Un Servidor de Sistema de Archivos para un Sistema de Virtualización Distribuido

Resumen
Cuando se piensa en infraestructuras en la nube, el rol del almacenamiento tiene un protagonismo especial para lograr flexibilidad, versatilidad y escalabilidad. Deben ofrecer sistemas de archivos simples, escalables, robustos y transparentes, de forma tal de no adaptar o modificar las aplicaciones o los procedimientos que los utilizan o gestionan. Dentro de las nuevas tecnologías de virtualización que ofrecen recursos de cómputo, comunicación y almacenamiento distribuidos en varios nodos de un cluster, este artículo presenta el diseño e implementación de un servidor de sistema de archivos para un Sistema de Virtualización Distribuido.

Abstract
When we think about cloud infrastructure, the main role of storage has a special meaning in order to achieve flexibility, versatility and scalability. They must offer simple, scalable, robust and transparent file systems, so that not adapt or modify the applications or procedures that use or manage them. As new virtualization technologies offer computing, communication and storage resources distributed in several nodes of a cluster, this article presents the design and implementation of a file system server for a Distributed Virtualization System.

Palabras clave*: "sistema de archivos", "virtualización", "sistemas distribuidos"*

INTRODUCCIÓN

La tecnología de virtualización, tiene la habilidad de consolidar múltiples Máquinas Virtuales (VM: Virtual Machine) en una única computadora física. Esta característica, permite aumentar la eficiencia energética y proporcionar entornos de ejecución seguros y aislados para aplicaciones críticas.

Un Sistema Operativo (OS: Operating System) ordinario, es la capa de software que se encuentra entre el hardware y las aplicaciones. Es posible construir un OS virtual (VOS: Virtual OS) [1] que brinde sus servicios a las aplicaciones pero que no gestione directamente sobre el hardware sino solicitando servicios al OS subyacente. A esta tecnología se la denomina Virtualización de Sistema Operativo. Como ejemplos de este tipo de virtualización se puede mencionar a User Mode Linux (UML) [2] y Minix over Linux (MoL) [3]. Otra tecnología muy utilizada en la actualidad es la Virtualización basada en Sistema Operativo, la cual permite encapsular aplicaciones en espacio de usuario, en entornos aislados de ejecución denominados comúnmente: Contenedores. Como ejemplos de Virtualización basada en OS se puede mencionar a VServer [4], OpenVZ [5], Zap [6].

En general, los OSs y Servidores de Bases de Datos cuentan con versiones distribuidas con el objetivo de lograr mayor rendimiento, escalabilidad y disponibilidad. De igual forma, un entorno de virtualización distribuido, permitiría aumentar su rendimiento, escalabilidad y disponibilidad.

Si un Contenedor pudiese expandirse a varios nodos de un cluster, se podría ejecutar dentro de él un VOS distribuido (DVOS: Distributed Virtual Operating System). La tecnología capaz de ofrecer Contenedores Distribuidos (del inglés DCs) se denomina Sistema de Virtualización Distribuido [7] (DVS: Distributed Virtualization System).

Los DCs son entornos de ejecución aislados que pueden abarcar uno o más nodos del cluster. El DVS solo admite las comunicaciones entre procesos que se encuentran ejecutando en el mismo DC aunque pueden hacerlos ubicados en diferentes nodos del cluster de virtualización.

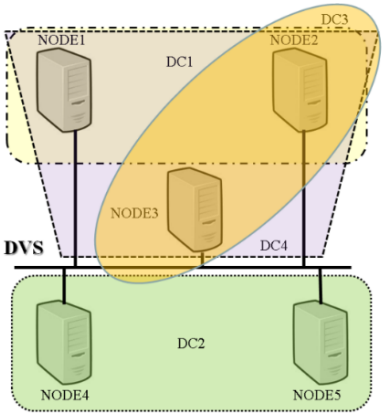
Un DVS está basado en componentes cuyos recursos pueden estar distribuidos en varios nodos de un cluster (Fig. 1).

Así, un DVOS que se ejecuta dentro de un DC puede abarcar más de un nodo (agregación) y los componentes, servicios y procesos de diferentes DCs pueden compartir un mismo nodo (consolidación).

Cada VOS le da formato a un dispositivo de bloques con su propio sistema de archivos (filesystem). Existen varios beneficios en mapear dispositivos virtuales a archivos regulares:

- Se puede utilizar en forma más eficiente el espacio de almacenamiento dado que los archivos regulares generalmente solicitan la asignación de bloque en forma dinámica.

- Resulta más sencillo tomar una instantánea (snapshot) y realizar una migración de un VOS o uno de sus procesos que utiliza archivos frente a otro que utiliza un dispositivo de bloques directamente.
- Existe una amplia gama de herramientas basadas en archivos que facilitan la gestión del almacenamiento.



