Un Driver de Disco Virtual Tolerante a Fallos

Resumen: *Uno de los principales modelos de servicio de la computación en la nube es el de Infraestructura como Servicio (IaaS). En ella se ofrecen recursos computacionales y de almacenamiento a través de tecnologías de virtualización. La virtualización de hardware es la tecnología más difundida, pero están surgiendo otras nuevas tales como la virtualización a nivel de Sistema Operativo y contenedores.*

Este artículo refiere a un driver de disco virtual para una nueva tecnología de virtualización distribuida (DRVS). En ella, los recursos computacionales, de red y de almacenamiento de cada máquina virtual se encuentran distribuidos en diversos nodos de un cluster. DRVS no opera sobre hardware real sino sobre dispositivos virtuales que utilizan los servicios de un Sistema Operativo subyacente. Si bien existen diversas tecnologías de virtualización de almacenamiento, el DRVS requiere de transparencia en la ubicación y capacidades de replicación y migración para mejorar la disponibilidad del servicio.

Palabras Claves: *Virtualización, Almacenamiento, Sistemas Distribuidos.*

Abstract: *One of the main service models of Cloud Computing is Infrastructure as a Service (IaaS). It offers computational and storage resources by using virtualization technologies. The most widespread technology is known as Hardware virtualization, but new ones are emerging such as Operating System level virtualization and Containers.*

This article refers to a virtual disk driver to be used by a new technology of distributed virtualization (DRVS). In a DRVS virtual machine the computational, storage and network resources are distributed in different nodes of a cluster. DRVS not operate on real hardware but on virtual devices using the services of an underlying Operating System. While there are several storage virtualization technologies, DRVS requires location transparency and replication/migration capabilities to improve service availability.

Keywords: *Virtualization, Storage, Distributed Systems.*

Mariela I. Alemandi, Oscar Jara

Lavaisse 610, Santa Fe - Facultad Regional Santa Fe, UTN – Telléfono (0342) 460 1579.

Mail: mariealemandi@gmail.com - oajara@gmail.com

Este trabajo ha sido realizado bajo la dirección del Mg. Pablo A. Pessolani en el marco del proyecto “Hipervisor de Máquinas Virtuales basado en sistema operativo de microkernel”.

INTRODUCCIÓN

La tecnología de Virtualización que fue desarrollada a finales de la década de los '60 [Galley S. (1969)] resurgió en los '90 como consecuencia de lo atractivo de sus características tales como la capacidad de consolidar múltiples Máquinas Virtuales (sus siglas en inglés, VM) en un único computador. Esta característica distintiva permite obtener una mayor eficiencia energética y proporcionar entornos de ejecución (dominios) seguros y aislados para las aplicaciones críticas.

Actualmente la tecnología de virtualización de hardware es la más difundida y conocida en la cual los dispositivos de hardware son virtualizados por un hipervisor. La virtualización de nivel de Sistema Operativo (sus siglas en inglés, OS), realiza la virtualización en un nivel más alto encapsulando aplicaciones de espacio de usuario dentro de Contenedores o Jaulas o Sistemas Operativos Virtuales (sus siglas en inglés, VOS [Hall, D., et al. (1980)]). Ejemplos de estos sistemas de virtualización son: VServer [Soltesz, S.(2007)], OpenVZ-(http://wiki.openvz.org) y LXC-(http://lxc.sourceforge.net/) .

Un VOS ofrece abstracciones y servicios a las aplicaciones de usuario, pero no gestiona hardware real; para ello utiliza dispositivos virtuales que le brinda una capa inferior de software de un OS subyacente en modo paravirtualizado.

Los OSs y los sistemas de bases de datos tienen versiones distribuidas que les permiten obtener mayor rendimiento e incrementar la disponibilidad de los servicios mediante técnicas de replicación y migración de procesos. Parece entonces que tendría sentido pensar en una versión distribuida de una tecnología de virtualización dado que se tiene idénticos requerimientos.

La tecnología de virtualización basada en VOS y los Sistemas Operativos Distribuidos (sus siglas en inglés, DOS), inducen a pensar en una convergencia entre ellas para lograr los objetivos mencionados y expandir las fronteras de una VM más allá de un simple computador, pudiendo abarcar múltiples nodos de un cluster. Esta tecnología denominada en inglés Distributed Resource Virtualization System (DRVS) [Pessolani, P., et al. (2012)] permite ejecutar múltiples instancias aisladas y seguras de un VOS distribuido en donde los recursos computacionales, de red y almacenamiento se encuentran dispersos en diversos nodos de un cluster (Figura 1). Un DRVS presenta capacidades de agregación (permite utilizar múltiples computadores de un cluster para el mismo VOS) y de particionado (permite ejecutar múltiples componentes de diferentes VOSs en un mismo nodo) en forma simultánea. Cada VOS distribuido se ejecuta dentro de un contexto de ejecución o dominio al cual se le denomina VM (debe entenderse éste como un término más amplio que el asociado a la virtualización de Hardware).

