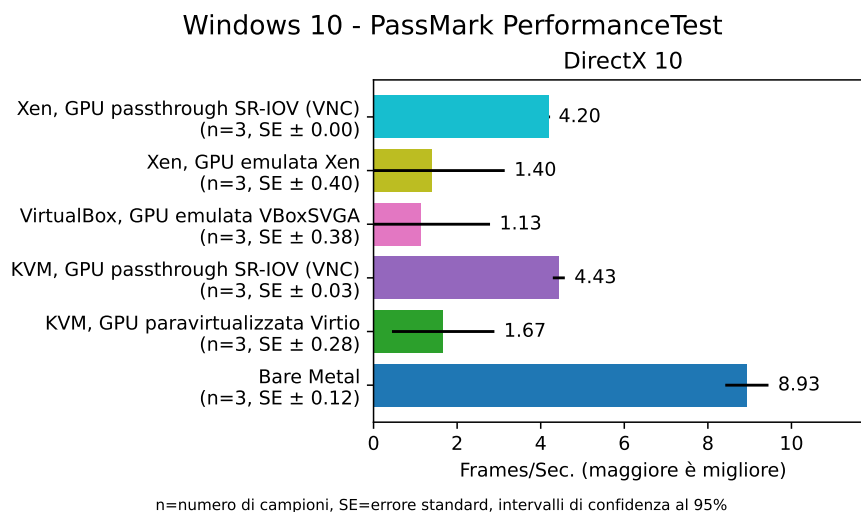


Figura 4.16: Benchmark PassMark PerformanceTest, grafica 3D DirectX 10, Windows 10

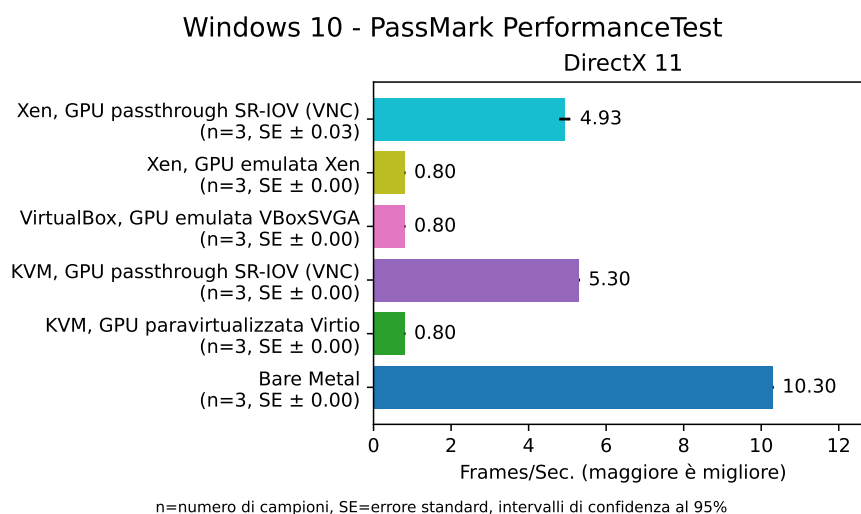


Figura 4.17: Benchmark PassMark PerformanceTest, grafica 3D DirectX 11, Windows 10

4.3. RISULTATI

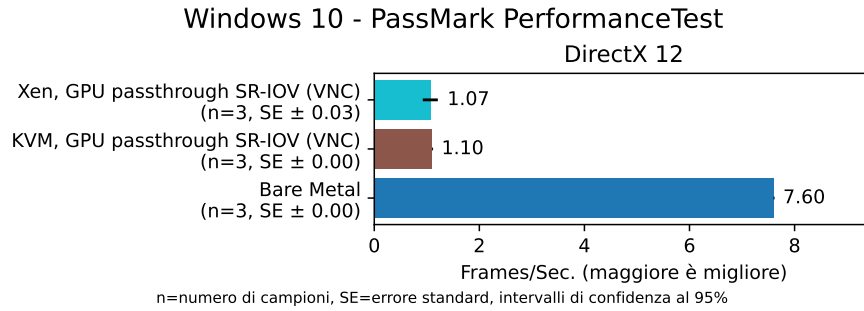


Figura 4.18: Benchmark PassMark PerformanceTest, grafica 3D DirectX 12, Windows 10