	DC1	DC2	DC3	DC4
NODE1	X			X
NODE2	X		X	X
NODE3			X	X
NODE4		X		
NODE5		X		

Figura 1. Topología de un DVS.

En este artículo se presentan detalles de diseño e implementación de un Servidor de Sistema de Archivos FAT para un DVOS de un DVS. Se recurrió a una biblioteca denominada FatFs [8] para integrar el soporte de sistemas de archivos FAT a un servidor preexistente.

La organización del resto del artículo es la siguiente: en la Sección 2, se realiza una breve descripción de los trabajos relacionados que conforman el sustento tecnológico del presente trabajo. En la Sección 3, se describe el diseño e implementación del Servidor de Archivos. En la Sección 4 se presentan diferentes escenarios que se conformaron para realizar la evaluación de rendimiento del prototipo desarrollado. Finalmente, en la Sección 5 se resumen los aportes del presente trabajo en sus conclusiones y su potencialidad para desarrollar trabajos de investigación futuros.

TRABAJOS RELACIONADOS

Existe una amplia variedad de implementaciones de sistemas de archivo preparados para trabajar en entornos virtualizados. Entre las más populares se encuentra NFS [9] que proporciona acceso local o remoto en forma transparente a sistemas de archivo. De esta manera, los clientes pueden compartir el mismo host que el servidor, o bien, el cliente y el servidor pueden estar en diferentes hosts unidos por una red IP. NFS fue diseñado para ser portable a diferentes OSs y arquitecturas de hardware.

En NFS, tanto el cliente como el servidor han sido integrados en el kernel de Linux como módulos (LKM: Loadable Kernel Modules) pero, están disponibles servidores en espacio de usuario como por ejemplo UNFS [10] y clientes tal como HSFS [11] que utiliza FUSE [12].

RadFS [13] es un prototipo de sistema de archivo virtual que ahorra espacio en disco mediante la compartición de datos usando el enfoque copy-on-write (COW). Proporciona almacenamiento para Máquinas Virtuales (VMs) de una manera rápida y sencilla. También soporta instantáneas (snapshots) y la habilidad de incluir un sistema de archivos virtual en otro en forma recursiva. RadFS está basado en EXT3 y FUSE los cuales ofrecen una abstracción a nivel VFS (Virtual FileSystem) del kernel de Linux. RadFS fue construido específicamente pensando en tecnologías de virtualización.

VirtFS [14] también fue propuesto como un sistema de archivos apto para virtualización. Es una interfaz de sistema de archivo paravirtualizada para el entorno KVM de Linux. Fue utilizada como solución para compartir archivos de hosts a través de sistemas de archivos genéricos de red tales como NFS o CIFS, pero optimizada para virtualización.

Minix y Minix over Linux (MoL)

La elección para la primera implementación de un VOS que se ejecutara en el DVS, fue una adaptación del OS Minix, denominada Minix over Linux (MoL) [3] ejecutando en espacio de usuario.

Luego de varias pruebas con diferentes mecanismos de comunicaciones con resultados no satisfactorios, los autores decidieron desarrollar un mecanismo IPC embebido en el kernel de Linux basado en el IPC de Minix al que denominaron M3-IPC [15].

Minix [16] es un OS completo, de propósito general, multitarea, multiservidor, basado en microkernel y compatible con POSIX desarrollado desde cero por Andrew S. Tanenbaum. Ha sido ampliamente utilizado en universidades para casos de estudio e investigaciones basado en una arquitectura de tipo Cliente/Servidor donde cada proceso se ejecuta en un entorno aislado y se comunica con otros procesos mediante la transferencia de mensajes (IPC).

Las aplicaciones de usuario hacen llamadas al Sistema utilizando transferencia de mensajes, empaquetando los argumentos de la función y los resultados de la ejecución al igual que ocurre en una llamada RPC.

Los dos servidores que atienden solicitudes de programas en espacio de usuario, usando llamadas al sistema POSIX, son el Administrador de Procesos (PM) y el Servidor de Sistema de

Archivos (FS). A su vez, el FS y el PM también realizan peticiones a las tareas que controlan los dispositivos mediante IPC.

En este contexto, MoL podría considerarse como un conjunto compuesto de procesos servidores ejecutando enteramente en espacio de usuario como procesos Linux, logrando así un Minix virtual.

En éste trabajo se presenta un prototipo de servidor de sistema de archivos llamado MoL-FAT, para soportar sistemas de archivos tipo FAT, ejecutando en modo de usuario y accedido mediante M3-IPC.

M3-IPC

Para desarrollar tecnologías de virtualización distribuidas se requiere de herramientas de comunicación adecuadas. M3-IPC fue diseñado para permitir una comunicación en forma transparente entre procesos que se encuentran localizados en diferentes nodos de un cluster. Su semántica imita la del IPC de Minix pero implementada como un co-kernel en Linux.