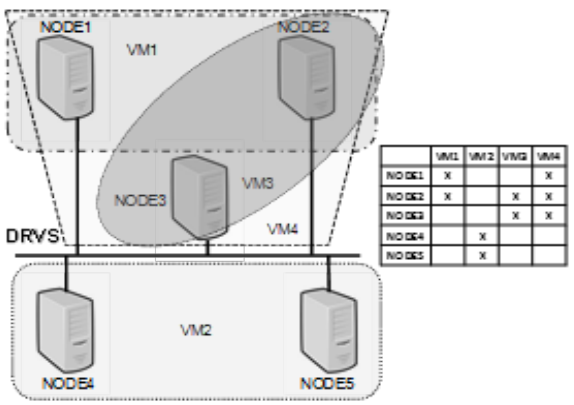

Figura 1. Topología de un DRVS.

En un entorno de ejecución típico, se asignan archivos regulares a dispositivos de bloques de las VMs. Cada OS invitado (Guest) le da formato a este dispositivo de bloques con su propio sistema de archivos (filesystem). Existen varios beneficios en mapear dispositivos virtuales como archivos:

1. Se puede sobre-utilizar el espacio de almacena-

miento dado que los archivos regulares generalmente solicitan almacenamiento en forma dinámica.

2. Resulta más sencillo tomar una instantánea (snapshot) y realizar una migración de una VM que utiliza archivos frente a otra que utiliza un dispositivo de bloques directamente.

3. Existe una amplia gama de herramientas basadas en archivos que facilitan la gestión del almacenamiento.

A continuación se presentan trabajos relacionados referentes a tecnologías de almacenamiento orientadas a la virtualización (virtualization-aware), como así también información básica (background) acerca de las tecnologías que fueron utilizadas para la realización del driver de disco tolerante a fallos.

Trabajos Relacionados

Se detallan aquí descripciones de las características de los trabajos referidos anteriormente que resultan representativos por su difusión y probado funcionamiento.

Protocolo iSCSI

iSCSI (Abreviatura de Internet SCSI) es un estándar (RFC 3720) que permite el uso del protocolo SCSI sobre redes TCP/IP. Permite realizar un acceso remoto a dispositivos de almacenamiento que soportan este protocolo mediante una infraestructura sencilla y barata. Como contrapartida sus críticos argumentan que este protocolo tiene un peor rendimiento que Fiber-channel ya que se ve afectado por la sobrecarga que generan las transmisiones TCP/IP.

Network Block Device (NBD)

NBD(http://nbd.sourceforge.net/) consiste en un módulo cliente en modo kernel y un módulo cliente en modo usuario. El módulo servidor gestiona el almacenamiento. El OS se comunica utilizando el módulo cliente de kernel con el módulo cliente de usuario realizando las peticiones de E/S. El módulo cliente en modo usuario transfiere esas peticiones de E/S al dispositivo de bloques remoto a través de una red mediante protocolo TCP. Ese dispositivo de bloques puede ser un archivo del OS remoto.

Distributed Replicated Block Device (DRBD)

DRBD [Reisner, P. (2005)] refiere a un conjunto de dispositivos de bloques diseñado como parte constitutiva de un cluster de Alta Disponibilidad (High Availability - HA). Esto se logra realizando una replicación o espejado de cada bloque del dispositivo a través de la red. Puede decirse que DRBD es como un sistema de almacenamiento en RAID-1 pero a través de la red.

DRBD trabaja con su propio protocolo de replicación basado en TCP. Una de sus principales desventajas es que los clientes sólo pueden acceder al servidor Activo.

Base Tecnológica

A continuación se describen con mayor nivel de detalle las tecnologías en las que se sustenta el desarrollo del Driver de Disco Virtual tolerante a fallos.

Minix over Linux (MoL)

Minix [Tanenbaum A., et al. (2006)] es un OS completo que fue desarrollado por Andrew S. Tanenbaum. Es de propósito general, de tiempo compartido, multitarea, basado en microkernel y que cumple con el estándar POSIX. Minix utiliza un mecanismo de Comunicación entre Procesos (Interprocess Communications - IPC) implementado en el microkernel. Las aplicaciones de usuario, los procesos del sistema e

incluso los Device Drivers se ejecutan aislados uno de otros y con los mínimos privilegios. Estas características mejoran sustantivamente la confiabilidad del sistema.

