Rethinking the Design of Virtual Machine Monitors

CARLOS MARTÍN FLORES GONZÁLEZ, Carné: 2015183528

Instituto Tecnológico de Costa Rica
Maestría en Computación
Sistemas Operativos Avanzados
Profesor: Francisco Torres Rojas, Ph.D

Un monitor de máquina virtual (*virtual machine monitor* VMM) es un sistema de software que particiona un máquina física en múltiples máquinas virtuales. Tradicionalmente, los VMMs crean replicas precisas de la máquina subyacente a través de una emulación fiel. Los VMMs soportan la ejecución de sistemas operativos invitados(*guest*) legados como Windows o Linux sin necesidad de modificaciones. Un gropo de investigación de la Universidad de Washington ha desarrollado Denali VM, con la premisa qeu es posible y útil el considerar una abstracción de VM que difiera de una máquina física. Los dos mayores resultados de este esfuerzo son paravirtualización e interimposición de hardware.

VMM Advantages: Recientemente, los VMMs han experimentado un renacimiento debido en gran parte al éxito de la monitor de máquina virtual de VMware. Algunos factores contribuyen a la popularidad actulas de los VMMs:

Implementación simple: en comparación con un SO completo como Linux o Windows. Los VMMs logran simplicidad al evitar la implementación de abstracciones de alto nivel como *sockets* TCP/IP y sistemas de archivos. Esta simplicidad hace que los VMMs sean muy adecuados para aspectos de fiabilidad y seguridad del sistema. Además, su simplificidad los hace más fácil de extender y modificar que los SO tradicionales.

Whole-system services: Una máquina virtual(VM) captura un sistema completo de software, incluyendo el SO y su grupo de aplicaciones. Esto es importante porque muchos servicios de interés transcienden mecanismos de encapsulación tradicionales como espacios de direccioens o procesos de SO.

Soporte para sistemas operativos invitados legados: La habilidad de correr múltiples SO legados en una sola máquina ha probado ser muy útil con el tiempo. Hoy en día los administradores de sistemas usan esta capacidad para consolidar varios servidores poco utilizados en una sola máquina y para forzar aislamiento para código inseguro o no confiable.

Rendimiento tolerable: Historicamente los VMMs han sufrido del inconveniente de un rendmiento lento en relacion con arquitecturas convencionales de sistemas. Conforme la velocidad de procesador ha aumentado, esta penalidad por virtualización se ha vuelto tolerable en muchas configracioens. Además, recientes avances en el diseño de VMMs han llevado a que el costo de virtualización sea menor.

Scaling a VMM:. La escalabilidad se refiere a la habilidad de correr muchas VMs en un sola máquina física. Dos observaciones motivaron el interés en la escalabilidad: (1) Han emergido dominios de aplicación que requieren mínimo o esporádico tiempo de procesador. (2) La ley de Moore ha producido una abundancia de poder crudo de CPU, habilitando colocación de muchos servicios para disminuir la sobrecarga administrativa. La investigación ha revelado que los VMMs tradicionales sufren de cuellos de botella en escalabilidad los cuales artificialmente reestringuen el número de VMs que un sistema puede soportar. Estos cuellos de botella existen porque la noción

de tiempo es más compleja en un VMM. Un VMM corre múltiples VMs en paralelo, asi que cada VM solo corre en el procesador de la máquina real por $1/N$ del total del tiempo de CPU en promedio. Este efecto crea una noción de *tiempo virtual* que avanza a una proporción diferente que el reloj físico. Conforme el número de VM incremeta, la separación entre tiempo virtual y tiempo físico aumenta, afectando adversamente cualquier aspecto de hardware dependiente de tiempo, incluyendo entrega de interrupciones y temporizadores. Para abordar estos desafíos, se propone paravirtualización. La idea clave de esta técnica es el exponer una arquitectura virtual de hardware que difiere de la arquitectura del hardware físico subyacente. Pequeños cambios a la arquitectura virtual son suficientes para eliminar los cuellos de botella artificiales que afectuan los sistemas tradicionales. En los 70s se proponen arquitecturas de máquina virtual *impuras* para mejorar el rendimiento o bien reducir la complejidad de la implementación de los sistemas. De igual forma los sistema microkernel exponen abstracciones de bajo nivel que son similares pero difieren de la interface de hardware. Estos enfoques tienen un inconveniente: los SO *guest* legados requieren modificaciones para correr en la arquitectura modificada. Uno de los objectivos principales del trabajo expuesto en el artículo es el de evitar grandes cambios a las bases del código legado. Este objectivo se logró al confinar las modificaciones arquitecturales para que encajen dentro de la capa de abstracción de hardware (*hardware abstraction layer* - HAL) del SO. El propósito principal de HAL es el de asegurar la portabilidad a través de un conjunto de arquitecturas de hardware, así, al confinar los cambios de arquitectura virtual en el HAL, portar un SO a Denali no es más complicado que portar ese SO a una nueva arquitectura física de hardware. En contraste con Denali, los sistemas microkernel introducen cambios arquitecturales más disruptivos, complicando el soporte para SO *guests* y sus aplicaciones. Uno de los SO que se ha portado utilizando la arquitectura Denali es NetBSD.

