UNIVERSIDAD COMPLUTENSE
MADRID

Mejoras para la Gestión Interna de Clouds Privados

FACULTAD DE INFORMÁTICA

SISTEMAS INFORMÁTICOS

2011-2012

Componentes

Ángel Carrasco Villaverde.

José Carlos Esteban Gómez

Rubén Sánchez Malagón.

Dirigido por:

Eduardo Huedo

Rubén Santiago

Declaración de conformidad

Los alumnos Ángel Carrasco Villaverde, José Carlos Esteban Gómez y Rubén Sánchez Malagón, aquí firmantes, autorizan a la Universidad Complutense de Madrid a la difusión de la memoria, implementación, imágenes y código del proyecto realizado con fines exclusivamente académicos y mencionando expresamente a los autores del mismo.

ÁNGEL CARRASCO VILLAVERDE

JOSÉ CARLOS ESTEBAN GÓMEZ

RUBÉN SÁNCHEZ MALAGÓN

Agradecimientos

ÁNGEL CARRASCO VILLAVERDE

Quiero agradecer en primer lugar a todos los compañeros con los que he compartido clase, que han hecho que esta importante etapa de mi vida sea más sencilla y de los que me llevo un gran recuerdo. Por supuesto a mis dos compañeros en este proyecto, Rubén y José Carlos, que han realizado un magnífico trabajo.

Por supuesto, quiero dar las gracias a todos los profesores. Ellos ha hecho posible que adquiera los conocimientos necesarios y algunos de ellos han contribuido a mi formación como persona, que posiblemente es más importante.

Por último, y quizá el punto más importante. Quiero agradecer a mi familia, especialmente a mi madre, por todo el apoyo que me ha proporcionado. Ella es la artífice de todo lo que he conseguido hasta ahora, gracias a su esfuerzo y dedicación he llegado hasta aquí. No quiero olvidar a mi novia, que además de brindarme su apoyo, me ha aguantado en esos duros momentos. Mil gracias

JOSÉ CARLOS ESTEBAN GÓMEZ

Este agradecimiento no puede empezar sin acordarme de mis padres y familia, los cuales han estado siempre apoyándome en cualquier circunstancia. Por darme la oportunidad de estudiar una carrera que no solo me ha formado como informático sino que me ha inculcado unos valores y conocimientos que podré utilizar en cualquier ámbito de la vida.

No me puedo olvidar de mis compañeros de facultad tanto en la técnica como en la superior, con ellos las penas académicas siempre han sido menos dolorosas y las alegrías más eufóricas.
Mención especial a mis dos compañeros de proyecto: Rubén y Ángel. De los cuales siempre he recibido toda la ayuda que he podido necesitar y han hecho más llevadero las largas tardes de estudio en la biblioteca.

A todos ellos, GRACIAS

RUBÉN SÁNCHEZ MALAGÓN

A todas esas personas que me han dado la fuerza, la energía, el apoyo y la confianza que en ocasiones hacen falta para seguir adelante en todo momento, a todas las han hecho posible que haya podido llegar hasta aquí. GRACIAS

Agradecer en especial a mi familia y mis amigos que además de todo lo anterior tengo que agradecerles su paciencia en los momentos de mayor trabajo, sin todos ellos mi camino hubiera sido mucho más difícil.

Para terminar dar las gracias a mis compañeros de proyecto Ángel y José Carlos además de todos los compañeros de la universidad con los que he tenido la oportunidad de compartir todo o parte de este tiempo, gracias por hacerme disfrutar de unos grandes años. No quisiera olvidarme de todos los que intentan crear un mundo mejor haciendo lo que mejor se saben hacer en la vida, enseñar a aprender.

A todos MUCHAS GRACIAS.

Resumen y palabras clave

Resumen

El objetivo de este proyecto es proporcionar al ecosistema de OpenNebula un nuevo sistema de gestión de imágenes de máquinas virtuales mediante el módulo de “Transfer Manager”.

Vimos que los diferentes gestores de transferencias de imágenes virtuales de OpenNebula se comportaban de la misma forma con aquellas imágenes que habían sido desplegadas por un host con anterioridad que con aquellas que nunca hubieran desplegado. Es decir, un host que siempre utilizaba una imagen tardaba lo mismo que un host que utiliza diferentes imágenes.

Nuestro proyecto resuelve dichos tiempos incluyendo un completo sistema de gestión de memoria caché, centralizado en servidor, para el almacenamiento de imágenes en nodos y gestión de aquellas imágenes que se vayan a desplegar con más frecuencia en diferentes hosts, para ello se utilizan un conjunto configurable de políticas de reemplazo que indicarán qué imágenes son susceptibles de abandonar la caché debido a la falta de espacio.

Con este sistema se pretende minimizar el tiempo de despliegue de las máquinas, de forma que el impacto en el usuario final a la hora de solicitar una imagen sea menor. Por otro lado, se elimina la congestión que pueda tener la red tanto a nivel general como en picos de solicitudes de máquinas virtuales. Por todo lo anterior, nuestra intención es ayudar a la comunidad de OpenNebula aportado una nueva perspectiva en el manejo de imágenes.

El diseño del “plugin” se ha realizado de forma que sea ampliamente configurable y extensible, proporcionando facilidad hacia futuras implementaciones tanto para la realización de extensiones para nueva políticas de reemplazo y como para mejoras en su diseño.

Palabras clave

OpenNebula, políticas de reemplazo, memoria caché, Cloud Computing, IaaS, Transfer Manager

Summary and keywords

Summary

The objective of this project is to provide the ecosystem of OpenNebula a new system image management of virtual machines using the module "Transfer Manager".

We saw that the different managers of virtual image transfers of OpenNeubla behave the same way with images that had deployed before by a host that with those who had never deployed. That is, a host who always use the same image spend the same time that a host that uses different images.

Our project resolves these times including a complete management system centralized server cache for storing images on management node images that are to be deployed more often in different hosts, this will use a configurable set of replacement policies that indicate which images are likely to leave the cache due to lack of space.

This system aims to minimize deployment time machine, so that the impact on the end user when requesting an image is less. On the other hand, removes congestion network may have both general and peaks of applications of virtual machines. For all these reasons, we intend to help the community of OpenNebula provided a new perspective in the management of images.

The design of the "plugin" has been made so that it is highly configurable and extensible, providing ease to future implementations for both completion of extensions for new and replacement policies to design improvements

Keywords

OpenNebula, replacement policies, memory cache, Cloud Computing, IaaS, Transfer Manager.

CAPÍTULO 1: Introducción

OpenNebula es un conjunto de herramientas de código abierto destinadas a la gestión de un data center virtualizado con una infraestructura distribuida heterogénea. Por ello, enmarcamos este proyecto en el ámbito de la “Virtualización y el Cloud Computing”, concepto que se está popularizando en estos últimos años y que se posiciona como concepto clave en la computación de nuestros días y en los del futuro.

Dicha tecnología se basa en la abstracción de los recursos de una computadora denominada “Hypervisor” o VMM (“Virtual Machine Monitor”) que crea una capa de abstracción entre el hardware de la máquina física y el sistema operativo de la máquina virtual, dividiéndose el recurso entre uno o más entornos de ejecución.

Como ejemplo representativo, gracias a la virtualización, podemos tener varios sistemas operativos funcionando simultáneamente en una misma máquina física. De esta manera, podemos aprovechar al máximo las capacidades de una máquina y a la vez, ser completamente independiente de su arquitectura hardware.

Además, la virtualización nos proporciona portabilidad, hecho que aumenta enormemente su facilidad de gestión y representa una gran ventaja para los clientes. Pueden moverse en segundos sistemas completos, aplicaciones, sistemas operativos, BIOS y hardware virtual totalmente configurados, desde un servidor físico a otro, sin paradas por mantenimiento y con una consolidación continua de la carga de trabajo.

La virtualización también proporciona seguridad debido a que cada máquina virtual es totalmente independiente del resto y un ataque sobre una sólo afectaría esta. Nos proporciona agilidad, ya que la creación de una máquina virtual es un proceso muy rápido, realizándose simplemente mediante la ejecución de un comando.

Otra gran ventaja es la recuperación rápida en caso de fallo. Si se dispone de una copia de los ficheros de configuración de la máquina virtual, en caso de desastre la recuperación será muy rápida, simplemente arrancar la máquina virtual con los ficheros de configuración guardados.

Como con cualquier otra tecnología, la virtualización también tiene algunas desventajas, tales como el desaprovechamiento de recursos o reducciones de rendimiento de los sistemas virtualizados. Estas desventajas pueden paliarse con un hardware potente y una buena gestión de las máquinas virtuales.

El concepto de “La Nube“ hace referencia a algunas de las características y ventajas de la virtualización, tales como la portabilidad o la independencia de la plataforma. Atrás queda el tiempo en el que todo el software que utilizamos se encontraba instalado en nuestros equipos. Hoy en día, somos capaces de ejecutar aplicaciones remotas de gran complejidad y valernos de programas externos capaces de generar contenidos, alojar nuestra información y satisfacer nuestras necesidades en los ámbitos más diversos y sin que sea precisa ningún tipo de instalación previa.

Lo importante para el usuario es que la información está en “La Nube” y esta puede ser accedida en cualquier momento y lugar, por otro lado para el administrador lo que le interesa es el aprovechamiento de los recursos y la adaptación de las demandas de los usuarios.

Es evidente el éxito de la virtualización en nuestros días, el “Cloud Computing” se encuentra extendido en pequeñas y grandes empresas de todo tipo y necesidades.

El gestor OpenNebula, desarrollado por miembros de la Universidad Complutense de Madrid, permite gestionar centros de datos virtualizados e infraestructuras en la nube, es decir se encarga del tipo de solución Cloud para IaaS. La infraestructura como servicio se encarga de proveer de servicios hardware virtualizados bajo demanda.

Destaca sobre todo por los siguientes puntos:

- Adaptable, extensible e integrable: Gracias a que es un proyecto de código abierto, podemos tener a nuestra disposición la arquitectura. Esta es adaptable y extensible facilitando la adecuación a nuestras necesidades.