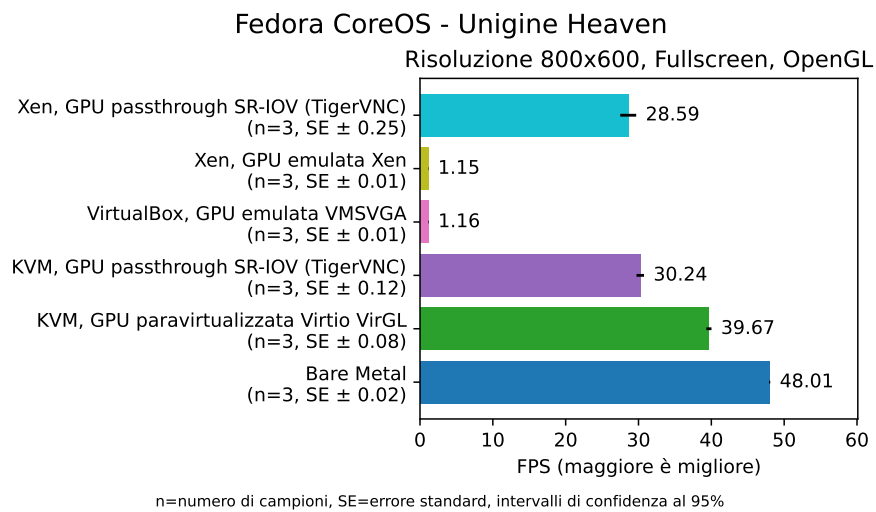


Figura 4.19: Benchmark Unigine Heaven, grafica 3D OpenGL, Fedora CoreOS

4.3. RISULTATI

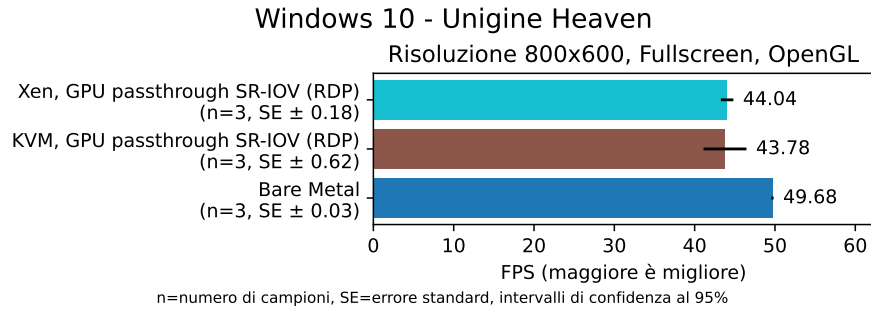


Figura 4.20: Benchmark Unigine Heaven, grafica 3D OpenGL, Windows 10

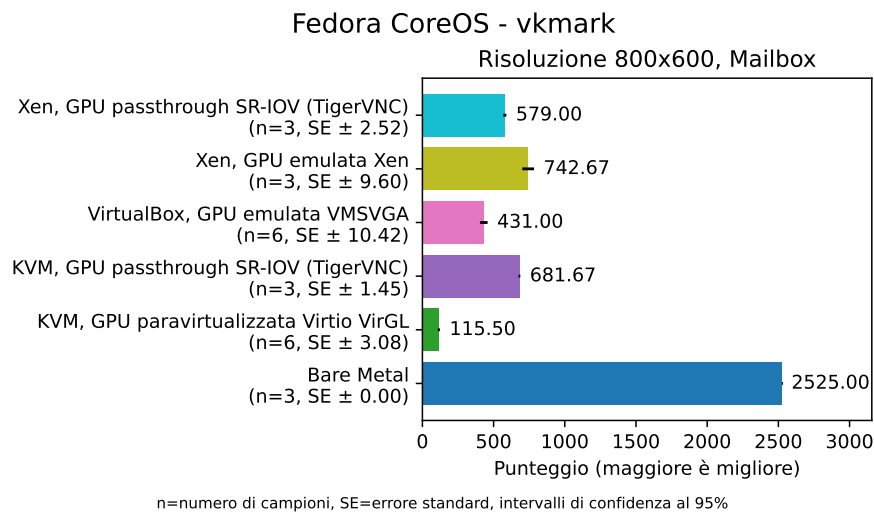


Figura 4.21: Benchmark vkmark, grafica 3D Vulkan, Fedora CoreOS

4.3. RISULTATI

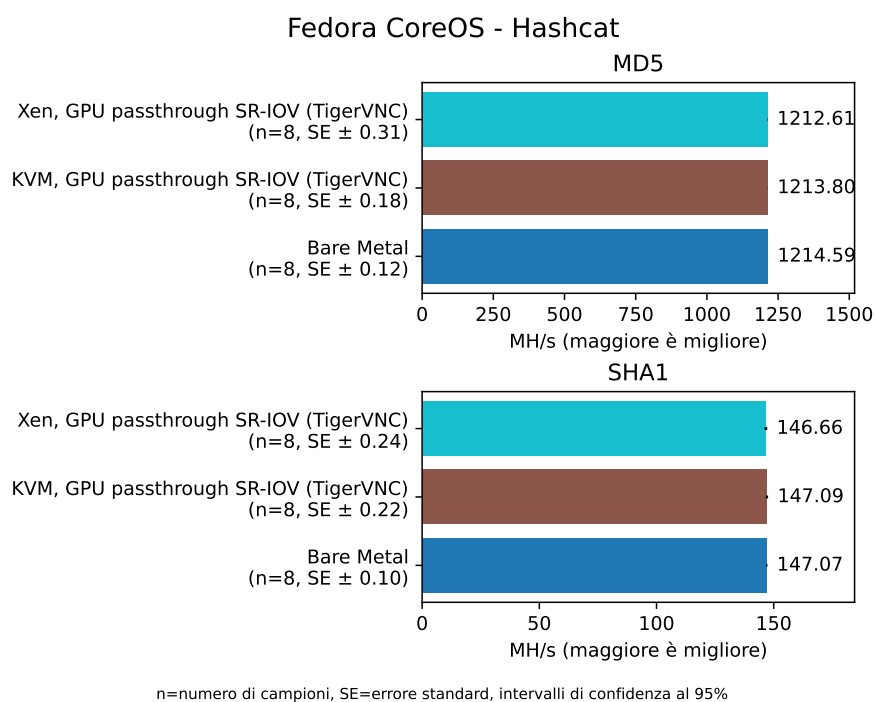


Figura 4.22: Benchmark hashcat, computazione GPU OpenCL, Fedora CoreOS

4.3.5 Rete

Per i benchmark della rete sono confrontate le seguenti configurazioni:

- **Xen:** Hypervisor Xen.
File: `fcos-xen.xml` (Fedora CoreOS), `win10-xen.xml` (Windows 10);
- **VirtualBox:** Hypervisor VirtualBox.
File: `fcos-virtualbox.vbox` (Fedora CoreOS), `win10-virtualbox.vbox` (Windows 10);
- **KVM:** Hypervisor KVM.
File: `fcos-qemu.xml` (Fedora CoreOS), `win10-qemu.xml` (Windows 10);
- **Bare Metal:** Sistema bare metal.

Nei benchmark della rete con `iperf3` su Fedora CoreOS (Figura 4.23) tutti gli hypervisor raggiungono le stesse prestazioni del sistema bare metal, senza differenze statisticamente significative, sia utilizzando il protocollo TCP che UDP. Nello stesso benchmark su Windows 10, (Figura 4.24) con protocollo TCP, VirtualBox mostra un calo del 13% rispetto al sistema bare metal, mentre KVM e Xen ottengono prestazioni migliori di circa l'1%. Con il protocollo UDP, tutti gli hypervisor ottengono prestazioni simili al sistema bare metal, senza differenze statisticamente significative.

