Virtualización Directa a I/O
GPU Passthrough

JIMÉNEZ GONZÁLEZ JOSÉ EDUARDO

Universidad Nacional Autónoma de México
Facultad de Ingeniería

Marzo 28, 2020

Abstract

I/O virtualization provides great advantages in computing applications, a single physical device can be multiplexed, as many virtual devices bringing big improvements in hardware utilization, It also enables popular features in VMs just as live migration with seamless implementation, and convenient hook for interposing I/O operations such as better balance in workloads, replication, encryption, and security checks. Nonetheless there are big challenges of performance issues related to scheduling and prioritization. These problems can be minimized using clever approaches of hardware and software implementations. I/O virtualization remains an active area of research and development in the academia, and industry, due to its complexity and demand for new virtualized systems.

1. Introducción

Antes de comenzar, es necesario hacer un repaso de algunos conceptos claves, que nos ayudaran a entender mejor el tema. Empezando con el pilar de todo esto:

¿Qué es la virtualización?

La virtualización es el crear una versión virtual de un dispositivo o recurso, como un servidor, dispositivo de almacenamiento, red o incluso un computadora.

¿Qué es una maquina virtual?

Una máquina virtual es una representación virtual, o emulación, de un equipo de computo, incluyendo sistema operativo, dispositivos I/O, memoria, disco, etc. En éstas se denominada como *guest* al sistema operativo a emular, mientras que la máquina física en la que se ejecutan se conoce como el *host*

Todo esto es posible gracias a un software / hardware / firmware que se le denomina como *Hypervisor,* el cual separa los recursos de la maquina desde el mismo hardware, y los distribuye de forma apropiada que puedan ser usadas por una VM.

Existen dos tipos de Hypervisor

- Tipo-1: Estos corren de manera directa en el hardware del host, de forma que tienen mejor lactancia, y responden de mejor forma. Muchas veces se les denomina como *bare-metal hypervisors.* Algunos ejemplos de éste tipo son: Oracle VM Server, Microsoft Hyper-V.

- Tipo-2: Este tipo de hypervisor corre de forma convencional en el Sistema Operativo del *host*, tal y como lo haría cualquier otro programa. De forma que Sistema Operativo *guest* corre como un gran proceso. Algunos ejemplos de éste tipo son: Oracle VirtualBox, VMware, Parallels Desktop for Mac, QEMU.

Kernel-based Virtual Machine (KVM)

El *KVM* es un modulo de virtualización que se encuentra dentro del Kernel de Linux, de manera que éste pueda funcionar como un *bare*-metal *Hypervisor,* fue añadido a partir de la versión 2.6.20, y desde ahí ha ganado bastante popularidad, dado que es rápido, y viene ya de forma integrada en el kernel. Sin embargo éste requiere de extensiones de hardware especializadas para virtualización tales como Intel VT o AMD-V, las cuales ayudan al proceso de la virtualización.

Direct Memory Access (DMA)

El acceso directo a memoria como su nombre lo indica en español, permite a cierto hardware de la computadora acceder a la memoria de forma independiente, de modo que el CPU no se use para dicha transferencia. Actualmente es una característica esencial en las computadoras modernas.

¿Qué es la virtualización I/O ?

En palabras simples es la habilidad de multiplexar lógicamente dispositivos de E/S (I/O en inglés) virtuales, a un sólo dispositivo físico real I/O.

Es ampliamente usado en muchos escenarios de Cloud Computing, Data Centers, etc.

¿Por qué?

Mencionado lo anterior, la virtualización I/O permite una utilización mayor, ya que lógicamente se puede usar el mismo dispositivo físico en varias VMs, o en el caso de servers permite una instalación mas limpia, reduciendo incluso la cantidad de equipos a emplear. Ésta nos permite mejorar la seguridad de algunos sistemas, que más adelante se describirá sobre ello.

2. Beneficios

Para aprovechar los beneficios de la virtualización hay que entender que estos dependen de la *separación* lógica que existe entre los dispositivos I/O de una VM, y de su implementación física.