- Facilidad de configuración e instalación.

- Potente: Permite administrar máquinas físicas, redes e imágenes (almacenamiento) permitiendo la monitorización de los recursos.

- Compatible con varios sistemas operativos: Hoy en día esta herramienta es completamente funcional en las siguientes distribuciones Linux: Ubuntu, Suse, Debian.

- Flexible: Permite la utilización de varios sistemas de virtualización o hipervisores como KVM, XEN o VMWare.

- Comunidad de usuarios muy activa.

- Amplia documentación: Si es cierto que como toda nueva herramienta, los inicios para su comprensión fueron complicados, pero gracias a la extensa documentación y a la comunidad, la curva de aprendizaje se suavizó bastante.

- En continuo desarrollo.

A todas estas buenas características de OpenNebula, esperemos añadir con nuestro desarrollo la semilla para que los intercambios de imágenes entre máquinas sean más livianos y óptimos ofreciendo una solución adaptativa al usuario con diferentes políticas de reemplazamiento.

1.1 Objetivo del Proyecto

Este proyecto tiene por objetivo el desarrollo de un driver que permita la utilización de una memoria caché en el módulo de Transfer Manager del gestor OpenNebula.

Dicho módulo se encarga de gestionar las transferencias de imágenes de máquinas virtuales en OpenNebula.

El driver estará a cargo de todas las transferencias de ficheros involucrados en el despliegue de las máquinas virtuales. Esto incluye entre otras operaciones, la transferencia desde el servidor o repositorio de imágenes al nodo, la comprobación de integridad de datos en la caché, borrado de la imagen en caso de reemplazamiento o la orden de uso de una imagen desde la caché cuando el nodo ya ha desplegado previamente la imagen de la máquina virtual.

Además el gestor de políticas llevará un mantenimiento de estadísticas para cada política de reemplazamiento habilitada, de esta manera si el administrador observa que el coste de rendimiento obtenido no es el esperado, puede cambiar los valores de la configuración de pesos asignados a cada política y obtener nuevos resultados que se ajusten más a sus necesidades y usos.

Todo ello implementado teniendo en cuenta posibles problemas de integridad de datos y robustez que puedan darse debido a problemas internos o externos como puede ser un fallo en la conexión de red y su posible daño en la transferencia de datos o la eliminación de datos de manera accidental tanto en las estadísticas del servidor como en las imágenes cacheadas en los nodos.

Con este driver intentamos ofrecer una nueva alternativa al manejo de imágenes con la utilización y configuración de una memoria caché así como un ahorro en su manipulación.

La memoria caché dispondrá de un gestor, que será el encargado de decidir qué imagen o imágenes deben abandonar el directorio de caché en caso de fallo y falta de espacio en el mismo. Para tomar esta decisión el gestor usa una o varias políticas de reemplazamiento típicas (LRU, LFU, FIFO, etc.).

Debe ser posible configurar de forma sencilla qué política de reemplazamiento se quiere usar, en función de las necesidades que tenga el administrador. También, para enriquecer el gestor y que disminuya la tasa de fallos, debe ser posible combinar varias políticas de reemplazamiento, otorgando pesos a cada una de ellas.

Además, el desarrollo del gestor de reemplazamientos debe permitir de forma sencilla y rápida su extensión, pudiendo desarrollar una nueva política o estrategia ad-hoc en caso de que fuera necesario y realizar su integración de la manera menos intrusiva posible en el sistema previo.

Como objetivo de la implementación se ha desarrollado el driver de forma independiente a la distribución original de OpenNebula para que posteriormente pueda ser integrado a modo de plugin a través de un instalador independiente.

CAPÍTULO 2: Virtualización

La Virtualización es una técnica empleada sobre las características físicas de los computadores, para ocultarlas de otros sistemas, aplicaciones o usuarios que interactúen con ellos. Esto hace que un recurso físico, como un servidor, un dispositivo de almacenamiento o un sistema operativo, sea visto como si fuera varios recursos lógicos a la vez, o que varios recursos físicos, como servidores o dispositivos de almacenamiento, parezcan un único recurso lógico.

Esto nos permite crear una separación entre hardware y software, haciendo que se puedan ejecutar simultáneamente en un mismo servidor o computador múltiples sistemas operativos, aplicaciones o plataformas lógicas.

Existen diferentes formas de virtualización: podemos virtualizar el hardware o el software de un servidor, virtualizar sesiones de usuario, aplicaciones y también se pueden crear máquinas virtuales en un ordenador personal.

La virtualización no es un invento reciente, pero el gran auge que está teniendo la computación en la nube ha hecho que la virtualización haya pasado a ser uno de los componentes fundamentales de las llamadas infraestructuras de nube privada.

En este capítulo expondremos los diferentes tipos de virtualización existentes hoy día, así como las ventajas e inconvenientes de realizar un uso de ellas.

2.1 Virtualización de plataforma

La virtualización de plataforma consiste en simular una máquina real con todos sus componentes (aunque estos no tienen por qué ser proporcionados completamente por la máquina física), y prestarle todos los recursos necesarios para su funcionamiento. En general, hay un software anfitrión que es el encargado de controlar que las diferentes máquinas virtuales sean atendidas correctamente, situado habitualmente entre el hardware y las máquinas virtuales. Dentro de este esquema se encuentran la mayoría de las formas de virtualización más usadas y conocidas, incluidas la virtualización de sistemas operativos, la virtualización de aplicaciones y la emulación de sistemas operativos.

2.1.1 Virtualización completa

La máquina virtual simula el hardware necesario para soportar un sistema operativo huésped que se ejecuta de forma aislada. La virtualización es completa debido a que el sistema operativo huésped no es modificado para ser ejecutado.

Este método tiene todas las ventajas de la paravirtualización, con el añadido de que no es necesaria ninguna modificación a los guests. La única restricción es que estos últimos deben soportar la arquitectura de hardware utilizada.

La siguiente figura muestra la estructura de este tipo de virtualización:

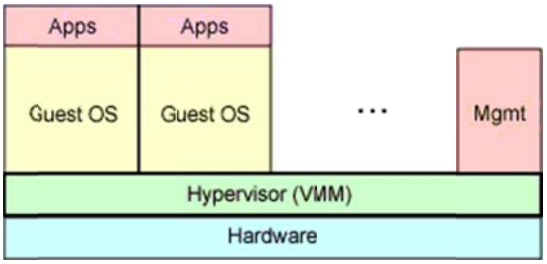


Fig 1. Virtualización completa.

2.1.2 Paravirtualización

Se trata de un tipo de virtualización que necesita que los sistemas operativos “huésped” sean modificados para adaptarse a la API del Hypervisor. Por tanto, no se considera una virtualización pura, aunque proporciona algunas de las ventajas de la virtualización total.

Las ventajas de este enfoque son un muy buen rendimiento y la posibilidad de ejecutar distintos sistemas operativos como guests. Se obtienen, además, todas las ventajas de la virtualización enunciadas anteriormente. Su desventaja es que los sistemas operativos guests deben ser modificados para funcionar en este esquema.

En la siguiente figura está representado el esquema que sigue:

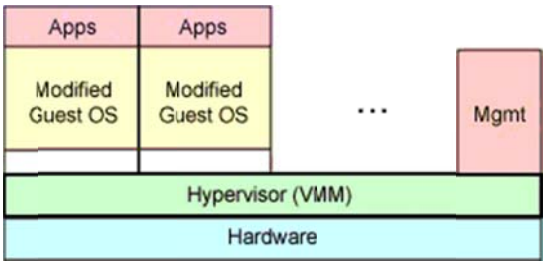


Fig 2. Paravirtualización.

2.1.3 Virtualización por S.O.

En este tipo de virtualización sólo hay un sistema operativo en el que se virtualizan aplicaciones (generalmente servidores) de manera que las aplicaciones que corren en el entorno *"huésped"* lo ven como un sistema autónomo.

Aquí podemos ver un esquema:

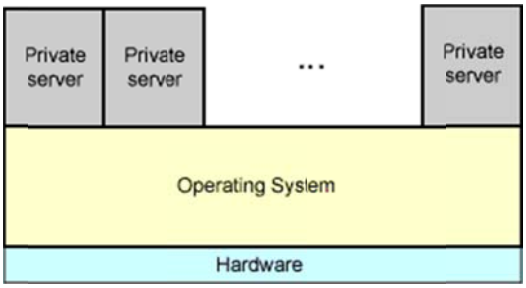


Fig 3. Virtualización por S.O.

2.1.4 Virtualización de aplicaciones

Aunque no es una virtualización de plataformas también existe la posibilidad de virtualizar aplicaciones.

Para ello la máquina virtual crea un entorno virtual que hace que las aplicaciones no dependan del hardware, del sistema operativo o de otras aplicaciones.

Un caso de esta virtualización lo vemos en la máquina virtual usada por Java, que hoy en día se encuentra presente en todo tipo de dispositivos. Otras máquinas virtuales de este tipo son: Ruby, Parrot (usada por Perl), Smalltalk, etc.

El siguiente esquema ilustra lo comentado anteriormente:

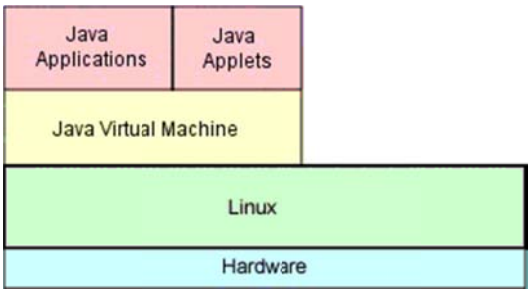


Fig 4. Virtualización de aplicaciones.

2.2 Ventajas e inconvenientes de la virtualización

La virtualización presenta las siguientes ventajas:

- Aislamiento**:** las máquinas virtuales son totalmente independientes entre sí, e independientes del hypervisor. En caso de producirse algún fallo en una máquina virtual afectará únicamente a esa máquina. El resto de máquinas virtuales y el hypervisor no se verán afectadas en absoluto.