Las APIs de M3-IPC, son de tipo cliente/servidor, sincrónicas, potentes y sencillas de usar. Las mismas se podrían clasificar en:

- *APIs de Comunicación*: Son aquellas relacionadas con la transferencia de mensajes. Éstas no incluyen ninguna referencia al DC ni al nodo donde se encuentra el proceso. Las únicas referencias de direccionamiento son los denominados ***endpoints*** que identifican a cada proceso dentro de un DC. M3-IPC soporta las comunicaciones con procesos aún después de haber migrado de nodo. De igual forma soporta el mantenimiento de las comunicaciones de los procesos clientes con procesos de tipo Primario que han sido reemplazados por otros que previamente eran de tipo Backup.
- *APIs de Administración*: Son las relativas a los DVS, DCs, proxies, nodos, y administración de procesos, permitiendo a los programadores especificar la topología de sus aplicaciones.

Para la transferencia de mensajes entre los distintos nodos de un cluster, M3-IPC utiliza procesos proxies, generalmente implementados en modo usuario. M3-IPC fue desarrollado para lograr una alta performance para procesos en el mismo o en diferentes nodos del mismo pertenecientes al mismo DC.

MoL-FAT

En un DVS los procesos clientes, servidores como Sistemas de Archivo (FS) y Gestores de Disco pueden ejecutar compartiendo el mismo nodo, o bien, pueden hacerlo en nodos diferentes. Esta característica es muy valorada, ya que brindan gran flexibilidad, mayor disponibilidad y escalabilidad de servicios, logrando un aprovechamiento más eficiente de los recursos de almacenamiento, de red y de cómputo.

En el prototipo del DVS actual existen dos VOS modelos. Un VOS de tipo Unikernel [17] y un DVOS denominado en MoL[3]. Un último rasgo significativo es que MoL hace fácilmente posible utilizar bibliotecas de código abierto de fácil acoplamiento al código preexistente. Por esta razón, es posible que MoL pueda soportar diferentes sistemas de archivo como FAT, EXT2 y otros.

En este trabajo, la biblioteca elegida para crear el prototipo MoL-FAT fue FatFs [8]. Si bien MoL-FAT fue diseñado como un servidor para ser utilizado como componente de MoL, complementariamente se desarrolló una adaptación FUSE para permitir que aplicaciones comunes de Linux accedan a MoL-FAT en forma local o remota al sistema de archivos en forma transparente.

Biblioteca FatFs

FatFs es una biblioteca de sistema de archivos FAT/exFAT genérico para pequeños sistemas. Está escrito en ANSI C (C89) y cuenta con una capa de E/S completamente separada del disco. Esto lo hace independiente de la plataforma.

Algunas de sus características son:

- Sistema de archivos FAT/exFAT compatible con DOS, Windows y Linux.
- Independiente de la plataforma. Fácilmente portable
- Varias opciones de configuración como, por ejemplo, nombres largos de archivos, sistema exFAT, tamaño de sector variable, múltiples volúmenes, etc.

Como toda biblioteca, FatFs cuenta con una interfaz (API) que está estructurada en los siguientes grupos de funciones:

- Acceso a archivos: *f_open*(), *f_close*(), *f_read*(), *f_write*(), etc.
- Acceso a directorios: *f_opendir(), f_closedir(), f_readdir(),* etc.
- Administración de archivos y directorios: *f_stat(), f_rename(), f_mkdir(), f_chdir()*, etc.
- Administración de volúmenes y configuración del sistema: *f_mount(), f_mkfs(), f_fdisk(),* etc.

FUSE

El Sistema de Archivos en Espacio de Usuario (Filesystem in User-Space: FUSE) [12] es un mecanismo del que disponen algunos OSs que habilita a usuarios sin privilegios, crear sus

propios sistemas de archivos sin modificar el código del kernel. Esto se logra ejecutando el código del sistema de archivos en espacio de usuario, mientras el modulo FUSE hace de "puente" con las interfaces del kernel (Ver Fig. 2). Esto lo vuelve especialmente útil al momento de crear sistemas de archivos virtuales y prototipos.

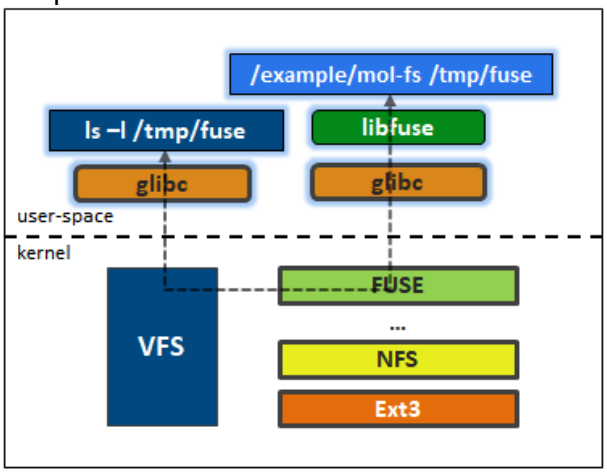


Figura 2. Camino de FUSE (obtenido de [18]).