Ésta *separación* permite el multiplexeo en tiempo, y espacio de forma que la implementación de múltiples dispositivos lógicos I/O se puede dar como uno solo, o como un pequeño número de dispositivos físicos. Lo cual permite tanto a administradores como sistema automatizados, una mayor eficiencia, y utilización del hardware.

El mapeo entre la parte lógica, y física de estos dispositivos suele ser flexible, facilitando la portabilidad de las maquinas virtuales, incluso entre sistemas heterogéneos, la misma imagen de una maquina virtual puede ser utilizada en diferentes computadoras, con diferentes dispositivos I/O, ya que la conversión, o el manejo de diferentes interfaces estaría dada por ésta capa de virtualización I/O.

Esto a su vez permite características populares, o bien conocidas de las maquinas virtuales tales como la capacidad de suspender una maquina virtual, y resumirla posteriormente. O también la habilidad de mover una maquina virtual entre diferentes maquinas físicas (hosts) conocido como *live migration*.

Para ilustrar esto, por ejemplo supongamos un array de discos, y una maquina virtual, la cual puede migrar su unidad de disco, de manera transparente entre los diferentes que hay en el array, incluso mientras ésta se encuentra en uso, esto no solo tiene mayor utilidad, si no que también nos permite mayor disponibilidad de estos discos, suponiendo que varias computadoras, o maquinas virtuales pueden hacer uso de éste array, e incluso ayudar a moderar el balance de carga que hay entre los diferentes canales I/O. Por ejemplo en un sistema como ella. antes mencionado donde hay múltiples conexiones entre maquinas, e unidades de almacenamiento, la capa de virtualización nos puede ayudar a prevenir demores que puedan ocurren entre las conexiones debida a la contención causada.

Ésta habilidad de interponer flujos I/O a una maquina virtual, ha sido fuertemente usado en la industria.

Otra gran ventaja de la virtualización I/O es la agregación de dispositivos, donde múltiples dispositivos físicos pueden ser combinados en un sólo dispositivos lógico hacia nuestra maquina virtual. Un ejemplo de esto podría ser la combinación de múltiples unidades de almacenamiento, que se exportan como una sola.

Además de multiplicar su capacidad, también nos trae ciertas ventajas, como la capacidad de tener escrituras múltiples a estos discos, o la capacidad de tolerar fallos a un sistema, protegiendo nuestra información, ya que en caso de que uno falle, los demás no se verían afectados.

Pero no sólo para eso se ha usado la virtualización I/O también trae mejoras en la seguridad de un sistema, un ejemplo sería una transferencia de datos de un disco a otro, donde estos se encuentran encriptados, interponer el trafico de ésta transferencia permitirá implementar una capa de seguridad como algún firewall, o método de detención de intrusos, analizando paquete por paquete que se reciba.

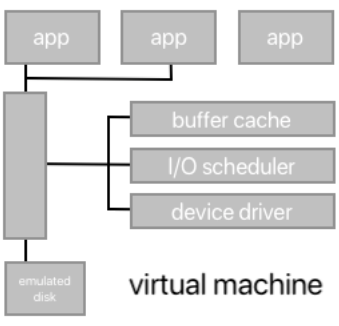
3. Inconvenientes

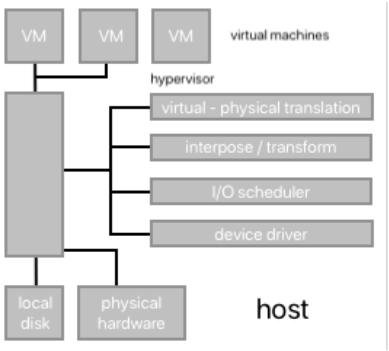
Problemas de desempeño

A pesar de que la virtualización de I/O provee grandes beneficios, también nos trae inconvenientes significantes, el más importante de ellos es conseguir un buen desempeño, incluso cuando éste puede ser flexible. El manejo complejo de recursos, es principalmente un problema relacionado con la planificación, y priorización de los mismos, problema que es introducido precisamente gracias a su concepto, el multiplexar varios dispositivos fiscos a una maquina virtual, afecta primordialmente su desempeño.