4.3. RISULTATI

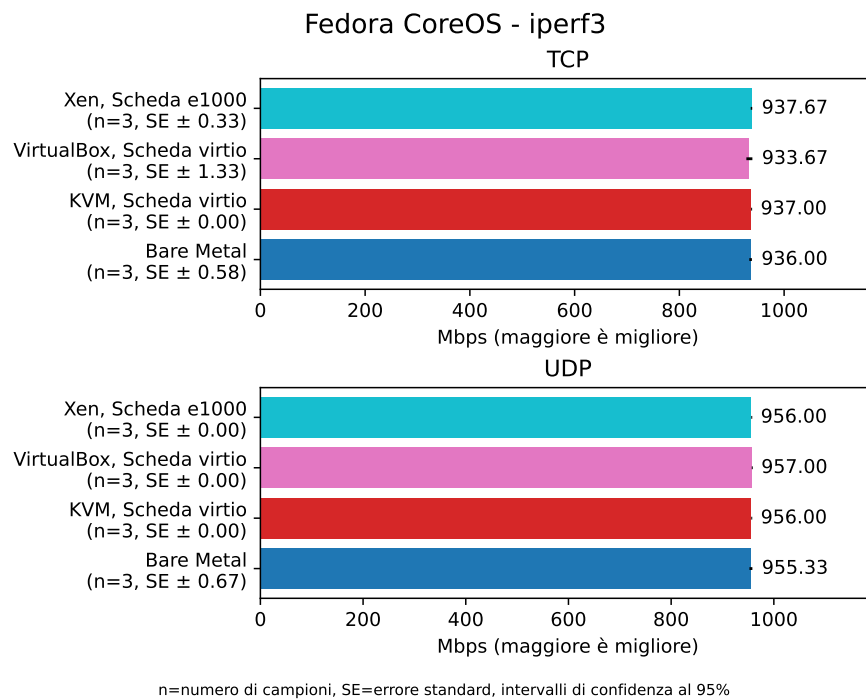


Figura 4.23: Benchmark iperf3, Fedora CoreOS

4.3. RISULTATI

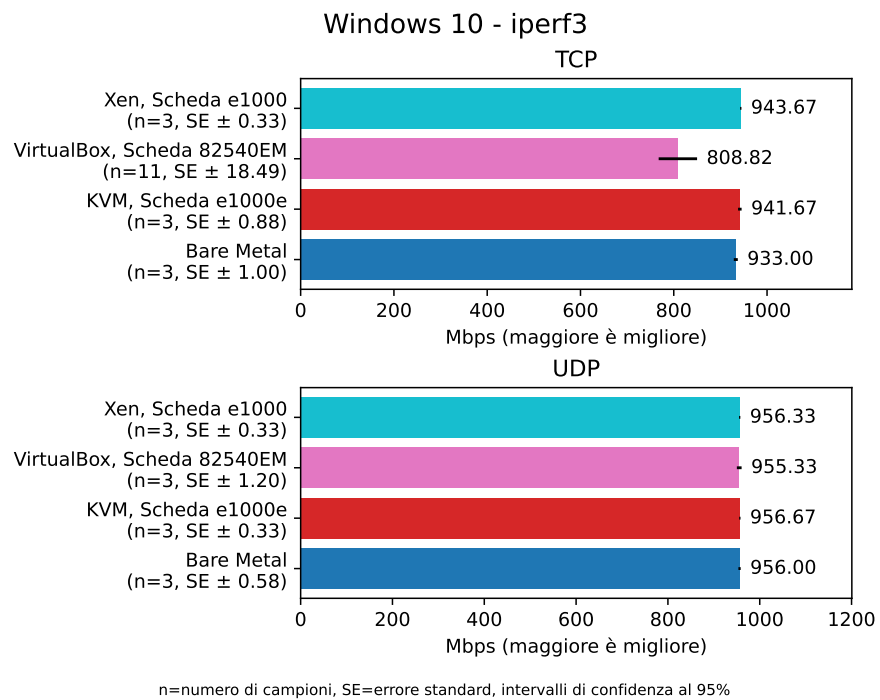


Figura 4.24: Benchmark iperf3, Windows 10

4.4 Discussione

Gli hypervisor VirtualBox, KVM e Xen mostrano prestazioni simili al sistema bare metal nei benchmark di CPU, memoria e rete. Nei benchmark della CPU, VirtualBox risulta il meno performante, a eccezione di **PerformanceTest** su sistema Windows 10, dove Xen subisce un calo di prestazioni rispetto al bare metal del 17%, il più elevato in tutti i benchmark del processore.

Nei benchmark della memoria Xen risulta l'hypervisor più performante, superando anche il bare metal su sistema Fedora CoreOS del 7%. Complessivamente, i cali sono tra il 3% e il 13%.