- Flexibilidad**:** es posible crear las máquinas virtuales configurando las características de CPU, memoria, disco y red que necesitemos, en lugar de tener que invertir en una máquina física con unas características concretas. Además, es posible crear varias máquinas virtuales con características muy diferentes dentro de un mismo sistema operativo.

- Seguridad**:** cada máquina alojada en el equipo o servidor tienen la posibilidad de tener diferentes login de acceso. Por tanto, un ataque que afecte a una de las máquinas, por lo general no afectará al resto.

- Agilidad**:** es posible crear máquinas virtuales de forma rápida y sencilla, haciendo que obtener una máquina de unas determinadas características sea tan sencillo como ejecutar un comando.

- Portabilidad**:** es sencillo clonar y transportar una máquina virtual de un servidor a otro, ya que la configuración de la máquina reside en uno o varios ficheros. Además permite tener copias del sistema completo de forma sencilla.

- Recuperación rápida en caso de fallo: como realizar una copia es tan sencillo como copiar unos ficheros, en caso de desastre la recuperación será muy rápida, simplemente arrancar la máquina virtual con los ficheros de configuración guardados. No es necesario reinstalar, recuperar backups o cualquier otro proceso necesario para otros sistemas.

- Aprovechamiento del hardware: no es necesario destinar un servidor o computador a cada sistema operativo, haciendo que con una misma máquina física pueda tener varios SO.

- Balanceo de recursos: es posible asignar un grupo de servidores físicos para que proporcionen recursos a las máquinas virtuales y asignar una aplicación que haga un balanceo de los mismos, otorgando más memoria, recursos de la CPU, almacenamiento o ancho de banda de la red a la máquina virtual que lo necesite.

Sin embargo, la virtualización también presenta los siguientes inconvenientes:

- Emulación de controladores: es posible encontrar problemas con ciertos componentes de hardware o controladores, que impiden operar y funcionar de la misma forma que trabajando sobre el sistema operativo anfitrión.

- Rendimiento: las máquinas virtuales imponen unos límites de adjudicación de recursos, puesto que el sistema anfitrión, debe de seguir manteniendo unos mínimos de recursos para poder virtualizar al sistema invitado.

- Dependencia del sistema anfitrión: un fallo de un servidor anfitrión de virtualización afecta a todos los servidores virtuales que aloja, por lo que es importante no solo tener copias de seguridad de las máquinas, sino incluso puede hacer falta un clusters de servidores anfitriones para evitar tener posibles fallos.

- Falta de uniformidad: cada hypervisor soporta máquinas virtuales con unas características distintas, no estandarizadas, haciendo que una máquina virtual, normalmente sólo pueda ser ejecutada por un gestor concreto.

CAPÍTULO 3: Memoria caché

La memoria caché es un sistema de almacenamiento de alta velocidad, empleado para mejorar la eficiencia de algunos procesos de obtención de información, cuando ésta ya ha sido usada con antelación. Reutilizando los datos o ficheros almacenados en el sistema de memoria caché se consigue ahorrar el tiempo empleado en obtener dicha información de la fuente original.

Este sistema de memoria suele tener un tamaño mucho menor al tamaño de la fuente de datos original, por tanto necesita ser gestionado usando políticas de reemplazamiento y/o estrategias de ubicación.

En este capítulo introducimos al lector en las bases de la memoria caché, los factores a tener en cuenta para su correcto diseño y rendimiento y los diferentes tipos de memorias caches que coexisten en el día a día del mundo web.

3.1 Caché de CPU

La memoria caché de una CPU es una memoria pequeña y rápida que se interpone entre la propia CPU y la memoria principal para que el conjunto opere a mayor velocidad. Para ello es necesario mantener en la memoria caché los bloques o zonas de la memoria principal con mayor probabilidad de ser usadas en el futuro. Esto es posible gracias a la propiedad de localidad referencial de los programas.

3.1.1 Localidad referencial, temporal y espacial

Los programas manifiestan una propiedad muy utilizada para mejorar los diseños de los sistemas de gestión de memoria de los computadores, **la localidad referencial**: los programas tienden a reutilizar las instrucciones y datos usados recientemente. Existe una regla que se suele cumplir en gran parte de los programas: un programa gasta el 90% de su tiempo de ejecución sólo en el 10% de sus instrucciones. Esto ayuda a tener una previsión bastante precisa de qué instrucciones o datos serán usados en el futuro próximo.

Localidad temporal: hay una alta probabilidad de que las palabras accedidas recientemente, sean accedidas en el futuro. Esto se debe principalmente a la presencia de bucles en la mayoría de los programas.

Localidad espacial: hay una gran probabilidad de que las palabras próximas a las que han sido referenciadas, sean referencias en un futuro muy cercano. O dicho de otra manera, los bloques de palabras suelen ser referenciados juntos en instantes de tiempo muy pequeños. Esto se debe a que los programas son secuenciales, es decir, las instrucciones se suelen ejecutar de forma lineal, excepto en el caso de los bucles. Además las estructuras de datos suelen ser accedidas también de forma secuencial.

3.1.2 Tasa de aciertos y tasa de fallos

Un factor fundamental en este tipo de sistemas es el número de fallos que se tiene al solicitar una dirección de memoria, para determinar si el dispositivo es realmente útil y permite mejorar el rendimiento en el acceso a memoria o no.

Dada la ejecución de un determinado programa, si se realiza un número N_r de referencias a memoria, de las cuales N_a son los aciertos y N_f son los fallos, tenemos:

Tasa de acierto $(T_a) = N_f / N_r$

Tasa de fallos $(T_f) = N_f / N_r$ Donde $T_a = 1 - T_f$.

3.2 Diseño

Hay varios factores que influyen de forma directa en el rendimiento de una memoria caché y por tanto es importante tenerlos en cuenta en el diseño de la misma. Son los siguientes:

3.2.1 Función de correspondencia

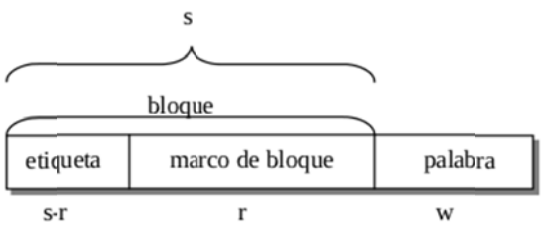
Es la encargada de decidir la posible ubicación de un bloque de memoria principal referenciado, en la memoria caché.

Existen tres funciones de correspondencia para ubicar un bloque de memoria principal (Mp) en la memoria caché (Mc): directa, asociativa y asociativa por conjuntos.

Correspondencia directa

En la correspondencia directa el bloque B_j de Mp se puede ubicar sólo en un marco de bloque que llamamos MB_i y cumple la siguiente relación: i = j mod m, donde m es el número total de líneas que tiene la caché.

A continuación podemos ver la interpretación de una dirección física para realizar la correspondencia:
De donde extraemos los siguientes datos:



2^w palabras/bloque

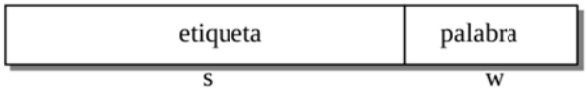
2^s bloques de Mp

2^r marcos de bloque en Mc

Correspondencia asociativa

En la correspondencia asociativa un bloque B_j de Mp se puede ubicar en cualquier marco de bloque de Mc, donde la dirección física tiene la siguiente interpretación:

De donde extraemos los siguientes datos:



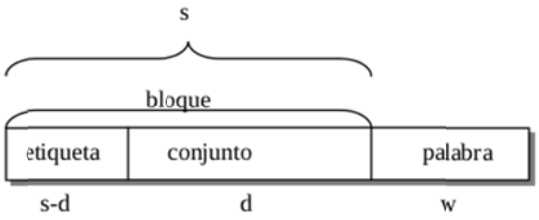
2^s bloques de Mp

2^r marcos de bloque de Mc

Correspondencia asociativa por conjuntos

En la correspondencia asociativa por conjuntos los marcos de bloque de Mc se agrupan en v=2*d conjuntos con k vías cada uno. Se cumple que el número total de marcos de bloque que tiene la caché m = v*k. Un bloque B_j de Mp se puede ubicar únicamente en el conjunto C_i de Mc que cumple la siguiente relación i = j mod v.

A continuación podemos ver la interpretación de una dirección física para realizar la correspondencia:



La etiqueta tendría s – d bits para poder diferenciar cada uno de los bloques de Mp que pueden ubicarse en el mismo conjunto de Mc. El directorio caché en correspondencia asociativa por conjuntos contendrá un registro de s - d bits por cada línea de Mc.

3.2.2 Política de reemplazamiento

Se encarga de decidir qué bloque hay que sacar de la memoria caché para poder incluir uno nuevo, en el caso de que la memoria caché se encuentre llena.

El espacio de ubicación de un bloque en memoria caché depende directamente del método de correspondencia usado.

La correspondencia directa reduce el espacio a un marco de bloque, por lo que no existe alternativa cuando hay que ubicar un nuevo bloque, y el lugar ya está decidido de antemano. Por tanto en este caso no es necesario usar una política o estrategia de reemplazamiento.

En las correspondencias asociativa y asociativa por conjuntos si es necesario decidir qué bloque hay que sustituir. En la correspondencia asociativa la elección se tiene que realizar sobre cualquier bloque presente en Mc, mientras que en la segunda se reduce al conjunto único donde puede ubicarse el nuevo bloque.

Las políticas de reemplazamiento más utilizadas son las siguientes:

FIFO (First In First Out)

Se representa con una cola, siendo el bloque sustituido, el primero en llegar de los que hay en la memoria caché.

No tiene en cuenta los usos del bloque, sino la antigüedad del mismo.

LFU (Least Frequently Used)

Se sustituye el bloque menos usado, es decir, el que menos referencias ha experimentado. Se puede implementar asociando un contador a cada marco de bloque.