Los problemas a enfrentar son por ejemplo el definir la correcta semántica para los dispositivos virtuales, y sus interfaces. Dado que para una operación I/O requerimos conceptualmente que sea separada en dos *I/O stacks,* una para el *guest* que se encarga de manejar el hardware virtual, y otra para el *hypervisor* que se encuentra manejando todo el hardware físico.

Cuando una aplicación corriendo sobre una maquina virtual, solicita una operación I/O, por lo regular se realiza una llamada a sistema, ésta es inicialmente procesada por el I/O stack del sistema operativo *guest*, para posteriormente ser manejada por algún driver el cual intentará comunicarse con el dispositivo I/O virtual, éste es el momento donde el *hypervisor* intercepta. El *hypervisor* se encargará de agendar / planificar la solicitud de la maquina virtual, o en su caso de varias, al dispositivo físico I/O usualmente mediante otro driver manejado por el mismo. Cuando el dispositivo físico termina de procesar una solicitud I/O, los dos stack vuelven a ser nuevamente usados pero ahora en orden contrario. El dispositivo físico entrega una interrupción de que éste ha terminado, la cual es manejada por el hypervisor, éste determina a qué maquina virtual está asociado, y notifica a la misma mediante una interrupción virtual lo mismo.





Para reducir la sobrecarga, algunos hypervisor realizan la fusión de interrupciones virtuales en software, de forma similar a las optimizaciones de procesamiento por lotes de hardware que se encuentran en las tarjetas físicas, que retrasan la entrega de interrupciones con el objetivo de registrar una sola interrupción para varios eventos entrantes.

Para mejorar el rendimiento, algunos hipervisores paralelizan partes de el procesamiento, descargando el trabajo a núcleos de procesador adicionales. Sin embargo, cuando existe contención en el CPU, esto deja menos núcleos disponibles para ejecutar máquinas virtuales.

Hay que recordar que hecho de ver varias maquinas virtuales, como si cada una tuviera su propio hardware es meramente una ilusión, ya que en la realidad el *hypervisor* debe multiplexar entre el limitado hardware que tiene a su disposición. Si se llega a encontrar un contención como en la del CPU, esto resultaría en retrasos dentro de la planificación que tiene, *ya que el hypervisor* tiene como responsabilidad el prevenir que una maquina virtual monopolice los recursos, negando o causando retardos a otras.

Por otro lado, muchas veces se demanda el uso de dispositivos complejas interfaces, tales como tarjetas gráficas modernas, mientras que por el otro lado puede que sean idénticas al dispositivo físico.

El problema es garantizar que la virtualización preserve fielmente la semántica que el software espera de los dispositivos físicos. Por ejemplo, algunos sistemas de virtualización aumentan el rendimiento de I/O al aprovechar una memoria caché de búfer al nivel de hypervisor entre la máquina virtual y el almacenamiento físico.

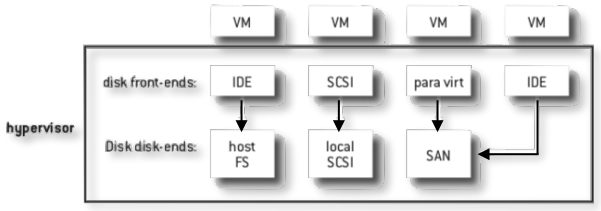
Aunque las lecturas de almacenamiento en caché no presentan ningún problema, las escrituras en caché infringirían la semántica de durabilidad en la que se basan los sistemas de archivos invitados, las bases de datos y otro software.

En un sistema nativo, la finalización de un proceso I/O indica que se ha confirmado una escritura.
Los hypervisor, por su parte deben usar una memoria caché de escritura, para conservar esta propiedad.

4. Intentos de solución

La forma típica de implementar la virtualización de I/O es estructurar el software en dos partes: un dispositivo virtual emulado que se exporta a la máquina virtual, y una implementación de *back-end* que se utiliza por el código de emulación del dispositivo virtual para proporcionar la semántica del dispositivo.