FUSE está compuesto por un módulo de kernel y una biblioteca que se utiliza por una aplicación que gestiona el sistema de archivos ejecutando en espacio de usuario para invocar y ser invocado por el módulo.

A su vez, FUSE permite al usuario montar el sistema de archivos para que luego los accesos a éste sean enviados a través del módulo de kernel hacia la función callback correspondiente (open, read, write, etc.) implementada en modo usuario.

Se desarrolló la adaptación a FUSE de MoL-FAT para habilitar su utilización en forma transparente desde aplicaciones típicas de Linux a los efectos de comprobar su rendimiento y para demostrar su versatilidad.

CARACTERÍSTICAS DE MoL-FAT

MoL-FAT es el resultado de combinar diferentes tecnologías existentes para conformar un servidor de archivos de un DVOS de un DVS. El servicio de sistema de archivos debe ser versátil y transparente en lo que refiere a mecanismos de comunicación con los procesos clientes y con el resto de servidores y tareas que componen un DVOS.

El Protocolo MoL-FAT

Las aplicaciones basadas en MoL, hacen sus llamadas al sistema tal como si realizaran una Llamada a Procedimiento Remoto (RPC), enviando peticiones al servidor mediante transferencia de mensajes. Para el caso de llamadas al sistema relacionadas con el sistema de archivos, el servidor MoL-FAT resuelve dichas peticiones y devuelve la respuesta a los clientes también en un mensaje.

La conversión de una llamada al sistema a un par de transferencias de mensajes (*request-reply*) es realizada por la biblioteca MoL que empaqueta los argumentos de la llamada al sistema en el contenido de diferentes campos del mensaje de petición (*parameter marshalling*). De forma similar, los resultados de la llamada al sistema se retornan en diferentes campos del mensaje de respuesta y se devuelven a la aplicación como valor de retorno de la función invocada, en la variable ***errno*** y/o en variables pasadas como argumentos por referencia. Los mensajes del protocolo MoL, tienen una longitud de 36 bytes y están disponibles en varios formatos que combinan números enteros, punteros y caracteres.

En la Fig. 3, se presenta un pseudo-código de la llamada al sistema *mol_read()* y *molsyscall()* para ilustrar cómo funciona el protocolo.

En este ejemplo, la función *molsyscall()* es la encargada de enviar el mensaje con la operación requerida y sus parámetros, como así también de retornar los resultados. Por su parte, *mol_read()* empaqueta los argumentos de la función dentro del mensaje.

La aplicación de usuario (cliente) utiliza *mol_read()* con los parámetros especificados: un descriptor de archivo (***fd***), un puntero a un buffer donde se almacenarán los datos leídos (*buffer*), y la cantidad de bytes a leer (*bytes*). Esta función invoca a *molsyscall()*, que además de empaquetar los campos ya nombrados, completa con el tipo de mensaje de la operación requerida en el campo *m_type* , que en este caso corresponde al código MOLREAD. Luego, se realiza la transferencia de mensajes *mnx_sendrec()* de M3-IPC con el **endpoint** (FS_PROC_NR) del destinatario MoL-FAT, para luego quedar bloqueado el proceso a la espera del mensaje de respuesta.

MoL-FAT, que se encontraba bloqueado a la espera de peticiones en la función *mnx_receive()* de M3-IPC, ahora es desbloqueado y puede comenzar con el tratamiento del requerimiento.

El pseudo-código de la Fig. 4 muestra que para realizar la copia de datos desde MoL-FAT (origen) hacia el espacio de direcciones del cliente (destino) se utiliza *mnx_vcopy()* de M3-IPC.

```
ssize_t mol_read(int fd, void *buffer, mnx_size_t nbytes)
{
   int rcode;
   message m __attribute__((aligned(0x1000)));

   m.m1_i1 = fd;
   m.m1_i2 = nbytes;
   m.m1_p1 = (char *) buffer;
```

```
    rcode= molsyscall(FS_PROC_NR, MOLREAD, &m);
    return(rcode);
}

int molsyscall(int who, int syscallnr, message *msgptr)
{
    int status;
    /* send the request to the server and wait for the reply*/
    msgptr->m_type = syscallnr;
    status = mnx_sendrec(who, msgptr);
    if (status != 0) {
        msgptr->m_type = status;
    }
    if (msgptr->m_type < 0) {
        /* request has failed fill errno */
        errno = -msgptr->m_type;
        return (-1);
    }
    return (msgptr->m_type);
}
```

Figura 3. pseudo-código de mol_read() y molsyscall()

En este ejemplo se utiliza archivos de imagen de disco como dispositivo de bloque y la función utilizada para leer los datos es *f_read()* de la biblioteca FatFs para leer datos desde un dispositivo con formato FAT.