LRU (Least Recently Used)

Se sustituye el bloque que hace más tiempo que no ha sido referenciado.

Para implementar esta política, se asocian contadores a los marcos de bloque:

Si se produce un acierto en el marco de bloque MB_i, se realiza lo siguiente:

Para todo MB_K:
$contador(MB_K) \leq contador(MB_i) \Longrightarrow contador(MB_K) := contado(MB_K) + 1$, $contador(MB_i) = 0$

En caso de fallo: Cuando se produce un fallo se sustituye el MB_i $contador\ (MB_i) = MÁXIMO$

3.2.3 Política de escritura

Mantiene la coherencia entre la memoria caché y la principal cuando se realizan escrituras. Frente a aciertos en la caché existen dos alternativas: escritura directa y post-escritura.

Escritura directa o inmediata (write through)

Todas las operaciones de escritura se realizan en memoria caché y memoria principal, generando bastante tráfico hacia la memoria principal. Es posible solucionar esto incluyendo un buffer de escrituras.

Postescritura (copy back)

Las actualizaciones se hacen sólo en memoria caché.

Se utiliza un flag de actualización en cada marco de bloque para indicar la escritura del marco en memoria principal cuando es sustituido por la política de reemplazamiento.

Genera inconsistencias temporales entre la memoria caché y la principal, complicando el acceso a de la E/S a memoria, que debe realizarse a través de memoria caché.

Y frente a fallos en la caché otras dos: asignación en escritura y no asignación.

Con asignación en escritura (write allocate)

El bloque se ubica en memoria caché cuando se registra un fallo de escritura y a continuación se actúa como si se hubiera producido un acierto de escritura, es decir, con “write through” o “copy back”.

Sin asignación en escritura (No write allocate)

El bloque se modifica en memoria principal sin cargarse en caché.

3.2.4 Política de búsqueda de bloques

Determina la causa que desencadena que un bloque sea llevado a la caché (habitualmente por un fallo en la referencia). Existen los siguientes tipos de búsqueda de bloques:

Bajo demanda

Se lleva un bloque a memoria caché cuando es referenciada alguna palabra del mismo desde el procesador y éste no se encuentra en la caché.

Anticipativa (prebúsqueda)

Se puede realizar siempre pre-búsqueda, realizando el siguiente proceso: cuando se busca el primer bloque, se lleva a cache también uno o varios bloques vecinos al primero, estimando que se van a usar en próximas referencias.

O por el contrario, prebúsqueda por fallo: cuando se produce un fallo al acceder a un bloque se busca este bloque y el siguiente.

3.2.5 Funcionamiento

El siguiente diagrama resume de forma gráfica el funcionamiento de una memoria caché:

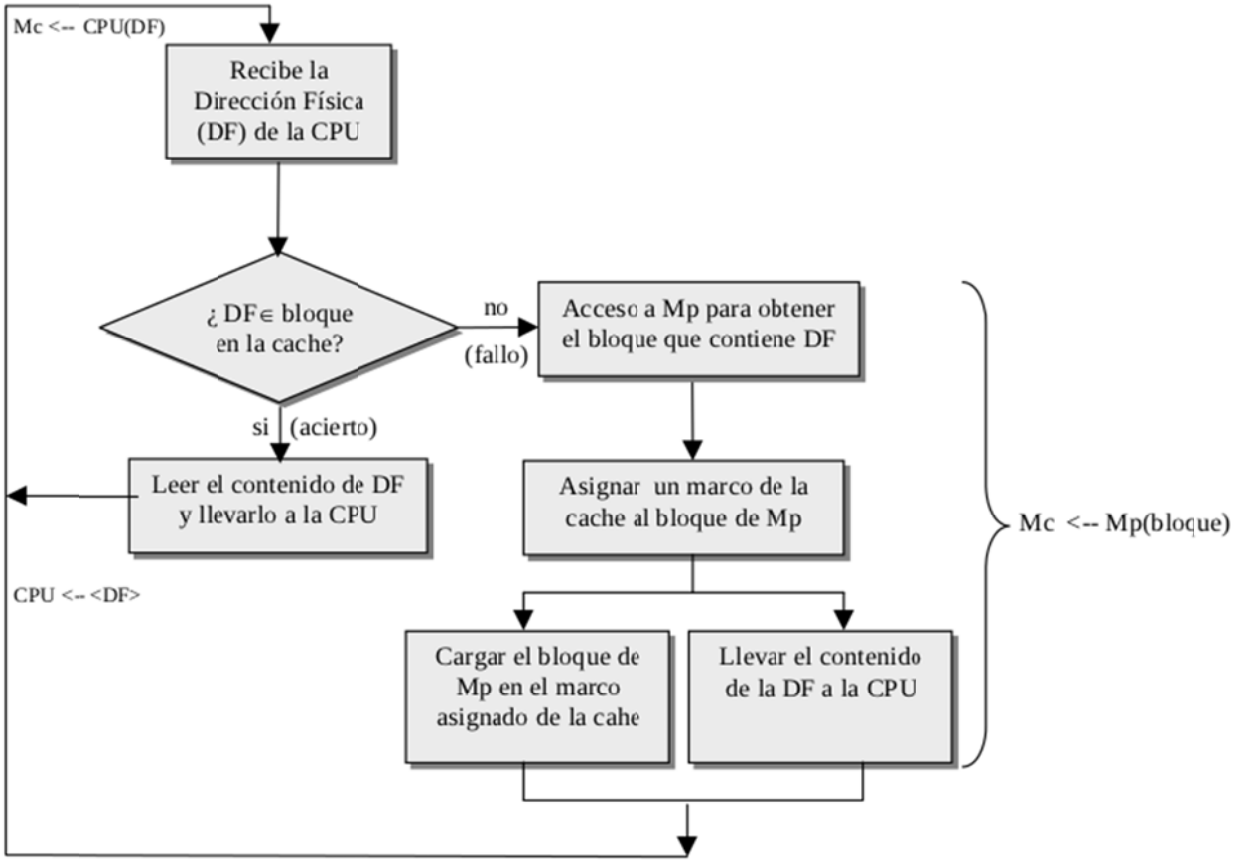


Fig 5. Esquema de funcionamiento de la Memoria Caché.

3.2.6 Rendimiento

Cuando se registra un fallo, el bloque viaja de la memoria principal a la cache y en paralelo la palabra viaja al procesador. El tiempo de acceso a memoria sería:

$$T_{acceso} = N_a * T_c + N_f * T_p$$

donde:

N_a : número de referencias con acierto

N_f : número de referencias con fallo

T_c : tiempo de acceso a una palabra de memoria caché (Mc)

T_p: tiempo de acceso a un bloque de memoria principal (Mp)

El tiempo de acceso medio durante la ejecución del programa valdrá:

$$T_{acceso\_medio} = T_{acceso} / N_r = T_a * T_c + T_f * T_p$$

donde:

$T_a = N_a / N_r$ es la tasa de aciertos

$T_f = N_f / N_r$ es la tasa de fallos

N_r : número total de referencias a memoria

A la primera componente se le denomina $T_{acierto} = T_a * T_c$

En cambio, si frente a un fallo, el bloque de Mp se lleva primero a Mc y después se lee la palabra referenciada de Mc y se lleva a la CPU, el tiempo de acceso a memoria durante la ejecución de un programa será:

$$T_{acceso} = N_r * T_c + T_f * T_p \quad y \quad T_{acceso\_medio} = T_{acceso} / N_r = * T_c + T_f * T_p$$

En este caso $T_{acierto} = T_c$

3.3 Caché en navegadores WEB

Una caché web es un tipo de cache usada en los navegadores web, dedicada a almacenar los ficheros o recursos que componen una página para reducir el tiempo de carga y el ancho de banda utilizado, en posibles nuevas visitas a una página en un futuro cercano.

3.3.1 Tipos de cachés web

Las caché web pueden usarse de diferentes maneras. Existen las caches de agente de usuario (User-Agent) como las usadas por los navegadores web. Estas cache son privadas, es decir, funcionan para un único usuario.

Los intermediarios en la comunicación cliente-servidor pueden implementar cachés compartidas (o proxy-cachés directos) que sirvan páginas a varios usuarios. Los proxy-cachés suelen ser usados por los proveedores de servicios de internet (ISP), universidades y empresas para ahorrar ancho de banda. La intermediación de estos proxy-cachés difieren de la de los privados en que los clientes no necesitan ser explícitamente configurados para usarlos.

Las caches pasarela (o proxy-cachés inversos) funcionan en el propio servidor, de forma que los clientes no distinguen unos de otros. Puede hacerse funcionar conjuntamente varias cachés pasarela para implementar una Content Delivery Network (CDN).

Los intermediarios que hacen funciones de caché realizan con frecuencia otras tareas, tales como la de filtrado de contenidos o autenticación de usuarios. Varias cachés pueden ser coordinados entre sí con las ayuda de algunos protocolos específicos como ICP o HTCP.

3.3.2 Control

El protocolo HTTP define los siguientes mecanismos para controlar las cachés:

- **Frescura**: permite que una respuesta sea usada sin comprobar de nuevo el servidor origen, y puede ser controlada tanto por el servidor como el cliente. Pueden usarse ciertas directivas para informar a la caché del tiempo máximo que debe utilizar algún recurso.

- **Validación**: puede usarse para comprobar si una respuesta cacheada sigue siendo buena tras caducar. Puede usarse la fecha de última modificación para preguntar al servidor si desde ese momento ha cambiado.

- **Invalidación**: normalmente es un efecto secundario de otra petición que pasa por la caché. Cuando una URL es solicitada de nuevo por POST, PUT o DELETE, la respuesta cacheada es invalidada.

CAPÍTULO 4: OpenNebula

Como se ha explicado anteriormente, OpenNebula es una herramienta de virtualización Open Source para la gestión de infraestructuras virtuales ofreciendo una solución flexible, completa e integral para la gestión de centros de datos virtualizados.