Las llamadas al sistema (POSIX system calls) se implementan utilizando IPC, en donde los argumentos son empacados en un mensaje y transferidos por el microkernel al destinatario. El destinatario desempaca e interpreta estos argumentos y lleva a cabo las operaciones requeridas, la respuesta retorna en otro mensaje. Las llamadas al sistema POSIX son atendidas por dos servidores, ellos son el Administrador de Procesos (Process Manager - PM) y el servidor de Sistema de Archivos (Filesystem Server - FS). Tanto el FS como PM realizan peticiones a otros procesos tales como Device Drivers utilizando transferencia de mensajes en la forma descripta previamente.

Tal como se mencionó anteriormente un DRVS se basa en varios VOSs distribuidos. Se eligió como la primera implementación de referencia de un VOS para que trabajará en forma distribuida a una versión adaptada de Minix a la que se denominó Minix over Linux (MoL) [Pessolani P. et al. (2011)].

Una versión experimental de MoL fue implementada utilizando mecanismos estándares de IPC Linux tales como POSIX Message Queues, Named Pipes, Unix Sockets o protocolo de red como sockets UDP/TCP sockets, RPC, etc. que brinda Linux, pero su desempeño no fue el esperado. Estos problemas de rendimiento llevaron a los autores a desarrollar un mecanismo de IPC incrustado en el kernel de Linux al que denominaron M3-IPC (Minix 3 Interprocess Communications) [Pessolani, P., et al. (2013)].

De manera similar a Minix, MoL está compuesto por un conjunto de servidores del sistema y de tareas que gestionan dispositivos (virtuales) que se comunican entre sí mediante IPC. Todos estos procesos se ejecutan en el modo-usuario de Linux y básicamente se comportan como un Minix virtualizado en modo usuario ejecutando sobre Linux.

Este artículo refiere al diseño e implementación de un Virtual Disk Driver para MoL (MoL-VDD) que se ejecuta como proceso en modo usuario sobre Linux y que utiliza M3-IPC como mecanismo de comunicación.

M3-IPC

La construcción de una tecnología de virtualización que pueda extender los límites de una VM más allá de un único computador (por razones de rendimiento, escalabilidad y/o disponibilidad) requiere que mecanismos de IPC puedan operar a través de una red en forma transparente para el programador.

M3-IPC se diseñó para permitir las comunicaciones entre procesos distribuidos de un VOS. Las comunicaciones entre procesos de un VOS no deben interferir (aislamiento) con las comunicaciones de procesos de otros VOSs que comparten el mismo cluster.

La semántica de M3-IPC es idéntica a la de Minix pero implementada en el kernel de Linux extendiendo su uso a procesos localizados en diferentes nodos de un cluster.

Entre sus principales características podemos mencionar al confinamiento de IPC (fundamental para la virtualización), redirección de mensajes cuando un proceso falla y el servicio se transfiere a otro proceso, o cuando el proceso migra de un nodo a otro. Estas características facilitan el desarrollo de un software de alta complejidad como lo es un sistema distribuido.

Como se mencionó previamente, para las comunicaciones entre nodos M3-IPC utiliza procesos proxies en modo-usuario. Esta característica, si bien tiene menos rendimiento que una implementación en el mismo kernel, no impone restricción alguna en cuanto a los protocolos de transporte/red que los proxies pueden utilizar. Los proxies pueden implementarse como

procesos simples, como pares de procesos o utilizando threads. En un nodo puede haber un solo proxy para comunicar con todos los demás nodos del cluster o puede haber un proxy por cada nodo, o cualquier configuración que el administrador decida. La versión actual de M3-IPC dispone de implementaciones de referencia utilizando protocolos TCP, UDP y TIPC.

The Spread Toolkit

Si bien M3-IPC soporta comunicaciones entre procesos localizados en diferentes nodos de un cluster, está orientado a un modelo Cliente/Servidor.

Para proporcionar tolerancia a fallos frecuentemente se usan técnicas de replicación de procesos y datos. Es recomendable entonces, utilizar un mecanismo de comunicaciones grupales (Group Communication Systems - GCS). Para facilitar el desarrollo, este GCS debe dar garantías sobre la entrega de los mensajes y respecto al orden de entrega de los mismos, aún en presencia de fallos de procesos o particiones de red.