Le prestazioni del disco sono le più impattate rispetto al bare metal, in alcuni casi con cali superiori al 50%. I cali di prestazione osservati con KVM in letture sequenziali (15-25%) sono leggermente inferiori rispetto a quanto riportato da Wen et al. [12] (35%). Considerando complessivamente tutti i benchmark, con il disco Virtio di KVM è osservata una deviazione standard più stabile rispetto ad altri tipi di disco e hypervisor.

Un problema riscontrato nella virtualizzazione della GPU riguarda la compatibilità tra le API utilizzate e le GPU emulate e paravirtualizzate, in cui devono essere implementate le specifiche API. In questo caso, la possibilità di effettuare il passthrough della GPU consente di aggirare questo problema, permettendo inoltre di sfruttare gli stessi driver utilizzati nel sistema bare metal. Le GPU emulate e paravirtualizzate non risultano adeguate per applicazioni grafiche 3D; nei benchmark sono osservati cali di prestazione fino al 95%. Il passthrough della GPU con SR-IOV ottiene prestazioni maggiori rispetto a GPU emulate e paravirtualizzate, a eccezione del benchmark **vkmark**. Inoltre, permette una migliore compatibilità con le API grafiche, tuttavia questo è ostacolato dalla necessità di utilizzare una GPU emulata per la visualizzazione dello schermo. In alcuni benchmark di grafica sono riscontrati risultati inattesi. Nel benchmark **Unigine Heaven**, che utilizza API OpenGL, la GPU Virtio con accelerazione 3D ottiene prestazioni superiori del 31-39% rispetto al passthrough con SR-IOV. Nel benchmark **vkmark**, che utilizza

API Vulkan, la GPU emulata Xen ottiene prestazioni superiori al passthrough con SR-IOV. Inoltre, nello stesso benchmark, la GPU paravirtualizzata Virtio ottiene il calo maggiore (95% rispetto al bare metal), un calo maggiore anche rispetto alle altre GPU emulate senza accelerazione 3D. Tuttavia, questo potrebbe essere spiegato dal fatto che l'accelerazione 3D tramite VirGL è limitata a API OpenGL. Nelle applicazioni di calcolo, cioè nel benchmark `hashcat` (API OpenCL), il GPU passthrough con SR-IOV ottiene prestazioni quasi identiche a quelle di un sistema bare metal, risultato in linea con quanto riscontrato da Walters et al. [9] e Younge et al. [10]. Per applicazioni grafiche invece rimangono margini di miglioramento: il benchmark delle API DirectX 12 con `PassMark PerformanceTest` risulta il più impattato, con prestazioni inferiori del 86% rispetto al bare metal. Questa differenza di efficacia del passthrough tra applicazioni grafiche e computazionali è stato osservato anche da Shea e Liu [8].

Le prestazioni di rete con tutti gli hypervisor risultano identiche o migliori dell'1% rispetto al bare metal, a eccezione di VirtualBox su Windows 10, con cui è osservato un calo del 13% nel benchmark con protocollo TCP. Questo potrebbe essere causato dall'utilizzo di una scheda di rete emulata e non paravirtualizzata.

Capitolo 5

Conclusioni

Obiettivo di questa tesi era quello di confrontare le prestazioni di un sistema virtualizzato rispetto al bare metal. È stata utilizzata una macchina HP Pro Mini 400 G9 e gli hypervisor VirtualBox, KVM e Xen, con sistemi operativi Fedora CoreOS e Windows 10. Nei benchmark di CPU, memoria e rete, gli hypervisor VirtualBox, KVM e Xen hanno raggiunto prestazioni simili a quelle di un sistema bare metal, tranne per alcuni casi dove sono stati riscontrati cali di prestazioni tra il 10% e il 20% rispetto al bare metal. I cali maggiori sono stati osservati nei benchmark del disco. In particolare, nei benchmark di scritture sequenziali e letture/scritture casuali sono stati osservati cali di prestazioni superiori al 50% rispetto al bare metal con tutti gli hypervisor. In letture sequenziali, con i dischi SATA e Virtio di KVM sono state ottenute le prestazioni migliori rispetto agli altri hypervisor, ma con un calo tra il 15% e il 25% rispetto al bare metal.