4.1 Componentes de OpenNebula

OpenNebula engloba un conjunto de componentes:

- Conjunto de interfaces y APIs:
  Tenemos a nuestra disposición un conjunto variado de interfaces que se pueden utilizar para la interacción y administración de los recursos físicos y virtuales.

- Usuarios y grupos:
  OpenNebula permite la creación de diferentes cuentas de usuarios y grupos así como diferentes mecanismos de autenticación.

- Hosts:
  Varios tipos de hipervisores están soportados en la administración de la virtualización. Dichos hypervisores poseen la capacidad de controlar y monitorizar el ciclo de vida de las máquinas virtuales. Actualmente son compatible los hipervisores: Xen, KVM y VMware.

- Redes:
  OpenNebula permite la adaptación y personalización de los subsistemas de redes con el fin de poseer una integración sencilla con los requisitos específicos de la red de centros de datos ya existentes. Es compatible con VLAN y Open vSwitch.

- Clusters:
  Los clusters son un grupo de hosts que comparten el almacenamiento de datos y redes virtuales. Estas agrupaciones se utilizan para equilibrar la carga, para tener una alta disponibilidad y un alto rendimiento computacional.

- Storage:
  OpenNebula es lo suficientemente flexible para compatibilizar diferentes configuraciones de almacenamientos de imágenes.

  El soporte de múltiples subsistemas de almacenamiento permite una gran flexibilidad en la planificación del back-end del almacenamiento y en las ganancias de rendimiento.

Hay dos formas de interactuar con OpenNebula: mediante consola de comandos o a través de la interfaz gráfica facilitada para la herramienta Sunstone.

Ambos recursos son de gran utilidad, la herramienta Sunstone permite al usuario interactuar de forma rápida y sencilla con OpenNebula sin la necesidad de disponer de grandes conocimientos sobre las órdenes que necesitamos ejecutar para nuestro propósito, además proporciona herramientas muy visuales y descriptivas sobre el estado actual del sistema y sus componentes.

Por otro lado, la consola permite tener gran potencia de acción a más bajo nivel una vez que el usuario se familiariza con los comandos de OpenNebula. A través de este medio, seremos capaces de realizar ciertas acciones no disponibles en entorno gráfico, como son ciertas utilidades que facilitan el desarrollo de nuevos productos.

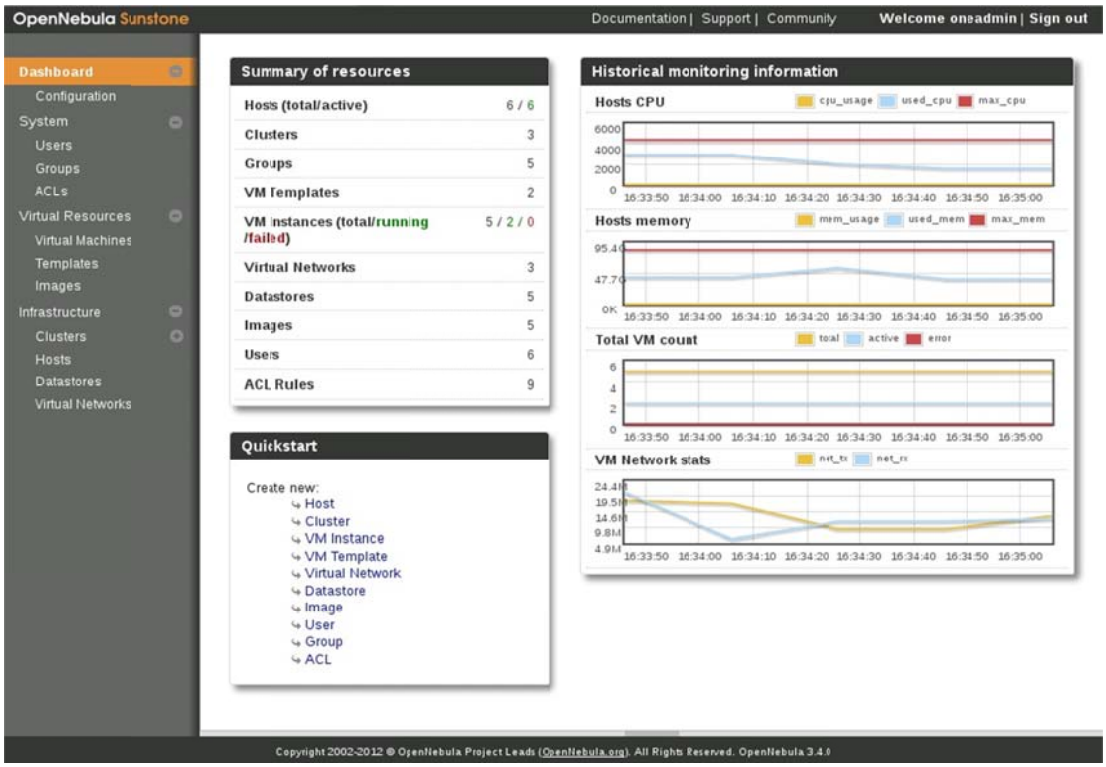


Fig 6. Herramienta Sunstone.

4.2 Arquitectura de OpenNebula

La arquitectura interna de OpenNebula está dividida en tres capas fundamentales:

- Tools
- Core
- Drivers

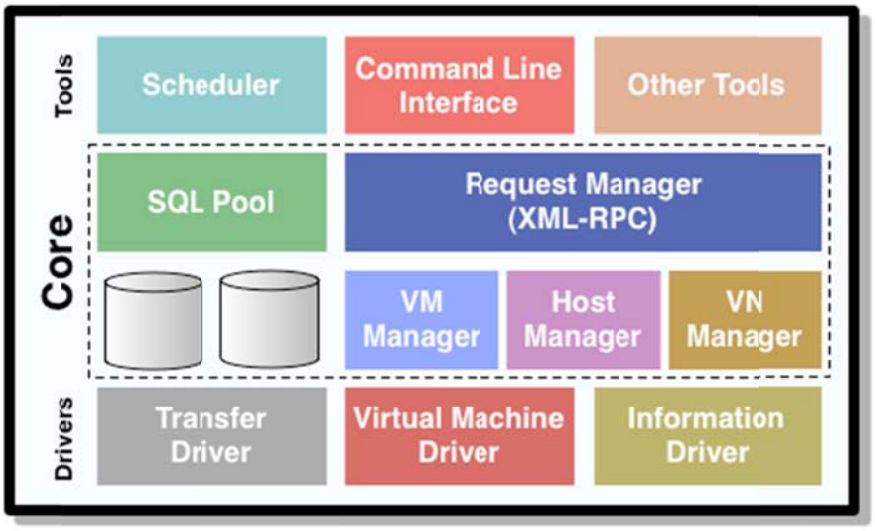


Fig 7. Arquitectura OpenNebula

4.2.1 Tools

Esta capa contiene las herramientas incluidas en OpenNebula, tales como el planificador (Scheduler), la interface de línea de comandos (CLI), y otras herramientas de menor peso.

Planificador

Se trata de una herramienta completamente desacoplada del resto de componentes de OpenNebula, pudiendo ser sustituida por otra en cualquier momento. Hace uso de una interfaz XML-RPC proporcionada por OpenNebula para realizar todo tipo de acciones sobre las máquinas virtuales. Además, el planificador permite la definición de múltiples políticas de carga.

El planificador incorporado por OpenNebula tiene como fin dar prioridad a los recursos de la máquina virtual.

Como en la mayoría de los componentes de OpenNebula, este planificador también es configurable, pudiendo utilizar los siguientes parámetros:

- Puerto de conexión a oned
- Tiempo entre dos acciones planificadas (en segundos)
- Número máximo de VM gestionadas en cada acción
- Número máximo de VM con las que comunicarse en cada acción
- Número máximo de máquinas virtuales con las que se comunica por un host dado

También es posible integrar el planificador Haizea, como módulo de planificación de tareas de OpenNebula. Haizea da la posibilidad de gestionar de forma muy eficiente los recursos destinados a las VM.

Command Line Interface

OpenNebula dispone de una serie de comandos para poder gestionar los recursos virtuales de forma rápida, pudiendo agregar, eliminar y monitorizar varios aspectos.

En concreto, estos son los principales comandos que ofrece:

- onevm: permite iniciar, monitorizar y controlar las máquinas virtuales
- onehost: añade, elimina y monitoriza los hosts
- onevnet: añade, elimina y monitoriza las redes virtuales
- oneuser: añade, elimina y monitoriza los usuarios
- oneimage: añade, elimina y controla las diferentes imágenes
- onetemplate: añade, elimina y controla las diferentes plantillas definidas
- onecluster: añade, elimina y controla los clusters
- oneauth: herramienta para la gestión de autorización y autenticación de usuarios

Además dispone de diferentes comandos para la integración de OCCI (Open Cloud Computing Interface).

4.2.2 Core

La capa “Core” está compuesta por una serie de elementos para controlar y monitorizar las máquinas virtuales, las redes virtuales y el almacenamiento.

Los principales elementos que componen el Core son los siguientes:

Request Manager (Gestor de solicitudes)

El Request Manager proporciona una interfaz XML-RPC para la invocación de sus métodos. Esta interfaz hace que no exista un acoplamiento entre el core de OpenNebula y los componentes externos (como por ejemplo el planificador de tareas).

Virtual Machine Manager (Administrador de Máquinas virtuales)

Este componente es el encargado de administrar y monitorizar las máquinas virtuales. Proporciona una visión uniforme del conjunto de recursos. Sus principales tareas son la administración de la virtualización, la creación de redes y de imágenes.

Transfer Manager (Gestor de transferencias)

El Transfer Manager se encarga de todas las operaciones de transferencia necesarias para el despliegue de la máquinas virtuales. Es decir, se encarga de la correcta transferencia de las imágenes al nodo del cluster deseado para el arranque de las máquinas virtuales, de la transferencia de la imagen desde un nodo al repositorio de imágenes, la transferencia de los ficheros de control entre diferentes nodos para la realización de migraciones o al nodo frontal cuando la máquina virtual está parada.