Para desarrollar el DRVS se optó por utilizar los servicios que ofrece Spread Toolkit (http://www.spread.org/) el cual brinda entrega confiable, multicast con orden global, notificación de membresías y de fallos entre los nodos de un cluster. Spread es un toolkit open-source que puede utilizarse en aplicaciones distribuidas que requieren gran confiabilidad, alto rendimiento y comunicaciones robustas entre sus miembros. Spread soporta el modelo de membresía de grupo denominado Extended Virtual Synchrony (EVS) [Moser, L. E., et al. (1994)]. EVS puede gestionar particiones de red y su re-unión, así como las caídas de nodos y su rearranque. Los nodos de un cluster pueden caerse y luego rearrancar, las redes pueden partirse y luego volver a unirse. Estos eventos son detectados por Spread y reportados a los procesos previamente registrados en él permitiéndole llevar a cabo las acciones necesarias para superar esos eventos.

Como se mencionó en párrafos anteriores, un DRVS permite la ejecución de múltiples VOS en los cuales los procesos de usuario, los servidores y los drivers pueden ejecutarse en diferentes nodos de un cluster. A partir de esta posibilidad, se pueden presentar varios escenarios de ejecución que van desde uno en el cual los procesos de usuario comparten el mismo nodo que el servidor de archivos y MoL-VDD, o que cada uno de éstos ejecuta en diferentes nodos. Más aun, estos escenarios pueden cambiar dinámicamente por la migración de procesos provocada en forma automática u ordenada por el administrador.

Los servicios en la nube suelen utilizarse como plataforma para la ejecución de aplicaciones críticas que requieren de altos niveles de disponibilidad, escalabilidad y robustez. Por esta razón, MoL-VDD dispone de la característica de tolerancia a fallos lograda a través de las técnicas de redundancia de datos y de procesamiento. Esta redundancia se logra en forma transparente para la aplicación mediante la utilización de las facilidades que ofrecen los protocolos de comunicaciones M3-IPC y Spread.

MoL-VDD soporta esta clase de entorno distribuido de ejecución en forma dinámica y transparente en el cual los procesos de usuario, lo servidores y los drivers pueden migrar por cuestiones de disponibilidad o rendimiento. Estas características son muy apreciadas por los proveedores de IaaS porque incrementan la elasticidad de sus servicios y la utilización de sus recursos computacionales y de almacenamiento.

METODOLOGÍA

Generalmente, el componente de OS tipo Cliente/Servidor que utiliza los servicios de un Driver de Disco

es el Servidor Sistema de Archivos (Filesystem Server – FS). En un VOS, el FS es un proceso aislado que brinda servicios a procesos de usuario y se comunica con el Driver de Disco mediante la transferencia de mensajes.

En el DRVS, los procesos de usuario, el FS (MoL-FS) y el Driver de Disco (MoL-VDD) pueden ejecutar en diferentes nodos de un cluster, presentando los siguientes escenarios de operación (ver Figura 3).

A. Los procesos Clientes, MoL-FS y MoL-VDD se ejecutan en el mismo nodo (Nodo 0). Las transferencias de mensajes y datos se realizan utilizando M3-IPC, embebido en el kernel de Linux.

B. Los procesos Clientes y MoL-FS se ejecutan en el mismo nodo (Nodo 0) y MoL-VDD se ejecuta en otro nodo (Nodo 1). Las transferencias de mensajes y datos se realizan utilizando M3-IPC. Para las comunicaciones entre procesos de nodos diferentes, M3-IPC se basa en el uso de procesos proxies de modo usuario, que pueden utilizar diferentes protocolos de transporte/red.

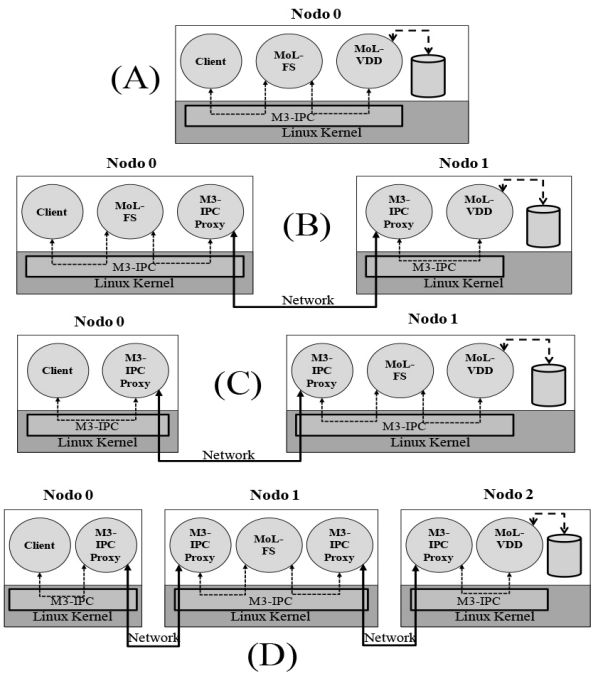

Figura 3. Escenarios de operación del Driver de Disco.