MoL-FAT no solo puede utilizar archivos de imágenes de disco como dispositivos de bloques, sino también MoL-VDD que es una tarea que gestiona dispositivos de bloques y que se comunica con M3-IPC. MoL-VDD puede configurarse con soporte de replicación para tolerar fallos.

La principal ventajas de utilizar archivos de imágenes de disco es que los mismos pueden estar localizados en un disco rígido, en discos RAM, en un dispositivo USB, un archivo remoto accedido por NFS, etc. La desventaja obvia está relacionada con el rendimiento.

FUSE para MoL-FAT

Para acceder a los servicios de MoL-FAT se requiere que la aplicación cliente se registre ante el M3-IPC para que se le otorgue un endpoint para identificarla. Esto limita la posibilidad de su utilización a aplicaciones y desarrollos específicos que utilizan M3-IPC.

```
//*************** CLIENT ******************
client{
  char clt_buff[BUFFSIZE];
  int bytes;
   ...
  bytes = mol_read(fd, clt_buff,  BUFFSIZE) ;
  if( rcode < 0) exit(EXIT_FAILURE);
   // Here, the buffer has data copied by the server
}

//*************** SERVER******************
int clt_ep; /*client endpoint */
int fatFs_fd;
int rcode, len, bytes;
char svr_buff[SSIZE_MAX];
message msg_in, msg_out;
```

```
while(true) {
  // blocks until receive a client request
 rcode = mnx_receive(ANY, &msg_in);
 if( rcode < 0) exit(EXIT_FAILURE);
 client_ep = msg_in.source;
 switch(msg_in.oper) {
   case MOL_READ:
     len = (msg_in.len<SSIZE_MAX)?
        msg_in.len : SSIZE_MAX;
     /* FatFS Read a chunk of source file */
     fr = f_read(fatFs_fd, svr_buff, len, &bytes);
     if(bytes < 0) error_reply(bytes);
     // copies Server's buffer to Client's buffer
     mnx_vcopy(SELF,  /* MoL-FAT endpoint */
                svr_buff /* local buffer addresss */
                clt_ep; /* client endpoint */
                msg_in.addr,/* client address */
                bytes); /* bytes to copy*/
     msg_out.rcode = OK;
     msg_out.bytes = bytes;
     //Send reply to the client
     mnx_send(clt_ep, &msg_out);
     break;
   case MOL_WRITE:
   ...
   break;
   default:
         msg_out.rcode = ERROR_CODE;
         mnx_send(clt_ep, &msg_out);
   }
}
```

Figura 4. Cliente y Servidor para mol_read().

Para que las aplicaciones Linux genéricas puedan acceder a MoL-FAT, se desarrolló un controlador FUSE específico que permite la utilización de una enorme gama de herramientas para realizar las pruebas y benchmarks. En este caso es el controlador FUSE es el que utiliza M3-IPC y funciona como un proxy hacia MoL-FAT.

FUSE utiliza varios hilos de ejecución para realizar sus tareas concurrentemente. Aunque M3-IPC está preparado para trabajar con hilos de ejecución, la biblioteca de FUSE crea los hilos en forma dinámica y los mismos no pueden enlazarse individualmente al kernel de M3-IPC (sin alterar la biblioteca estándar. Para evitar condiciones de competencia se utilizan *mutexes pthreads* de Linux como mecanismo de exclusión mutua de tal forma de realizar un acceso no concurrente al kernel de M3-IPC. Esto afecta de manera adversa el rendimiento puesto que las operaciones sobre *mutexes* implican cambios de contexto y de modo adicionales.

El controlador FUSE convierte en forma transparente cualquier llamada al sistema de tipo POSIX (por ejemplo *read()*) en una llamada al sistema MoL-FAT (por ejemplo *mol_read()*). De esta forma, es posible acceder al sistema de archivos MoL-FAT desde cualquier aplicación típica de Linux sin realizar modificación alguna.

EVALUACION

Para realizar los benchmarks y micro-benchmarks sobre el prototipo las configuraciones se agruparon de la siguiente manera:

- *Con FUSE*: El acceso al sistema de archivos para realizar pruebas utilizando el comando ***cp*** de Linux se hace a través del controlador FUSE. (Fig. 5)
- *Con M3Copy*: Un programa cliente que utiliza M3-IPC especialmente desarrollado para realizar copias entre archivos en MoL-FAT y archivos regulares en Linux (Fig. 6).

Los escenarios con FUSE quedan organizados con las siguientes combinaciones:

- A1: El cliente, el controlador FUSE y MoL-FAT se ejecutan en el mismo nodo (Node 0). MoL-FAT usa directamente un archivo regular Linux como imagen de disco.
- A2: El cliente, el controlador FUSE, MoL-FAT y MoL-VDD se ejecutan en el mismo nodo (Node 0). MoL-VDD usa un archivo regular Linux como imagen de disco.
- B: El cliente, el controlador FUSE y MoL-FAT se ejecutan en el mismo nodo (Node 0). MoL-VDD se ejecuta en otro nodo (Node 1) y usa un archivo regular Linux como imagen de disco.