Virtual Network Manager (Administrador de redes virtuales)

El Virtual Network Manager es el responsable del manejo de direcciones IP y direcciones MAC, permitiendo la creación de redes virtuales, compuestas por redes públicas y privadas, que manejan de forma interna el conjunto de IPs y direcciones MAC.

Host Manager (Gestor de nodos)

Gestiona y monitoriza los nodos físicos (hosts), estando dotado de una gran flexibilidad para ser extendido y así incluir cualquier atributo del host.

Database (Base de datos)

Se trata de una base de datos persistente para almacenar los diferentes datos que necesitan todos los componentes usados por OpenNebula. Soporta tanto SQLite3 como MySQL y da a OpenNebula la seguridad y escalabilidad necesaria que necesita un gestor de este tipo.

4.2.3 Drivers

OpenNebula está compuesto además por un gran conjunto de módulos o extensiones específicas tales como hipervisores de virtualización, mecanismos de transferencia de ficheros o servicios de información. Todas estas extensiones son denominadas como Drivers y gracias a la gran comunidad de desarrolladores que colaboran en el proyecto, el número de estos Drivers cada vez es mayor y mejoran de manera considerable.

4.2.4 Infraestructura física

OpenNebula trabaja bajo una típica arquitectura basada en clusters, compuesta por un nodo frontal y una serie de nodos donde las máquinas virtuales serán ejecutadas. Debe haber al menos una red física que une todos los hosts con el nodo frontal:

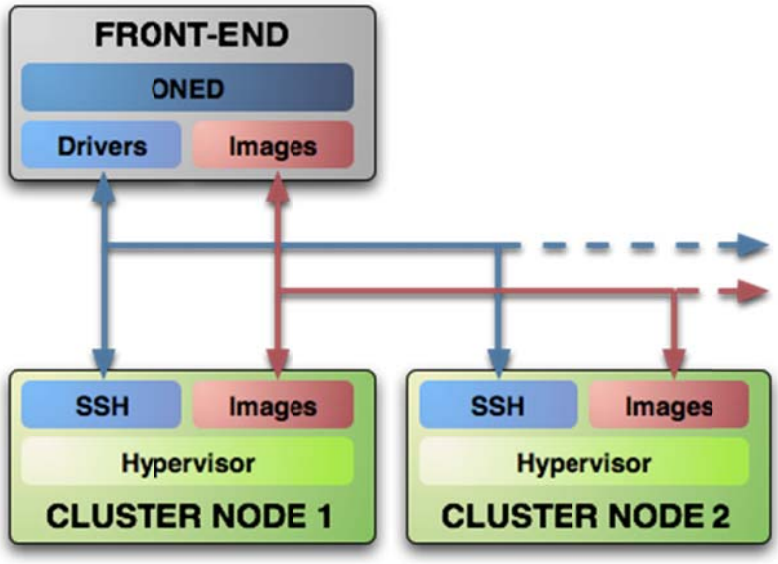


Fig 9. Ejemplo de infraestructura de red

En el nodo frontal tenemos los siguientes componentes:

ONED

Se trata del proceso “Demonio” de OpenNebula encargado de dar servicio a todas las solicitudes recibidas por las diferentes vías: CLI, API, Sunstone.

Drivers

Módulos encargados de realizar funciones específicas como la transferencia de ficheros, la gestión de información de los nodos, control de máquinas virtuales, etc.

Se dividen en tres áreas:

1. **Information Drivers**
   Los “Information Drivers” o Drivers de información se utilizan para recoger información de los nodos de los clusters, y dependen del tipo de virtualización que se esté usando. Se puede definir más de un gestor de información, pero siempre con diferentes nombres.

2. **Transfer Drivers**
   Los controladores de transferencia se utilizan para transferir, clonar, eliminar y crear imágenes de máquinas virtuales. El Driver “TM_MAD” por defecto incluye plugins para todos los modos de almacenamiento compatibles. Existen varios tipos de Transfer Manager con características distintas.

3. **Virtualization Drivers**
   Los controladores de virtualización se utilizan para crear, controlar y monitorizar las máquinas virtuales en los nodos del clúster. Se puede definir más de un Driver de Virtualización dándole a cada uno un nombre diferente.

DataStore

Se trata de un directorio que contiene las imágenes de las máquinas virtuales.

4.3 Descripción del sistema de transferencia, ciclo de vida de las imágenes y storage

Uno de los aspectos clave de la gestión de la virtualización son los procesos que tratan con las imágenes de las máquinas virtuales.

A causa de las características específicas de cada usuario, la gestión de las imágenes de las máquinas virtuales varían dependiendo de las necesidades de cada usuario.

Por ejemplo, si se desea tener el almacenamiento de las imágenes en un repositorio independiente con acceso HTTP, o se quiere compartir las imágenes mediante NFS entre todos los ordenadores.

Como vemos pueden existir diferentes variantes, y OpenNebula intenta ser flexible para adaptarse a la gran mayoría de ellas.

El modelo de almacenamiento de OpenNebula organiza las imágenes de las siguientes formas:

- DataStore: se refiere a cualquier medio de almacenamiento local o remoto, que poseen las imágenes de las máquinas virtuales. Un repositorio de imágenes puede un servidor de archivos dedicado o una URL remota, ambos tienen que ser accesibles desde el nodo frontal de OpenNebula.

- Directorio de Máquinas Virtuales: directorio ubicado en el nodo en el que se almacenará la máquina virtual en funcionamiento. Se encuentra ubicado en <VM_DIR>/<VID>. <VM_DIR> es la ruta definida en el archivo de configuración del demonio de OpenNebula (oned.conf) y <VID> hace referencia al identificador de la máquina virtual.

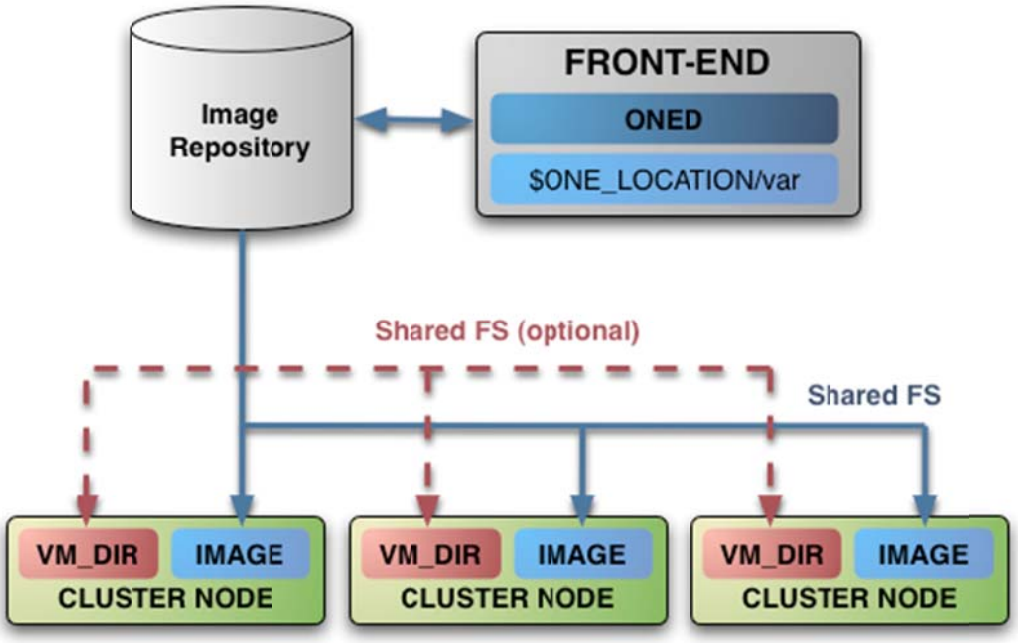


Fig 10. Ejemplo de infraestructura de red

Una instalación de OpenNebula puede contener múltiples almacenes de diferente tipo para guardar imágenes.

Pero para comprender el funcionamiento que realiza el sistema de almacenamiento de las imágenes debemos entender antes su ciclo de vida.

4.3.1 Ciclo de vida de las Imágenes

Cualquier imagen de una máquina virtual que se utilice en OpenNebula, transcurre por los siguientes estados:

- Preparación: implica todos los cambios necesarios que deben hacerse a la imagen de la máquina para que pueda ofrecer el servicio que se pretende.
  OpenNebula asume que la imagen o imágenes que conforman una máquina virtual, se preparan y se colocan en el repositorio de imágenes accesibles.

- Clonado: este estado coge la imagen del repositorio y lo coloca en el directorio de la máquina virtual antes de sea iniciada. La clonación de la imagen se puede hacer mediante enlaces simbólicos o mediante el copiado de la imagen al directorio de la máquina virtual.

- Guardado / Borrado: llegamos a este último estado cuando deseamos apagar la máquina virtual. Si hemos configurado la máquina virtual para que cuando se apague no se borre, se guardará dicha configuración en el directorio de la máquina virtual, si no se realizará la eliminación de toda la configuración y de la imagen virtual.

Como hemos comentado anteriormente OpenNebula es adaptable a los diferentes sistemas de almacenamientos que los usuarios pueden demandar dependiendo de las distintas premisas de trabajo en un sistema real. A continuación describimos los tipos soportados de almacenamiento que OpenNebula puede ofrecer:

Tipos de Almacenamientos

La última versión estable de OpenNebula suministra 4 tipos de Datastores:

- Sistema. Mantiene las imágenes de máquinas virtuales en ejecución, en función de la tecnología de almacenamiento utilizada estas imágenes temporales pueden ser una copia completa de la imagen original, o simples enlaces del sistema de archivos.

- Sistema de archivos, para almacenar imágenes de disco en un formato de archivo. Los archivos se almacenan en un directorio montado desde un servidor SAN / NAS.