C. Los Clientes se ejecutan en un nodo (Nodo 0), MoL-FS y MoL-VDD se ejecutan en otro (Nodo 1).

D. Los procesos Clientes se ejecutan en uno o varios nodos (Nodo 0), MoL-FS se ejecuta en otro nodo (Nodo 1) y MoL-VDD se ejecuta un nodo diferente (Nodo 2).

En los escenarios de operación presentados sólo se hace referencia a un MoL-VDD sin replicación.

Para utilizar la característica de replicación se requiere que otro proceso MoL-VDD se ejecute en un nodo diferente (Figura 4). El esquema de replicación hace uso del modelo Primario/Backup [Budhiraja, N., et al. (1993)], en donde el MoL-FS sólo se comunicará con el MoL-VDD Primario utilizando M3-IPC (Figura 4A). En las actualizaciones que se efectúen sobre la imagen de disco (operaciones WRITE), el MoL-VDD Primario realizará un multicast atómico a los procesos MoL-VDD (Backup) de los otros nodos utilizando Spread. Todas las réplicas se actualizarán y finalmente el Primario retornará el código del resultado de la operación al MoL-FS.

En caso de producirse un fallo de caída (crash) de MoL-VDD Primario, Spread lo detecta e informa a los otros nodos de tal evento. Los MoL-VDD Backup ejecutan un algoritmo de selección de líder basándose en la lista de procesos Backup activos y sincronizados. El líder se transforma en el nuevo MoL-VDD Primario y adquiere el endpoint con el que MoL-FS se comunicaba utilizando M3-IPC (Figura 4B). El nodo donde ejecuta MoL-FS del mismo modo recibe la comunicación de la caída del MoL-VDD Primario original y también realiza la elección del líder, por lo que ahora conoce en que nodo se encuentra el endpoint del nuevo Primario.

Mientras suceden todos estos eventos, el proceso MoL-FS es bloqueado cuando intenta enviar mensajes a MoL-VDD Primario, hasta tanto se conozca su localización; condición ésta que produce el desbloqueo y da continuidad en forma transparente a las comunica-

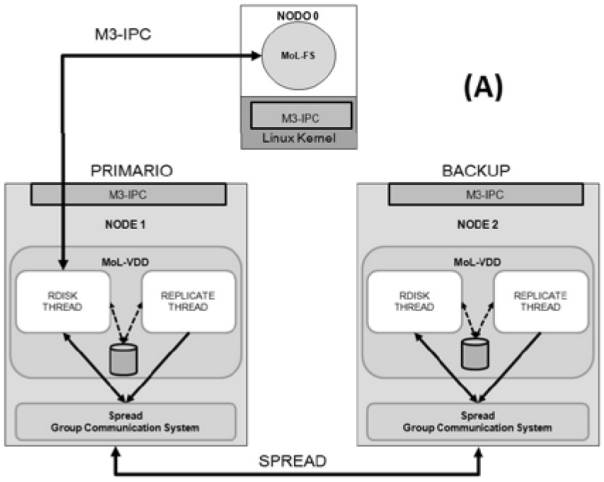

Figura 4 (A). Replicación con MoL-VDD.

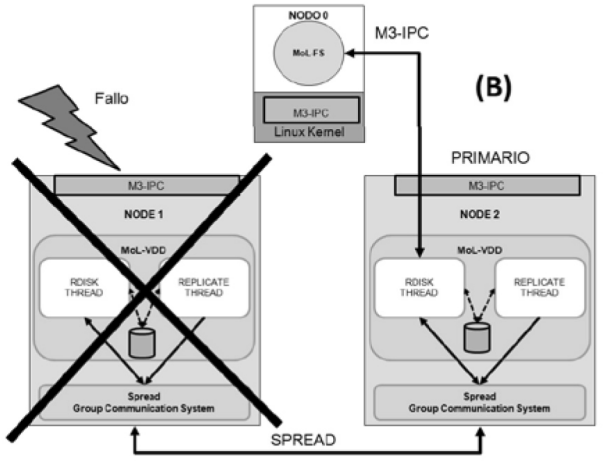

Figura 4 (B). Replicación con MoL-VDD.

ciones con el nuevo MoL-VDD Primario.