Los *hypervisor* modernos admiten una arquitectura de virtualización de I/O con una implementación de rejilla, en donde una máquina virtual puede seleccionar entre diferentes front-ends de emulación de interfaz de dispositivo virtual, así como varias implementaciones de *back-end* diferentes del dispositivo.



Para ello también es necesario una estructura especial de forma que se pueda abstraer como un disco. Una de ellas es la implementación de por ejemplo CDROM, o de SATA con tan solo accediendo a un archivo que contenga su imagen en un ISO, se puede aprovechar al máximo las capacidades de las maquinas virtuales tales como la de control entre versiones, o portabilidad.

Sin embargo siguen existiendo interfaces que simplemente son difíciles de emular, ya que requieren emular hardware que llega a ser complejo. Por ejemplo la interfaz IDE , la cual utiliza instrucciones OUT de ocho bits para comunicarse con el controlador de disco, el número de sector, la dirección del *búffer*, y la longitud que tiene el dato, esto requiere varias instrucciones de tipo trampa, al *hypervisor* que requieren para poder ser emuladas.

Para éste tipo de situaciones muchas veces lo que se realiza es simplemente cambiar a otra alternativa, como por ejemplo los discos SCSI, los cuales alcanzan la misma funcionalidad, pero con menos instrucciones “trampa, reduciendo en gran parte todos los dolores de cabeza que se genera emulando su interfaz.

Otras optimizaciones que se han hecho han sido directamente sobre hardware, optimizando la capa de virtualización en lugar de intentar emular algún dispositivo, a éste tipo se le llama paravirtualización, y en la práctica las más modernas plataformas de virtualización ofrecen soporte para compatibilidad de dispositivos tipo *legacy*, así como proveen éstas herramientas que ayudan a obtener mejor desempeño.

Un ejemplo podría ser el de una interfaz de disco *paravirtualizada* donde la emulación del dispositivo acepte comandos a través de un segmento de memoria compartida, entre el controlador y la misma, lo que permite la comunicación de comandos con una sobrecarga de prácticamente cero. El código de emulación simplemente pasa el comando a la implementación del *back-end*. Esto último es un intento de *pass-through.*

El verdadero *pass-through*, es un modo en el que las instrucciones del CPU virtualidad pueden hablar de forma directa con el dispositivo I/O, tal y como se tratase de una computadora física, lo cual nos lleva a eliminar la emulación, y la implementación back-end, permitiéndonos tener cero sobrecargas.

A pesar de que el *pass-through* puede sonar prometedor, también trae consigo bastantes limitaciones, así como problemas en su implementación que han alentando su desarrollo. La más obvia de ellas es la limitación que tiene a que cada dispositivo puede ser utilizado por una sola máquina virtual. Como resultado, se pierden muchas de las ventajas de portabilidad de la virtualización, junto con las ventajas clave, como la *live migration*.

El más grande problema con el *pass-through*, es la manera en que el dispositivo usa el *DMA*, ya que utiliza la noción de memoria que tiene el sistema *guest*, lo cual difiere de las direcciones de memorias reales en la que la maquina virtual reside.
Esto no solamente es incorrecto, si no que también trae grandes problemas de seguridad ya que eso permitiría que el dispositivo pueda acceder, y escribir a direcciones de memoria que le pertenecen al *hypervisor*, o a otra maquina virtual.

Para eliminar éstas limitaciones, y problemas, fabricantes han tenido que una vez más, optimizar su hardware en favor de la virtualización. La forma en la que lo han hecho es que el dispositivo esté "consiente" de esto, y así poder exportar múltiples interfaces a cada una de las maquinas virtuales con las que se desee trabajar. Dando a cada máquina virtual su propia copia de acceso *pass-through* al dispositivo.

También se requiere hardware especial para enfrentar los retos que tiene el *DMA*, éste sería una unidad de manejo de memoria, que se encargue de manera las direcciones de memoria a las correctas localidades que tiene cada maquina virtual.

Éste hardware adicional se le denomina *IOMMU*, o por sus siglas en ingles I*nput Output Memory Management Unit,* ésta se encuentra programada para cada maquina virtual que esté conectada a un dispositivo. Cada solicitud de *DMA* tiene que pasar por el *IOMMU*, el cual rutea de forma correcta la localidad de memoria real que necesita, o en su caso devuelve un error en caso de ser invalida.