È stato effettuato con successo il *mediated passthrough* tramite SR-IOV di una GPU integrata Intel UHD 770 con hypervisor KVM e Xen, ma solo per accelerazione hardware; per la visualizzazione dello schermo è stato necessario utilizzare una seconda GPU emulata. Questo ha portato a problemi di compatibilità con le API grafiche su sistema Windows 10, che ha richiesto di utilizzare due diversi server per il controllo della macchina virtuale, a seconda che l'applicazione utiliz-

zasse API OpenGL o DirectX. Tuttavia, questa soluzione non è adeguata nel caso sia necessario utilizzare entrambe le API grafiche.

La GPU paravirtualizzata Virtio con accelerazione 3D per applicazioni OpenGL, tramite VirGL, ha ottenuto prestazioni superiori rispetto al passthrough con SR-IOV. Tuttavia, in questo studio è stato necessario utilizzare un modulo i915 out-of-tree, dato che la tecnologia SR-IOV per GPU Intel non era ancora completamente supportata nel kernel upstream di Linux. È quindi necessario verificare nuovamente le prestazioni quando il supporto sarà integrato nel kernel upstream.

VirtualBox è risultato limitato dal mancato supporto al PCI passthrough; questo lo ha penalizzato nella virtualizzazione della GPU, non solo in termini di prestazioni, ma anche in termini di compatibilità con le API grafiche e di computazione, che richiedono di essere appositamente supportate da una GPU (e quindi un'apposita implementazione nel software di una GPU emulata).

Come possibili sviluppi futuri si propongono i seguenti suggerimenti:

- Investigare la possibilità di utilizzare una partizione del disco fisico come disco virtuale, o di effettuare il PCI passthrough di un intero disco aggiuntivo, invece di utilizzare dischi basati su file;
- Per ottenere una maggiore compatibilità con le API grafiche, valutare l'aggiunta di una GPU dedicata e l'effettuazione di *fixed passthrough*. Si suggerisce inoltre di monitorare QEMU per eventuali miglioramenti al supporto del *fixed passthrough* per GPU integrate Intel, e il modulo i915 del kernel Linux per il supporto ufficiale di SR-IOV per GPU Intel.

Acronimi

API Application Programming Interface.

BIOS Basic Input/Output System.

DKMS Dynamic Kernel Module Support.

GB Gigabyte.

GiB Gibibyte.

GPU Graphics Processing Unit.

I/O Input/Output.

IDE Integrated Drive Electronics.

IGD Intel Graphics Device.

IOMMU Input-Output Memory Management Unit.

KB Kilobyte.

KiB Kibibyte.

KVM Kernel-based Virtual Machine.

LXC Linux Containers.

MB Megabyte.

Mbps Megabit per second.

MiB Mebibyte.

PCI Peripheral Component Interconnect.

PF Physical Function.

ROM Read-Only Memory.

S.O. Sistema operativo.

SATA Serial AT Attachment.

SR-IOV Single-Root Input/Output Virtualization.

SVM Secure Virtual Machine.

UEFI Unified Extensible Firmware Interface.

VBIOS Video BIOS.

VF Virtual Function.

Virtio Virtual I/O Device.

VM Virtual Machine.

VMM Virtual Machine Monitor.

VMX Virtual Machine Extensions.

Bibliografia

- [1] Fernando Rodríguez-Haro et al. «A summary of virtualization techniques». In: *Procedia Technology* 3 (2012). The 2012 Iberoamerican Conference on Electronics Engineering and Computer Science, pp. 267–272. ISSN: 2212-0173. DOI: 10.1016/j.protcy.2012.03.029. URL: <https://doi.org/10.1016/j.protcy.2012.03.029>.
- [2] E. C. Hendricks e T. C. Hartmann. «Evolution of a virtual machine subsystem». In: *IBM Systems Journal* 18.1 (1979), pp. 111–142. ISSN: 0018-8670. DOI: 10.1147/sj.181.0111.
- [3] R. J. Creasy. «The Origin of the VM/370 Time-Sharing System». In: *IBM Journal of Research and Development* 25.5 (set. 1981), pp. 483–490. ISSN: 0018-8646. DOI: 10.1147/rd.255.0483.
- [4] Gerald J. Popek e Robert P. Goldberg. «Formal requirements for virtualizable third generation architectures». In: *Commun. ACM* 17.7 (lug. 1974), pp. 412–421. ISSN: 0001-0782. DOI: 10.1145/361011.361073. URL: <https://doi.org/10.1145/361011.361073>.
- [5] Harry Katzan. «Operating systems architecture». In: *Proceedings of the May 5-7, 1970, Spring Joint Computer Conference*. AFIPS '70 (Spring). Atlantic City, New Jersey: Association for Computing Machinery, 1970, pp. 109–118. ISBN: 9781450379038. DOI: 10.1145/1476936.1476960. URL: <https://doi.org/10.1145/1476936.1476960>.