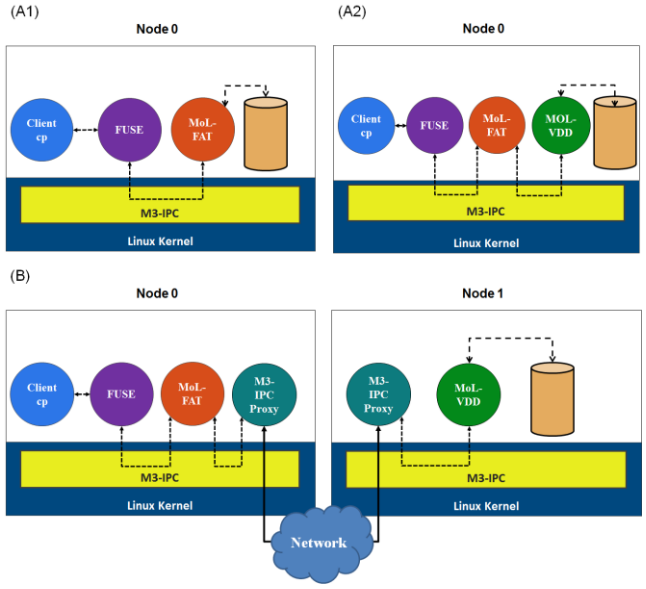


Figura 5. Escenarios de pruebas con FUSE.

Los escenarios con M3copy quedan organizados con las siguientes combinaciones (Fig. 6):

- C1: El cliente M3Copy y MoL-FAT se ejecutan en el mismo nodo. MoL-FAT usa directamente un archivo regular Linux como imagen de disco.
- C2: El cliente M3Copy, MoL-FAT y MoL-VDD se ejecutan en el mismo nodo (Node 0). MoL-VDD usa un archivo regular Linux como imagen de disco.
- D: El cliente M3Copy y MoL-FAT se ejecutan en el mismo nodo (Node 0). MoL-VDD se ejecuta en otro nodo (Node 1) y usa un archivo regular Linux como imagen de disco.

Con estos escenarios se pueden verificar los objetivos de diseño de MoL-FAT:

- Debe soportar archivos de imágenes de disco, tanto locales como remotas.
- La comunicación entre los clientes y MoL-FAT debe ser a través de mensajes M3-IPC.
- Los procesos sin soporte M3-IPC pueden usar el servidor MoL-FAT a través de FUSE, como el comando copy de Linux.

Para la ejecución de las pruebas de utilizó un cluster con Debian Linux (kernel 2.6.32) con PCs con CPU Intel(R) Core(TM) i7-4790 CPU @ 3.60GHz, de 8 núcleos y 8GB de RAM unidos por un switch de 1Gbps por puerto.

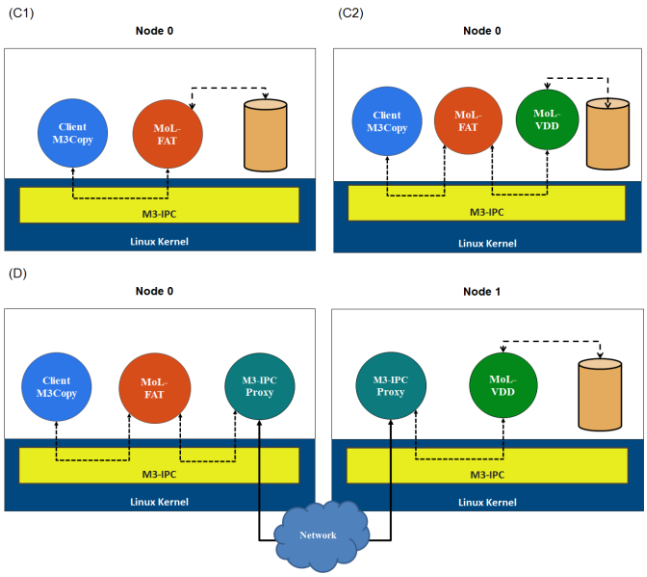


Figura 6. Escenarios de pruebas con M3Copy.

En la Fig. 7 se presentan los resultados de las pruebas de rendimiento de transferencias de archivos con cliente y servidor compartiendo el mismo Nodo (escenarios A1 y A2).

Los mejores resultados, como era de esperarse los tienen las copias realizadas utilizando M3Copy. Se aprecia también el impacto negativo que tiene en el rendimiento la utilización del controlador FUSE. Es de destacar el pobre rendimiento (2 Mbytes/s) que tienen HSFS (cliente NFS) y UNFS (servidor NFS que también utiliza FUSE) se debe a que ejecutan en modo usuario.