Todo proceso MoL-VDD está constituido por dos hilos de ejecución (threads). El hilo principal o rdisk() que es responsable de dar atención a las peticiones que recibe a través de M3-IPC y el hilo replicate() que es el responsable de replicar las operaciones de actualización de datos (WRITE) y de efectuar el protocolo de conformación de grupo de servidores Primary/Backup utilizando Spread. Esto incluye la incorporación de nuevos miembros al grupo (Join), la finalización explícita o por fallo de miembros (Disconnect), la detección de partición de red (Partition) y la re-unión de esas particiones (Merge).

Los hilos rdisk() y replicate() comparten variables de estado, por lo que para evitar condiciones de competencia se utilizan mutexes para el acceso serializado a las mismas.

El hilo rdisk() es responsable dar atención a las peticiones de E/S provenientes de MoL-FS utilizando el protocolo M3-IPC y sólo en las operaciones WRITE, envía un multicast de actualización a todo el grupo.

En hilo replicate() se implementa el esquema Primary/Backup de los servicios de MoL-VDD. En la versión actual se asume que las imágenes de disco del Primario y de los Backups son idénticas en el momento de comenzar su ejecución. Por ahora, tampoco admite la incorporación de nuevos procesos MoL-VDD al grupo una vez que comienza a operar el MoL-FS (que puede implicar actualizaciones de la imagen).

Cuando se inicia MoL-VDD se registra ante el servidor Spread. Si es el primer MoL-VDD, automáticamente se transforma en Primario y un semáforo habilita la ejecución del hilo rdisk() para atender a las peticiones de MoL-FS. Si no es el primer nodo queda a la espera de la información de estado que le suministrará el actual Primario mediante un multicast. Una vez sincronizado, el nuevo proceso MoL-VDD se transforma en Backup pero éste no habilita la ejecución del hilo rdisk(). De esta forma ningún proceso (incluido MoL-FS) podrá comunicarse con él utilizando M3-IPC. Los procesos MoL-VDD Backups solamente reciben mensajes a través de los multicast Spread que les envía el Primario.

Cuando el hilo rdisk() del Primario recibe una petición de WRITE, el mismo efectúa la escritura sobre su propia imagen de disco y luego envía un multicast atómico estable (Safe) de un mensaje tipo WRITE que transporta información sobre la localización, el

tamaño y los datos que deben escribirse en el dispositivo virtual. Todos los procesos replicate() (incluido el propio Primario) reciben el mensaje de WRITE. Cuando el proceso replicate() del Primario recibe su propio mensaje WRITE, Spread asegura que todos los procesos replicate() de los otros nodos lo han recibido, por esta razón no se requiere ningún tipo de mensaje de reconocimiento desde los procesos Backups. Esto asegura que el mensaje WRITE fue recibido en los proceso replicate() de los Backups (aunque esto no asegura que la operación de escritura se haya efectuado correctamente en ellos). Este orden de entrega de mensajes aparentemente sincrónico se asegura porque Spread implementa el modelo EVS.

RESULTADOS

Para evaluar el rendimiento de MoL-VDD y su driver BUSE se desarrollaron dos tipos de micro-benchmarks (Figura 5):

- Tests Locales: El cliente y el servidor se ejecutan en el mismo nodo.

- Tests Remotos o en Cluster: El cliente y el servidor se ejecutan en diferentes nodos.

La evaluación de rendimiento se realizó conjuntamente con NBD con fines comparativos.

Los micro-benchmarks utilizaron los comandos time y dd para realizar las transferencias de datos y tomar las mediciones. Se utilizaron archivos de 300 MBytes localizados en un disco RAM de Linux a fin de evitar la latencia propia de los discos rígidos. Los equipos utilizados fueron PCs Intel(R) Core(TM) i5 CPU 650 @ 3.20GHz, cache de 4096 KB y 4 GBytes de RAM y una red con un LAN switch de 1 Gpbs por puerto.

Cuando en la Figura 5 se hace referencia MoL-VDD significa que la transferencia fue realizada desde un programa cliente que utiliza M3-IPC para transferir hacia y desde el driver. Cuando se hace referencia a BUSE significa que la transferencia fue realizada por el cliente a través del driver BUSE. Si se menciona single indica que el servidor no trabaja replicado. Si se menciona replicated, indica que hay un segundo MoL-VDD Backup en otro nodo realizando las réplicas de las escrituras.

Micro-Benchmarks Locales

En la Figura 5 (A) se presentan los resultados de los tests en los que el cliente y servidor se ejecutan en el mismo equipo.

