PCIe GPU passthrough on Debian

Nao Pross

December 31, 2016

1 Introduzione

Recentemente grazie all'evoluzione delle tecnologie di virtualizzazione è diventato possibile passare direttamente un dispositivo collegato in uno slot PCI ad una macchina virtuale, che può quindi usufruire dell'hardware a performance quasi nativa. Un computer configurato per funzionare in tale maniera apparirà come se fossero due computer distinti con due monitors (o ad uno con un kvm switch). Il vantaggio di un setup del genere è la possibilità di cambiare immediatamente da un sistema operativo all'altro senza dover attendere che ma macchina si riavvii (in un dualboot) e si evitano potenziali problemi come Windows Update che rendono il PC inusabile.

2 Limitazioni

Come ogni sistema si ha delle limitazioni quali

- Per poter vedere l'uscita video della VM si deve collegare un display ad una delle uscite della scheda grafica passata
- Non è possibile passare la iGPU ad una VM (è possibile ma con altre tecnologie ancora sperimentali)
- Non è possibile passare una GPU discreta di un laptop (es NVIDIA Optimus) poichè non è possibile visualizzare il video in uscita

3 Funzionamento (In grandi linee)

Le macchine virtuali vengono emulate grazie ad un software chiamato QEMU che supporta il passaggio di hardware PCI alla VM, inoltre QEMU utilizza un estensione chiamata KVM che permette di creare delle macchine virtuali collegate alla kernel dunque ottimizzate. Altrimenti QEMU dovrebbe emulare l'intero processore su cui viene virtualizzato il sistema operativo. Infine per gestire questi due componenti LibVirt è un software che salva le configurazioni di QEMU in dei file XML e che permette alle VM di essere tenute accese in background.

Passando una scheda video in PCI ad una macchina virtuale permette di utilizzare la potenza grafica hardware che sarebbe altrimenti emulata (le grafichce virtuali di QEMU non sono per niente ottimizzate, per tali scopi è meglio utilizzare VirtualBox o VMWare Fusion). Quindi per poter vedere la grafica della macchina virtuale si deve collegare un monitor ad una delle entrate della scheda video.

4 Hardware Requirements

- Intel processor with iGPU (or 2 GPUs)
- Processor with VT-x and VT-d
- NVIDIA or AMD graphics card (2 if with no iGPU)
- Motherboard with VT-d support
- Motherboard with UEFI based bootloader
- Motherboard with IOMMU support (per configurazioni avanzate)

5 Software Requirements

- Linux Kernel with iommu_groups support (kernel >= 3.9)

6 Hardware utilizzato

Per questo questo test ho usato una configurazione hardware esagerata, anche con un sistema decisamente meno costoso è possibile realizzare il progetto.

- ASUS Z170-A motherboard
- Intel Core i7 Skylake 6700 CPU with Intel Graphics 530
- MSI NVIDIA GTX1050Ti OC 4GB
- 32GB DDR4

7 BIOS Settings

Per incominciare è necessario controllare che nel BIOS siano attivate le tecnologie di virtualizzazione VT-x e VT-d. Inoltre se si vuole utilizzare la iGPU in alcune motherboard (come nel mio caso) si deve modificare un impostazione nel bios per non disattivarla qundo viene inserita una scheda grafica secondaria.

8 Sistema Operativo

Per avere un sistema stabile consiglio di utilizzare Debian Linux poiché essendo utilizzato soprattuto nei servers il ciclo di testing e updates è molto lento in modo da avere sempre un supporto legacy e un sistema stabile. Purtroppo però attualmente (12.2016) Debian 8 Jessie ha ancora la kernel 3.16 quindi si deve utilizzare la versione testing (9) Stretch. Anche se Debian Testing non è pensato per essere utilizzato come sistema primario (i bug possono essere ignorati a lungo nei repo) il supporto di aggiornamenti software è garantito per un perido molto più esteso delle altre distro.

Prima di installare il sistema operativo è consigliato rimuovere la GPU che si utilizzerà per il passthrough dallo slot PCI per evitare che Linux si autoconfiguri per utilizzare i suoi driver grafici. Una volta installato il sistema si devono installare i seguenti pacchetti:

```
% sudo apt-get install \
  qemu-kvm libvirt-bin virtinst bridge-utils virt-manager ssh-askpass ovmf
```

9 Kernel Modules and GRUB

Normalmente Debian viene con GRUB2 preinstallato come bootloader, se si dovesse avere un altro bootloader si deve semplicemente passare gli stessi parametri alla kernel quando si avvia.