En la Fig. 8 se presentan los resultados para los mismos escenarios, pero ahora para escrituras. HSFS y UNFS no soportan escritura.

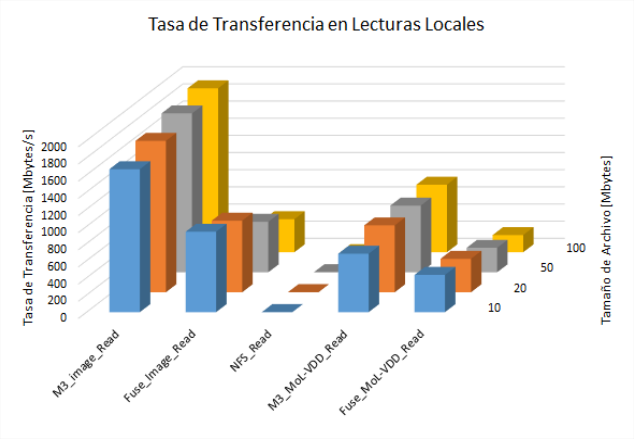


Figura 7. Tasa de Transferencias en Lecturas Locales.

En las escrituras, el impacto que le impone el controlador FUSE es mucho mayor, tal como se anticipa en [18].

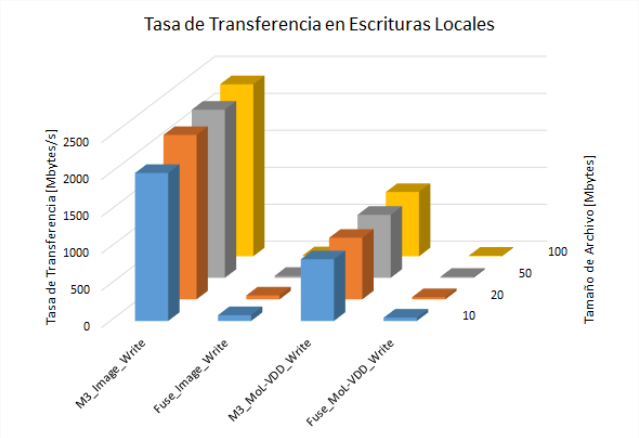


Figura 8. Tasa de Transferencia en Escrituras Locales.

Además, se desarrollaron micro-benchmarks para evaluar el desempeño de MoL-FAT cuando se utiliza un servidor de disco (MoL-VDD) en otro nodo (Fig. 9).

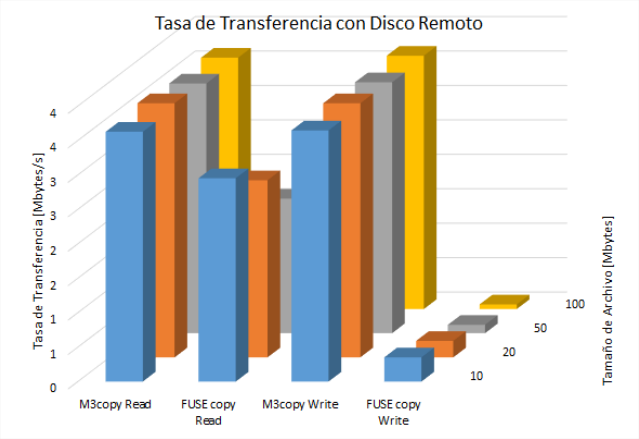


Figura 9. Tasa de Transferencia con Disco Remoto.

Las pruebas realizadas utilizando el programa cliente m3copy, muestran un desempeño regular independiente del tamaño de archivo y de si la operación es lectura o escritura. En tanto que cuando se utiliza el controlador FUSE, el rendimiento decrece a medida que aumenta el tamaño de archivo. No se ha realizado un análisis más profundo acerca de este comportamiento, pero las sospechas recaen en los buffers cache de FUSE (librerías) y no del controlador propiamente dicho que es muy sencillo. También existen notables diferencias de rendimiento entre las lecturas y las escrituras, lo que es consistente con las pruebas de transferencias locales.

Finalmente, se realizaron pruebas con el cliente ejecutando en un nodo y el MoL-FAT en otro nodo, es decir equivalente a un sistema de NFS Cliente y NFS Servidor en nodos diferentes (Fig. 10).

Se puede apreciar que el rendimiento del cliente m3copy es prácticamente idéntico al rendimiento del NFS en modo usuario. No existen diferencias apreciables en la tasa de transferencia entre las lecturas y las escrituras de datos. El rendimiento decrece apreciablemente si se utiliza FUSE en el cliente, y lo hace en forma realmente significativa si se trata de escrituras (apenas 563 KBytes/s en promedio), comportamiento consistente con otras pruebas realizadas.