La *IOMMU* permite al controlador en una maquina virtual, programar al dispositivo usando la noción que tiene de las direcciones de memoria virtuales, manteniendo además que el *hypervisor*, decida qué localidad de memoria le corresponde a cada maquina en la memoria física.

Aunque los IOMMUs pueden permitir de forma segura, y eficiente que los dispositivos I/O accedan directamente a la memoria de una máquina virtual, hay implicaciones para algunas de las operaciones de virtualización de memoria más sofisticadas en los *hypervisor* modernos que se basan en la reasignación dinámica de páginas.

Teniendo en cuenta características como el overcommitting de memoria, donde el hipervisor puede recuperar RAM, usando técnicas como la paginación a demanda en memoria secundaría, compresión de memoria, o el uso compartido de memoria transparente donde se pueden eliminar páginas duplicadas compartiéndolas en modo de solo lectura entre varias máquinas virtuales.

Éstas características requieren que ciertos accesos a la memoria de la máquina virtual que pueden causar errores, e invocar acciones del *hypervisor* antes de que se les permita seguir.

Por último recientes investigaciones han desarrollado nuevas técnicas de emulación IOMMU para proporcionar de manera eficiente vIOMMUs a los *guest*. Aún más significativamente, el mismo enfoque facilita una forma más flexible el pass-through a un dispositivo, donde una máquina virtual puede interactuar con un dispositivo I/O asignado directamente sin intervención del *hypervisor*.

Puesto que el g*uest* expone qué regiones de su memoria están implicadas en las operaciones del *DMA* a la *vIOMMU*, y el *hypervisor* puede modificar las asignaciones para otras regiones de memoria de forma segura. Al interponer solo en operaciones de *vIOMMU*, es posible lograr un rendimiento de I/O casi nativo, conservando la capacidad del *hypervisor* para administrar, reasignar, y hacer overcommit en memoria.

REFERENCIAS

1) David Davis (2011), "What is I/O Virtualization" recuperado el 27 de Marzo de 2019, de: https://www.petri.com/what-is-io-virtualization-iov
2) Carl Waldspurger, Mendel Rosenblum (2012), "I/O Virtualization" recuperado el 27 de Marzo de 2019, de: https://www.cse.iitb.ac.in/~cs695/papers/iovirt.pdf
3) Jun Nakajima (2007), "Intel Virtualization Technology Roadmap and VT-d Support in Xen" recuperado el 27 de Marzo de 2019, de: http://www-archive.xenproject.org/files/xensummit_4/VT_roadmap_d_Nakajima.pdf
4) Embedded (2009), "I/O Virtualization (IOV) & its uses in the network infrastructure" recuperado el 27 de Marzo de 2019, de: https://www.embedded.com/i-o-virtualization-iov-its-uses-in-the-network-infrastructure-part-1/
5) Rafal Wojtczuk, Joanna Rutkowska (2011), "Following the White Rabbit: Software attacks against Intel VT-d technology" recuperado el 03 de Abril de 2019, de: https://invisiblethingslab.com/resources/2011/Software%20Attacks%20on%20Intel%20VT-d.pdf
6) Thomas Krenn (2014), "Overview of the Intel VT Virtualization Features " recuperado el 05 de Abril de 2019, de: https://www.thomas-krenn.com/en/wiki/Overview_of_the_Intel_VT_Virtualization_Features
7) Hwanju Kim (2015), "I/O Virtualization" recuperado el 18 de Abril de 2019, de: https://es.slideshare.net/HwanjuKim/5io-virtualization
8) Pradeep Kumar (2016), "QEMU Disk IO" recuperado el 19 de Abril de 2019, de: https://www.slideshare.net/pradeepkumarsuvce/qemu-disk-io-which-performs-better-native-or-threads
9) Virtualization Deep Dive (2015),"Device & I/O Virtualization" recuperado el 20 de Abril de 2019, de: https://virtualizationdeepdive.wordpress.com/about/6-2/