9.1 IOMMU

Dalla kernel 3.9 Linux ha introdotto gli iommu_groups che permettono di mappare dispositivi di memoria reale ad indirizzi virtuali che possono essere a loro volta passati alla VM. Per poter utilizzare questa funzionalità la si deve abilitare inoltre è necessario anche il supporto degli ACS per poter suddividere i sottocomponenti del gruppo iommu (per esempio le porte individuali di un estensione di USB via PCI). Per attivare le funzionalità si deve passare alla kernel i seguenti parametri:

```
# file: /etc/default/grub
...
GRUB_DEFAULT_CMDLINE_LINUX_DEFAULT="intel_iommu=on iommu=1 pcie_acs_override=downstream"
...
```

Per aggiornale le impostazioni installate

```
% sudo update-grub
```

10 GPU Settings

Per evitare che Debian utilizzi la GPU all'avvio si deve modificare le impostazioni dell'initramfs in maniera tale che il modulo della kernel non venga avviato. Per sapere il nome del modulo della kernel caricato per la GPU si può usare il seguente comando (nel mio caso la scheda è una NVIDIA):

```
% lspci -nnk | grep -i -A2 nvidia
02:00.0 VGA compatible controller [0300]: NVIDIA Corporation GP107 [GeForce GTX 1050 Ti] [10de:1c82] (rev a1)
        Subsystem: Micro-Star International Co., Ltd. [MSI] GP107 [GeForce GTX 1050 Ti] [1462:8c96]
        Kernel driver in use: nouveau
--
02:00.1 Audio device [0403]: NVIDIA Corporation Device [10de:0fb9] (rev a1)
        Subsystem: Micro-Star International Co., Ltd. [MSI] Device [1462:8c96]
        Kernel driver in use: snd_hda_intel
```

Nel file di configurazione

```
# file: /etc/modprobe.d/blacklist.conf
...
blacklist nouveau
```

Infine per aggiornare la configurazione

```
% sudo update-initramfs -u
```

11 Driver binding

Per passare un dispositivo ad una VM si deve indicare alla kernel di utilizzare il modulo `vfio-pci', perciò è necessario inizializzare questi dispositivi all'avvio. Per questo possiamo usare questo script preso da qui salvandolo come `/usr/local/bin/vfio-bind'.

```
# file: /usr/local/bin/vfio-bind

#!/bin/bash

modprobe vfio-pci

for dev in "$@"; do
        vendor=$(cat /sys/bus/pci/devices/$dev/vendor)
        device=$(cat /sys/bus/pci/devices/$dev/device)
        if [ -e /sys/bus/pci/devices/$dev/driver ]; then
                echo $dev > /sys/bus/pci/devices/$dev/driver/unbind
        fi
        echo $vendor $device > /sys/bus/pci/drivers/vfio-pci/new_id
done
```

Attivando la flag per poterlo eseguire:

```
% sudo chmod +x /usr/local/bin/vfio-bind
```

Con questo script possiamo collegare il driver per la virtualizzazione a qualsiasi dispositivo PCI nella seguente maniera:

```
% sudo vfio-bind <indirizzi pci>
```

Per trovare l'indizirro dei dispositivi che vogliamo passare utilizziamo di nuovo il seguente comando:

```
% lspci -nnk | grep -i -A2 nvidia
02:00.0 VGA compatible controller [0300]: NVIDIA Corporation GP107 [GeForce GTX 1050 Ti] [10de:1c82] (rev a1)
        Subsystem: Micro-Star International Co., Ltd. [MSI] GP107 [GeForce GTX 1050 Ti] [1462:8c96]
        Kernel driver in use: nouveau
--
02:00.1 Audio device [0403]: NVIDIA Corporation Device [10de:0fb9] (rev a1)
        Subsystem: Micro-Star International Co., Ltd. [MSI] Device [1462:8c96]
        Kernel driver in use: snd_hda_intel
```

Nel mio caso gli indirizzi da passare alla VM sono `0000:02:00.0' per il video e `0000:02:00.1' per l'audio. Successivamente possiamo automatizzare la configurazione in modo che il driver per la virtualizzazione venga avviato subito all'avvio della macchina. Quindi lo facciamo con una unit di SystemD.

```
# file: /etc/systemd/system/vfio-pci-bind.service

[Unit]
Description=Bind PCI devices to the virtio driver

[Service]
Type=oneshot
ExecStart=/usr/local/bin/vfio-bind 0000:02:00.0 0000:02:00.1

[Install]
WantedBy=multi-user.target
Before=libvirt-guests.service
```

12 Installare la VM

Come sistema operativo guest conglio Windows 8+, poichè il supporto dell'UEFI è migliorato e per esperienza trovo che Windows 7 sia più complicato da configurare (errori di drivers video).

12.1 Dispositivo di installazione

Per installare l'OS guest è necessario avere un immagine di installazione bootable perchè OVMF non supporta l'avvio di CDROM (o di altri sistemi legacy). Da un disco di installazione normalmente è possibile genereare un disco di installazione uefi utilizzando una memoria USB formattata in FAT32.

12.2 Installazione

Per creare la VM in libvirt è possibile utilizzare virt-manager che ha uno strumento con interfaccia grafica, in alternativa è possibile utilizzare virt-install dalla linea di comando.

```
% virt-install --connect qemu:///system \
    --name win8 \
    --memory 8169 \
    --vcpus sockets=1,cores=2,threads=2 \
    --cpu Skylake-Client \
    --metadata title="Windows 8.1" \
    --disk ~/vm_images/windows8.raw,cache=none \
    --network ...
```