Los resultados muestran una diferencia importante de MoL-VDD dado que utiliza M3-IPC que se encuentra optimizado para su ejecución local.

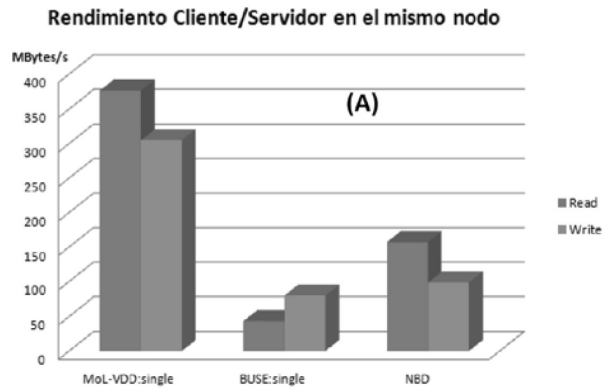

Figura 5 (A). Resultado de los Micro-benchmarks.

Micro-Benchmarks Remotos

En la Figura 5 (B) se presentan los resultados de los tests en los que el cliente y servidor se ejecutan en diferentes nodos de un cluster.

Los resultados muestran una diferencia importante a favor de NBD. Esto se debe a que M3-IPC utiliza procesos proxies en modo usuario para las comunica-

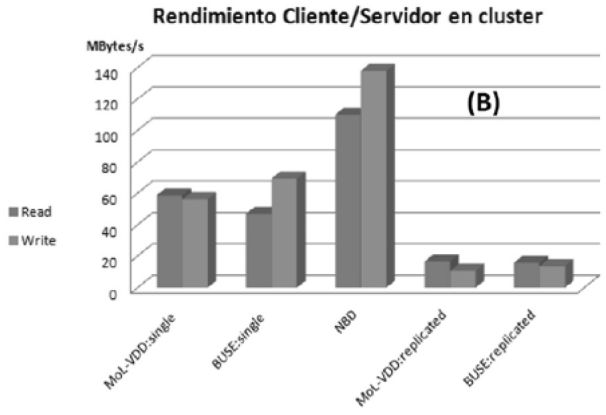

Figura 5 (B). Resultado de los Micro-benchmarks.

ciones con nodos remotos, lo que implica 4 cambios de contexto adicionales entre cliente y servidor por cada transferencia de bloque de datos. Por otro lado, se puede apreciar el impacto sobre en el rendimiento al habilitar la facilidad de replicación en MoL-VDD, básicamente producido por el protocolo de multicast atómico entre nodos.

DISCUSIÓN

MoL-VDD fue desarrollado como componente de un sistema de virtualización distribuido DRVS. Por ser un sistema distribuido, se requiere de características de redundancia para tolerancia a fallos y de soporte de migración de procesos. Para utilizar los servicios de MoL-VDD se debe hacer uso del protocolo M3-IPC desarrollado para el DRVS. Para extender su utilización a Linux, se desarrolló un módulo BUSE que permite que Linux use el dispositivo sin requerir la utilización de M3-IPC. NBD es un software que realiza una función similar a MoL-VDD que dispone de un componente cliente y un componente servidor en modo usuario que le permite a un Linux cliente acceder a un dispositivo remoto (imagen de disco) en un Linux servidor. NBD no soporta redundancia.

Cuando se requiere de mayores niveles de disponibilidad, MoL-VDD puede utilizarse en un esquema Primary/Backup. Los procesos que conforman el grupo MoL-VDD se comunican entre sí mediante el protocolo de comunicaciones grupales Spread. La versión actual de MoL-VDD es aún un prototipo en desarrollo que sólo soporta fallos de caídas (crashes) de procesos y nodos. Aún no están soportados fallos de partición de red y re-unión de red. El paquete DRBD disponible para Linux ofrece replicación (mirroring) de un disco en forma remota, pero carece de la posibilidad de que los datos puedan ser accedidos transparentemente después de un fallo del Primario. Esta es una característica diferencial de Mol-VDD porque utiliza M3-IPC, el cual permite redirigir las comunicaciones en forma transparente (a la aplicación Cliente) hacia el servidor Backup transformado en el nuevo servidor Primario luego del fallo.

En el prototipo actual se asume que tanto el servidor Primario como los servidores Backup son iniciados concurrentemente con imágenes de disco idénticas. La conexión de los clientes no se permite hasta tanto todos los procesos servidores estén iniciados. Quedan pendientes de análisis y desarrollo el control de versiones de imágenes de disco, la sincronización de imágenes entre el Primario y un nuevo Backup, el soporte de múltiples imágenes de dispositivos, la atención de peticiones concurrentes mediante múltiples hilos de ejecución y el soporte de compresión y cifrado en las transferencias de datos por la red entre otros.