Architecture changes for scale: (1) la presencia de *idle loops* dentro de un SO *guest* plantean una barrera para la escalabilidad. En hardware físico, el SO ejecuta un ciclo inactivo mientras espera que algún evento de interés transcurra. En VMM, estos ciclos inactivos gastan ciclos útiles que podrían ser dedicados a otra VM. Para evitar esta degradación de rendimiento, Denali expone una instrucción *idle-with-timeout*, que hace que el SO *guest* rinda el CPU por un tiempo limitado. Esto permite utilización total del procesador mientras se asegura que una VM se despierta para manejar funcionalidad relacionada con el temporizador. (2) En hardware físico, la entrega de interrupciones ocurre inmediatamente después del arribo de algún evento de hardware como la llegada de un paquete de comunicación. En un VMM, preservar la entrega inmediata de interrupciones es difícil porque una VM corre solo un $1/N$ porciento del tiempo. Para tener en cuenta esta limitación, Denali usa un modelo de interrupciones asincrónicas en lote que encolan eventos de hardware virtual hasta que el *normal-schuler* quantum de la VM ocurra. Usando este enfoque Denali evita un 30 % de degradación de rendimiento para muchas VMs. (3) Los SO usan el arribo de interrupciones de temporizador para medir el paso del tiempo físico. Con cada VM limitada a $1/N$ porciento del CPU, y por lo tanto perdiendo la mayoría de las señales del temporizador físico, Denali expone un temporizador global físico que los SO pueden leer para aprender cuánto tiempo físico ha transcurrido desde el último quantum programado por la VM.

Architecture changes for simplicity: la arquitectura virtual de Denali omite varias caraterísticas raramente usadas como el BIOS, segmentación de hardware x86 y anillos de protección. Denali también reemplaza el llenado por hardware del TLB[1] por llenado por software, dando como resultado una implementación más simple y fácil. Hay instrucciones que se consideran no virtualizables porque se comportan diferente en modo usuario y en modo kernel, por esto Denali no hace ningún intento para emular hardware de forma precisa, por lo tanto al evita la complejidad inherente del manejo de estas instrucciones.

Extending a VMM. El rol tradicional de una VMM ha sido multiplexar una sola máquina a través de múltiples usuarios o aplicaciones. Una ventaja clave de implementar servicios dentro de un VMM es que tales servicios tienen la perspectiva de una máquina completa: captura el estado completo de el SO que corre y su grupo de aplicaciones. Esta perspectiva es importante porque muchos servicios de interés atraviesan mecanismos de

[1] Translation lookaside buffer

encapsulación tradicionales del SO como procesos o espacios de direcciones. Los servicios de las VMs también se benefician de la simplicidad de un VMM en relacion con un SO completo. Por esta razón, la migración de primitivas de VMs ha probado ser más fácil de implementar y mantener que migración de primitivas de procesos en SO completos. A pesar de la utildad de los servicios de la VMs, la realización de tales servicios dentro de la generalización de un VMM puede ser difícil. Implementar un servicio requiere de la habilidad de cambiar o extender partes de la implementación de un VMM. Desafortunamente los VMMs tradicionales soporta solo una implementación fija de abstracciones de hardware y como resultado, los diseñadores de servicios tiene que poner mucho esfuerzo en refactorizar las implementaciones existentes de VMMs. Una alternativa de modificar el código fuente es hacer ingeniería reverse a un VMM de caja negra como VMware. Sin embargo, los desarrolladores no diseñaron los VMMs convencionales con extensibilidad en menet, y por eso los VMMs carecen de los medio necesarios para soportar algunos servicios. μDenali, un sistema que facilita el desarrollo rápido de servicios nuevos en una VM, es una solución a este problema. El sistema logra extensibilidad a través de do componentes primarios: primero expone una serie de interfaces programables - los desarrolladores puede extender estas interfaces para modificar la implementación de abstracciones de hardware tales como discos virtuales sin necesidad de atascarse en los detalles de implementación del VMM. Segundo, μDenali soporta interposición de hardware, lo que permite que las extensiones de usuario sobreescriban la funcionalidad del sistema default.