- [6] Micah Dowty e Jeremy Sugerman. «GPU virtualization on VMware's hosted I/O architecture». In: *SIGOPS Oper. Syst. Rev.* 43.3 (lug. 2009), pp. 73–82. ISSN: 0163-5980. DOI: 10.1145/1618525.1618534. URL: <https://doi.org/10.1145/1618525.1618534>.
- [7] Charles David Graziano. «A performance analysis of Xen and KVM hypervisors for hosting the Xen Worlds Project». Tesi di laurea mag. Iowa State University, 2011. DOI: 10.31274/etd-180810-2322.
- [8] Ryan Shea e Jiangchuan Liu. «On GPU pass-through performance for cloud gaming: Experiments and analysis». In: *2013 12th Annual Workshop on Network and Systems Support for Games (NetGames)*. Dic. 2013, pp. 1–6. DOI: 10.1109/NetGames.2013.6820614.
- [9] John Paul Walters et al. «GPU Passthrough Performance: A Comparison of KVM, Xen, VMWare ESXi, and LXC for CUDA and OpenCL Applications». In: *2014 IEEE 7th International Conference on Cloud Computing*. Giu. 2014, pp. 636–643. DOI: 10.1109/CLOUD.2014.90. URL: <https://doi.org/10.1109/CLOUD.2014.90>.
- [10] Andrew J. Younge et al. «Evaluating GPU Passthrough in Xen for High Performance Cloud Computing». In: *2014 IEEE International Parallel & Distributed Processing Symposium Workshops*. Mag. 2014, pp. 852–859. DOI: 10.1109/IPDPSW.2014.97.
- [11] Sultan Abdullah Algarni et al. «Performance Evaluation of Xen, KVM, and Proxmox Hypervisors». In: *Int. J. Open Source Softw. Process.* 9.2 (2018), pp. 39–54. DOI: 10.4018/IJOSSP.2018040103. URL: <https://doi.org/10.4018/IJOSSP.2018040103>.
- [12] Long Wen et al. «Bare-Metal vs. Hypervisors and Containers: Performance Evaluation of Virtualization Technologies for Software-Defined Vehicles». In: *IEEE Intelligent Vehicles Symposium, IV 2023, Anchorage, AK, USA, June*

- 4-7, 2023. IEEE, 2023, pp. 1–8. DOI: 10.1109/IV55152.2023.10186789. URL: <https://doi.org/10.1109/IV55152.2023.10186789>.
- [13] Hafiz ur Rahman et al. «Performance Evaluation of Hypervisors and the Effect of Virtual CPU on Performance». In: *2018 IEEE SmartWorld, Ubiquitous Intelligence & Computing, Advanced & Trusted Computing, Scalable Computing & Communications, Cloud & Big Data Computing, Internet of People and Smart City Innovation (SmartWorld/SCALCOM/UIC/ATC/CBDCCom/IOP/SCI)*. Ott. 2018, pp. 772–779. DOI: 10.1109/SmartWorld.2018.00146. URL: <http://www.smart-world.org/2018/BestPaperAwards/UIC-Hafiz+Performance%20Evaluation%20of%20Hypervisors%20and%20the%20Effect.pdf>.
- [14] Saverio Giallorenzo et al. «Virtualization Costs: Benchmarking Containers and Virtual Machines Against Bare-Metal». In: *SN Computer Science* 2.5 (ago. 2021). ISSN: 2661-8907. DOI: 10.1007/s42979-021-00781-8. URL: <http://dx.doi.org/10.1007/s42979-021-00781-8>.
- [15] David J. Lilja. *Measuring Computer Performance: A Practitioner's Guide*. Cambridge University Press, 2000. ISBN: 9780511612398. DOI: 10.1017/CB09780511612398.