MoL-VDD puede utilizar cualquier tipo de archivo que Linux ofrezca como imagen de disco, esto es: archivos regulares, archivos de dispositivos de bloques, archivos localizados en servidores remotos como NFS, archivos en discos RAM, pendrives, etc.). Además soporta la ejecución de múltiples servidores

en el mismo nodo (usando diferentes endpoints o configurándolos en diferentes VMs) permitiendo un mayor grado de concurrencia y explotación de los recursos. Mol-VDD admite diferentes escenarios donde puede ejecutarse en el mismo nodo donde reside el proceso que utiliza sus servicios o localizarse en nodos diferentes.

A diferencia de DRBD, MoL-VDD puede utilizarse en un esquema de un servidor Primario y múltiples servidores Backups incrementando así el nivel de redundancia y por lo tanto la disponibilidad del servicio.

Con un esquema de redundancia en el servicio de almacenamiento se evita un único punto de fallo en el sistema de virtualización, se dispone de respaldo de los datos en múltiples dispositivos de almacenamiento físicamente separados, se evitan tiempos muertos de producción por caídas en un servidor y finalmente se mejora la flexibilidad en el mantenimiento de la infraestructura ya que se permite retirar servidores en operación sin interrumpir el servicio.

CONCLUSIONES

La computación en la nube ha llevado a las aplicaciones críticas a ejecutar sobre máquinas virtuales imponiéndoles necesidades de alto rendimiento, escalabilidad, alta disponibilidad, aislamiento y elasticidad. Se presentó en este artículo un modelo de driver de disco para ser utilizado por una nueva tecnología de virtualización distribuida orientada a satisfacer esas necesidades. A fin de tolerar fallos e incrementar los niveles de disponibilidad, los servicios que el driver ofrece pueden replicarse en uno o más nodos de un cluster y donde los fallos se ocultan a los servidores que lo utilizan.

RECONOCIMIENTOS

Los autores agradecen el apoyo y asesoramiento recibido por el Departamento de Ingeniería en Sistemas de Información y la SCYT de UTN Facultad Regional Santa Fe, y el financiamiento recibido desde la SCYT de Rectorado de UTN en el marco del proyecto UTN 1766.

REFERENCIAS

Galley S. (1969). PDP-10 Virtual Machines. ACM SIGARCH-SIGOPS Workshop on Virtual Computer Systems.

D. Hall, D., Scherrer, J. Sventek, (1980) "A Virtual Operating System", Journal Communication of the ACM.

Budhiraja, N., et al. (1993). The Primary-Backup Approach. Distributed systems (2nd Ed.), Sape Mullender (Ed.). ACM Press/Addison-Wesley Publishing Co., New York, NY, USA 199-216, 1993.

Moser, L. E., Amir, Y., Melliar-Smith, P. M., Agarwal, D. A. (1994). Extended Virtual Synchrony. In Proceedings of the IEEE 14th International Conference on Distributed Computing Systems (Poznan, Poland). IEEE Computer Society Press, Los Alamitos, CA.

Reisner, P. (2005). DRBD v8 - Replicated Storage with Shared Disk Semantics, Proceedings of the 12th International Linux System Technology Conference. Hamburg, Germany.

Tanenbaum A., Woodhull A., (2006). Operating Systems Design and Implementation, Third Edition, Prentice-Hall.

Soltesz, S., Pötzl, H., Fiuczynski, M, Bavier, A., Peterson, L (2007). Container-based operating system virtualization: a scalable, high-performance alternative to hypervisors, EuroSys 2007.

Soundararajan, G. et. al (2008). Context-Aware Prefetching at the Storage Server, USENIX Annual Technical Conference, Boston, MA, USA, June 22-27, 2008.

Pessolani P.; Jara O., (2011). Minix over Linux: A User-Space Multiserver Operating System, Computing System Engineering (SBESC), Florianópolis-Brazil.

Pessolani, P., Cortes, t., Gonnet, S., Tinetti. F. (2012). Sistema de Virtualización con Recursos Distribuidos. WICC 2012. Pág. 59-43. Argentina.

Pessolani, P., Cortes, t., Gonnet, S., Tinetti. F. (2013). Un mecanismo de IPC de microkernel embebido en el kernel de Linux. WICC 2013. Pág. 11-15. Argentina.

FECHA DE PRESENTACIÓN: 22/02/16
FECHA DE APROBACIÓN: 10/08/16