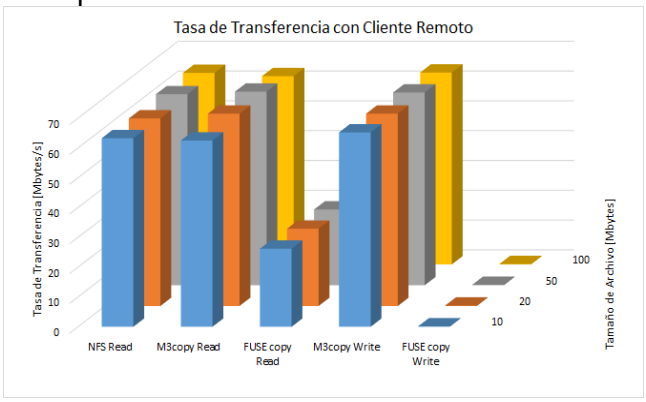


Figura 10. Tasa de Transferencia con Cliente Remoto.

CONCLUSIONES Y TRABAJOS FUTUROS

MoL-FAT es un sistema de archivos de espacio de usuario diseñado como un componente de un VOS denominado MoL. MoL-FAT puede utilizar archivos proporcionados por el OS subyacente como archivo de imagen de disco tal como si éstos fuesen dispositivos de bloques. También se puede utilizar un dispositivo de disco remoto virtual a través de la tarea de disco MoL-VDD.

Si bien el MoL-FAT se desarrolló en el marco del proyecto de DVS, puede ser utilizado por cualquier proceso ordinario de Linux mediante la instalación de un controlador FUSE.

Entre los trabajos futuros planeados como continuidad de este proyecto se puede mencionar la incorporación del soporte de otros tipos de sistema de archivos tales como Ext2/3/4

y mejorar la disponibilidad utilizando de replicación.

Como MoL-FAT utiliza M3-IPC, el acceso a sus servicios puede realizarse en forma local o remota respecto a la ubicación de la aplicación cliente que la utiliza. Como se ejecuta en espacio de usuario, es más adecuado para la migración y la replicación. Estas son características básicas requeridas para brindar servicios en la nube escalables y de alta disponibilidad.

REFERENCIAS

Artículos en publicaciones periódicas:

[1] D. Hall, D. Scherrer, J. Sventek, "*A Virtual Operating System*", Journal Communication of the ACM, 1980.

[2] J. Dike, "*A user-mode port of the Linux kernel*", USENIX Association. Proceedings of the 4th Annual Linux Showcase & Conference, Atlanta Oct 10 -14, 2000.

[3] P. Pessolani, O. Jara, "*Minix over Linux: A User-Space Multiserver Operating System*", in Proc. Brazilian Symposium on Computing System Engineering, Florianopolis, 2011.

[4] Soltesz, S., P ¨Otzl, H., Fiuczynski, M. E., Bavier, A. C.,And Peterson, L. L. "*Container-based operating system virtualization: a scalable, high-performance alternative to hypervisors*".In EuroSys (2007).

[5] OpenVZ. http://en.wikipedia.org/wiki/OpenVZ.

[6] Osman, S., Subhraveti, D., Su, G., And Nieh, J. "*The design and implementation of Zap: A system for migrating computing environments*". In OSDI (2002).

[7] P. Pessolani, P., T. Cortes, S. Go*nnet, F. Tinetti, "Un sistema de virtualización distribuida*", Workshop de Investigadores en Ciencias de la Computación (WICC 2017), La Plata, 2017.

[8] FatFS.http://elm-chan.org/fsw/ff/00index_e.html

[9] NFS. https://tools.ietf.org/html/rfc1813

[10] User-space NFSv3 Server, http://unfs3.sourceforge.net/

[11] HSFS. https://github.com/openunix/hsfs.

[12] FUSE. https://github.com/libfuse/libfuse.

[13] Anoop Karollil,"*RadFS- Virtualizing Filesystems*", Master thesis The University Of British Columbia, 2008.

[14] V. Jujjuri, E. Van Hensbergen, A. Liguori, and B. Pulavarty, "*VirtFS - A Virtualization Aware File System Pass-through*",In Proceedings of the 2010 Linux Symposium, 109-120, 2010.

[15] Pablo Pessolani, Tony Cortes, Fernando G. Tinetti, Silvio Gonnet; "*An IPC Software Layer for Building a Distributed Virtualization System*"; CACIC 2017 - Congreso Argentino de Ciencias de la Computación, La Plata, Argentina, 2017.

[16] Tanenbaum A., Woodhull A., "Operating Systems Design and Implementation, Third Edition", Prentice-Hall, 2006.

[17] A. Madhavapeddy, D. Scott, "*Unikernels: The rise of the virtual library operating systems*", Communications of the ACM (CACM) 57, 1(Jan. 2014), 61-69.

[18] https://engineering.facile.it/blog/eng/write-filesystem-fuse/