Programatic API: Los SO tradicionales como Unix proveen interfaces programables que terceros puede usan para extender la funcionalidad del sistema. La interface programable de μDenali provee un nivel similar de extensión del dominio del VMM. En donde las extensiones tradicionales de SO proveen altos niveles de abstracción como en sistemas de archivos, las extensiones de VMM proveen abstracciones de bajo nivel como si fuera un dispositivo virtual de disco. La extensibilidad a través de una interface programable trae numerosos beneficios por ejemplo, la interface le quita a los programadores los detalles de la implementación de la VMM subyacente y también evita gastar esfuerzo en refactorizar. Utilizar interfaces abstractas también puede servir como base para una programación basada en componentes. El sistema puede portar extensiones a través de cualquier VMM que exponga la misma API programable. Aspectos del comportamiento de un VMM que se benefician de un mecanismo limpio de extensión: (1) Extensión de dispositivos de entrada/salida: mucho servicios deben de ser capaces de monitorear o modificar el comportamiendo de dispositivos de I/O tales como discos virtuales y la Ethernet. (2) Exposición del estado de la máquina virtual: para mejorar el rendimiento, μDenali pone en caché algo del estado de la VM dentro del hardware o dentro del VMM. Por ejemplo, los contenidos de registro actuales de una VM puede residir en el procesador. μDenali provee un API para extraer este estado, el cual es usado por los servicios. (2) Rastreo del no-determinismo: hay servicios que requieren información precesa del tiempo para eventos no-determinísticos como interrupciones de temporizador. (4) Control de máquinas virtuales: los programadores pueden usar la API programable de μDenali para iniciar, detener o matar un VM. La API programable consiste de un conjunto de interfaces basadas en lenguaje C.

Hardware interposition: Para soportar extensibilidad, es necesario separar la interfase de las abstracciones virtuales de hardware de sus implementaciones. La idea clave de este enfoque es transformar la VMM en un *framework* de ruteo de mensajes general. El sistema transforma todo los ventos de hardware virtuales, como lecturas de disco, en mensajes que el VMM rutea a un destino apropiado. Algunos eventos son manejados por implementaciones por default dentro del VMM y algunas son manejadas por código de extensión corriendo dentro de VMs. La naturaleza general del *framework* de ruteo de mensajes hace posible construir topologías arbitrarias de ruteo de mensajes. Así, una VM hija podría tener múltiples padres, cada uno de los cuales implementa alguna porción de su funcionalidad de hardware virtual.

Service examples: Se han implementado con un rango de servicios de VM por encima del mecanismo de extensión de μDenali. Además se ha usado μDenali para re-implementar servicios propuestos previamente tales

como migración de VM. Usando el API programable de μDenali la implementación de la migración requirió solamente de 289 líneas de código fuente C.

1. ¿CUÁL ES EL PROBLEMA QUE PLANTEA EL *PAPER*?

La pobre escalabilidad y extensibilidad de los monitores de máquina virtual que particiona una sola máquina física en varias máquinas virtuales.

2. ¿POR QUÉ EL PROBLEMA ES INTERESANTE O IMPORTANTE?

En cuanto a escalabilidad porque la investigación ha revelado que los VMMs tradicionales sufren de cuellos de botella en escalabilidad los cuales artificialmente reestringuen el número de VMs que un sistema puede soportar. Estos cuellos de botella existen porque la noción de tiempo es más compleja en un VMM. Un VMM corre múltiples VMs en paralelo, asi que cada VM solo corre en el procesador de la máquina real por $1/N$ del total del tiempo de CPU en promedio. En cuando a extensibilidad porque la implementación de servicios de VM es una tarea difícil y que ha recibido poca atención por grupos de investigación en la forma en cómo deberían de cooperar.

3. ¿QUÉ OTRAS SOLUCIONES SE HAN INTENTADO PARA RESOLVER ESTE PROBLEMA?

Los VMMs tiene una larga historia. IBM concibió la tecnología a medidos de los 60s y logró un éxito notable con su VM/370, el cual sirvió como un sistema de tiempo compartido y un plataforma de desarrollo de sistema operativo.

4. ¿CUÁL ES LA SOLUCIÓN PROPUESTA POR LOS AUTORES?

La investigación que llevan a cabo da como resultado el desarrollo de Denali VMM. Su premisa es que es posible y útil el considerar una abstracción de máquina virtual que difiera de una máquina física. Los dos principales resultados de este esfuerzo son:

- Paravirtualización: la arquitectura de hardware virtual difiere de la arquitectura física subyacente. Se aprovecha esta característica para construir un VMM escalable que soporta cientos de máquinas virtuales en ejecución concurrente.
- Interposición de hardware: se deja que los programadores extiendan el VMM con nuevas implementaciones de componentes de hardware virtual como discos virtuales o dispositivos Ethernet. Estos nuevos componentes pueden diferir dramáticamente de dispositivos nativos.

5. ¿QUÉ TAN EXITOSA ES ESTA SOLUCIÓN?

El mecanismo de los VMMs ha probado ser útil para la realización de un amplio rando de servicios. El trabajo en Denali VMM ha expandido la aplicabilidad de los VMMs al mejorar la escalabilidad y la extensibilidad de estos sistemas. En el futuro se espera que las innovaciones puedan surgir en la aplicación de tecnología de de máquinas virtuales en formas nuevas a innovadoras. Una oportunidad será aprovechar el fuerte aislamiento de máquinas virtuales y evitar conflictos de configuración entre aplicaciones ejecutando un solo sistema. Otra oportunidad puede ser usar VMMs para simplificar elas pruebas y *debugging* de software.