- iSCSI / LVM, para guardar imágenes de disco en forma de dispositivo de bloque. Las imágenes son presentados a los hosts como targets iSCSI.

- VMware, un almacén de datos específico para el hipervisor de VMware que utiliza el formato vmdk.

Todos los tipos anteriormente citados de almacenes de datos necesitan un gestor que realice o enlace los archivos de imágenes, y el módulo que gestiona dicha operativa en OpenNebula se llama Transfer Manager (TM).

OpenNebula posee un subsistema de almacenamiento muy flexible. Estos drivers se organizan en dos grandes grupos:

- DS: Drivers del almacén de datos. Estos sirven para la administración de imágenes: registro, eliminación y crear datablocks vacías.

- TM: Drivers de transferencia. Estos sirven para la administración de las imágenes asociadas a las máquinas virtuales, y por tanto el objetivo de nuestro plugin será desarrollarlos.

4.3.2 Transfer Manager (TM)

Las imágenes de discos registrados en un almacén de datos se transfieren a los hosts a través de los drivers que proporciona el Transfer Manager (TM).

En la última versión de OpenNebula el mecanismo de transferencia se define para cada almacén de datos. De este modo, un único host puede acceder simultáneamente a múltiples almacenes de datos que utilicen diferentes controladores de transferencia realizando una ganancia notable de rendimiento.

Para la distribución de imágenes desde el almacén de datos a los host, OpenNebula proporciona cuatro posibilidades:

- Modo compartido (shared): El almacén de datos se exporta en un sistema de ficheros compartido con los hosts. Esta disposición del modelo de almacenamiento supone que el <VM_DIR> se comparte entre todos los nodos y el nodo frontal de OpenNebula.

- Modo SSH: Las imágenes del repositorio se copian a los nodos usando el protocolo ssh, es decir en este caso el <VM_DIR> no se comparte entre el Nodo frontal y los nodos, sino que existirán dos posibilidades distintas:

    - Cloning: Las imágenes se copian desde los repositorios al nodo en el que posteriormente se ejecutarán.

    - Saving: Si se activa esta opción la imagen se guardará de nuevo en el repositorio de almacenamiento en el momento en el que se apague la máquina virtual (se iniciará la copia de la imagen desde el nodo al repositorio de imágenes), sino se activa, la imagen será borrada.

- Modo iSCSI/LVM : OpenNebula soporta LVM, de tal forma que puede emplearse el snapshotting. En este caso el driver de transferencia asume que los dispositivos de bloques que se han definido en la plantilla de instanciación de la máquina virtual se encuentran también disponibles en el nodo. Esto puede efectuarse de forma manual creando el dispositivo directamente en el nodo correspondiente o empleando técnicas más sofisticadas.

- Modo VMWare: la copia de las imágenes se realizarán mediante las herramientas del sistema de archivos vmdk (Virtual Machine Disk Format).

Estructura de los drivers de transferencia

A continuación describiremos una lista con los controladores genéricos que posee cualquier driver que se desee desarrollar para TM y su acción.

- **clone**: clona las imágenes desde el almacén de imágenes (datastore). Es utilizado para imágenes no persistentes.
  - ARGUMENTOS: clone fe:SOURCE host:remote_system_ds/disk.i size
  - **fe**: Nombre de la máquina que ejecuta el nodo frontal.
  - **SOURCE**: camino donde se encuentra la imagen a ser clonada.
  - **host**: host destino donde se desplegará la máquina virtual.
  - **remote_system_ds**: camino del sistema de almacenamiento del host.

- **ln**: Enlaza la imagen del nodo a la del almacen de imágenes. Este comando es utilizado para imágenes persistentes.
  - ARGUMENTOS: ln fe:SOURCE host:remote_system_ds/disk.i size
  - **fe**: Nombre de la máquina que ejecuta el nodo frontal.
  - **SOURCE**: camino donde se encuentra la imagen a ser enlazada.
  - **host**: host destino donde se desplegará la máquina virtual.
  - **remote_system_ds**: camino del sistema de almacenamiento del host.

- **mvds**: Mueve la imagen de nuevo hacia su almacén de datos. Se utiliza para las imágenes que se quieren marcar como persistentes o salvar su estado.
  - ARGUMENTOS: host:remote_system_ds/disk.i fe:SOURCE
  - **fe**: Nombre de la máquina que ejecuta el nodo frontal.
  - **SOURCE**: camino donde se encuentra la imagen.
  - **host**: host destino donde se despliega la máquina virtual.
  - **remote_system_ds**: camino del sistema de almacenamiento del host.

- **mv**: Mueve imágenes o directorios entre diferentes hosts. Cuando se utiliza para mover una imagen sólo se especifica dicho argumento.
  - ARGUMENTOS: `hostA:system_ds/disk.i|hostB:system_ds/disk.i <OR> hostA:system_ds/|hostB:system_ds/`
  - **hostA**: Nombre de la máquina donde se encuentra la máquina virtual.
  - **hostB**: máquina donde se desplegará la máquina virtual.
  - **remote_system_ds**: camino del sistema de almacenamiento del host.
- **context**: Crea una ISO que contendrá todos los archivos pasados como argumentos.
  - ARGUMENTOS: `file1 file2 … fileN host:remote_system_ds/disk.i`
  - **host**: host destino donde se despliega la máquina virtual.
  - **remote_system_ds**: camino del sistema de almacenamiento del host.

- **delete**: Elimina el directorio que almacena los datos de la máquina virtual, ya sea la imagen o un disco que utilice
  - ARGUMENTOS: `host:remote_system_ds/disk.i|host:remote_system_ds/`
  - **host**: host destino donde se despliega la máquina virtual.
  - **remote_system_ds**: camino del sistema de almacenamiento del host.

- **mkimage**: Crea una imagen al-vuelo
  - ARGUMENTOS: `size format host:remote_system_ds/disk.i`
  - **size**: Tamaño de la imagen en MB
  - **format**: formato de la imagen.
  - **host**: host destino donde se despliega la máquina virtual.
  - **remote_system_ds**: camino del sistema de almacenamiento del host.

- **mkswap**:
  - ARGUMENTOS: `size host:remote_system_ds/disk.i size`
  - **size**: Tamaño de la imagen en MB
  - **host**: host destino donde se despliega la máquina virtual.
  - **remote_system_ds**: camino del sistema de almacenamiento del host.

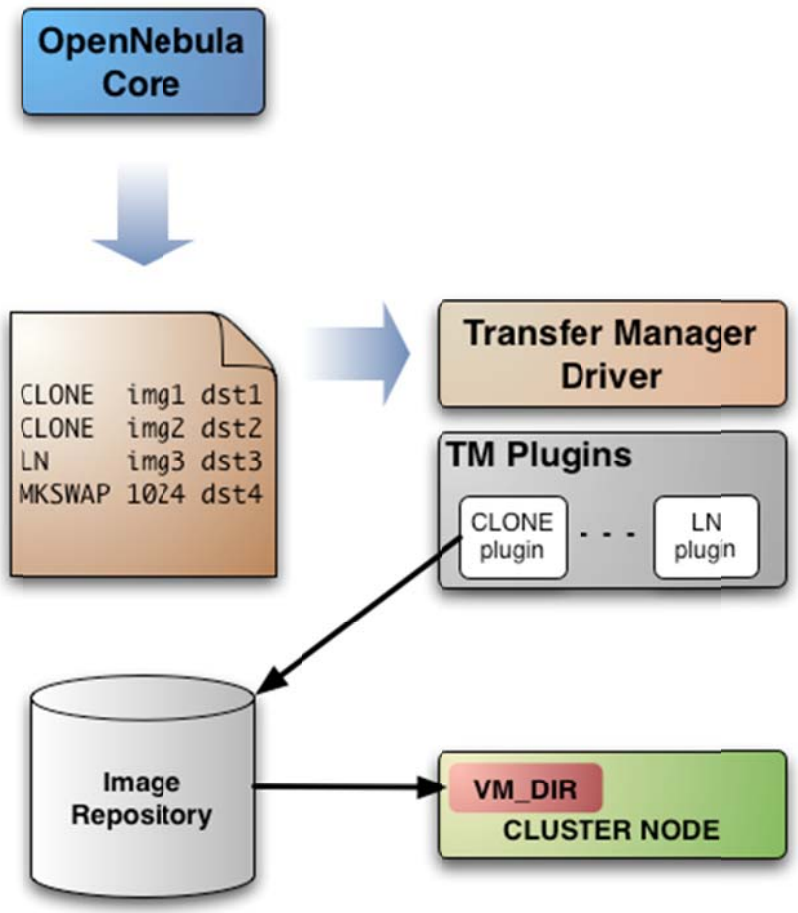


Fig 11. Arquitectura módulo de transferencia

CAPÍTULO 5: Diseño

En este capítulo repasamos el diseño del driver “oneCache” desarrollado, su organización y la forma en la que se desarrollan las operaciones para su funcionamiento.

El módulo que se encarga de la administración de la transferencia de archivos en el Datastore (almacén de imágenes) es el Transfer Manager.

Por tanto tendremos que fijar en dicho módulo el comportamiento que realizará nuestro plugin.

En el siguiente diagrama se muestran las principales funciones que realiza:

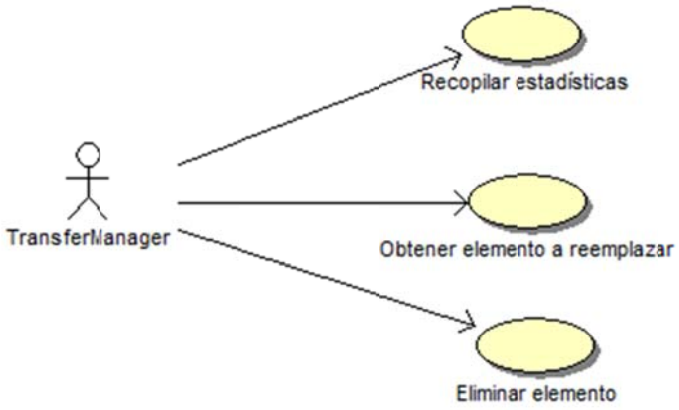


Fig 12. Funciones del módulo de transferencia tm_cache

- **Recopilar estadísticas**: Nuestro plugin hace uso de un servidor implementado como base de datos en MySQL, el cual mantendrá estadísticas necesarias para la administración de las imágenes a través de la memoria caché. El transfer manager se encargará de la administración de dicho servidor creando, borrando y actualizando las estadísticas de cada imagen que es o ha sido desplegadas en las máquinas virtuales.

- **Obtener elemento a reemplazar**: Con la información del servidor y las políticas de reemplazo, el Transfer Manager es capaz de disponer de una lista de imágenes que deben de abandonar la memoria caché para poder insertar una nueva.

- **Eliminar elemento:** Una vez que el Transfer Manager dispone de cuál es la imagen elegida para abandonar la memoria caché pasa a ser eliminada liberando el espacio en la memoria.

Estas funciones se añaden a las ya implementadas por el módulo: clonado, enlazado y creación de imágenes, las cuales serán utilizadas por las nuevas funcionalidades descritas.

5.1 Arquitectura

OpenNebula ofrece una gran modularidad y flexibilidad para el desarrollo de un plugin.

Como se describió en el módulo de Transfer Manager, debemos definir sobre todo el manejo que se va a realizar para la funcionalidad del clonado y copiado de imágenes a los nodos. En nuestro caso hemos definido esta funcionalidad como tm_clone y se describe su funcionamiento con el siguiente diagrama de flujo, de comunicación y una breve explicación del mismo:

Solicitud de Imagen por parte del host.

1. Comprobación de MD5 en el servidor.
    a. Si existe, se comprueba existencia del archivo en el equipo.
        i. Si existe se copia en el almacén de imágenes (datastore).
        ii. Si no existe se elimina de la bbdd del servidor la entrada y se continúa con el paso b.
    b. Si no existe se comprueba tamaño de caché.
        i. Si el tamaño < max se copia a caché, al directorio de la máquina virtual y se registra en el servidor.
        ii. Si el tamaño es > max, se ejecuta política de reemplazo y ejecuta paso i

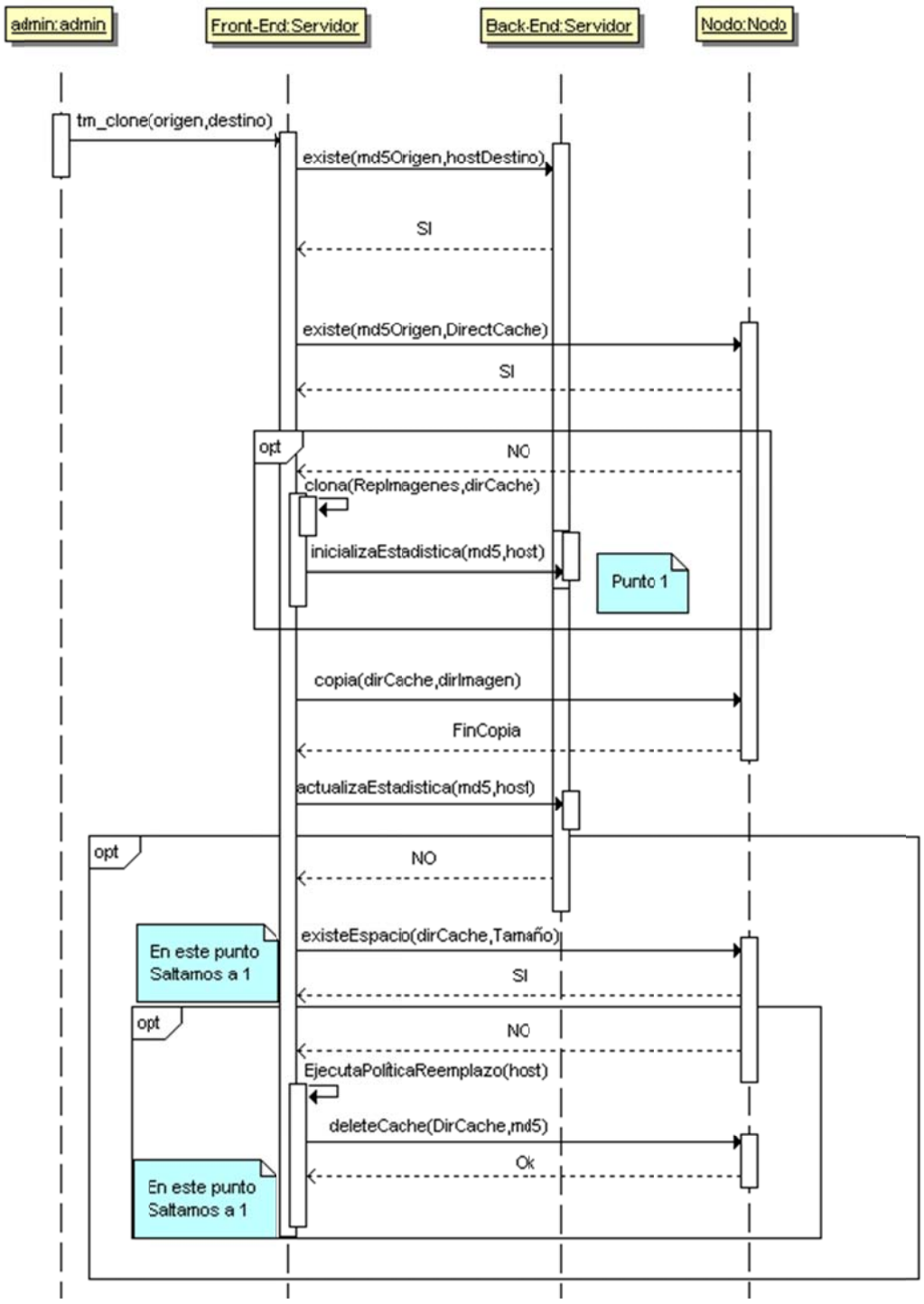


Fig 13. Diagrama de comunicación tm_clone

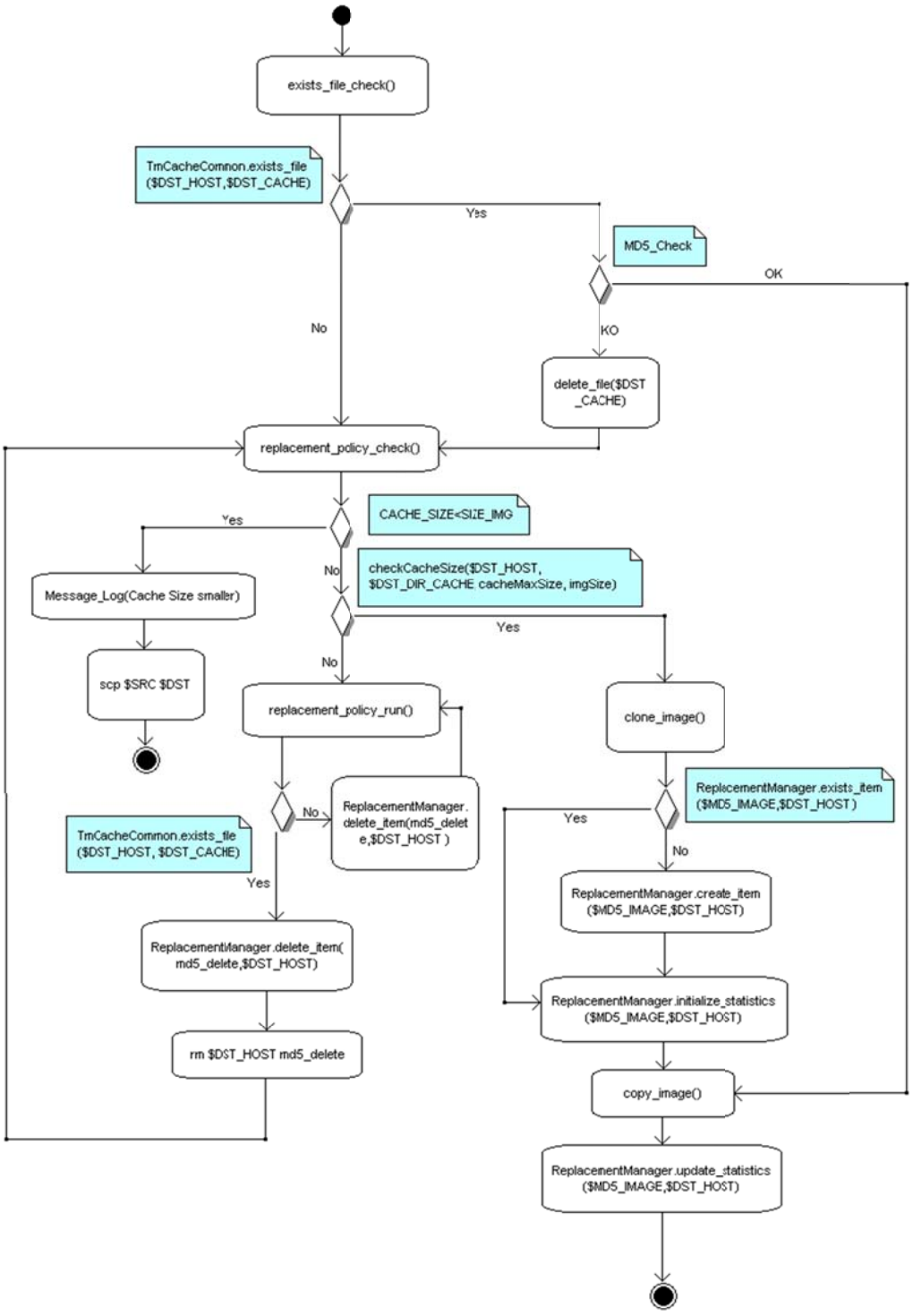


Fig 14. Diagrama de Flujo tm_clone

Dicho copiado de imágenes se apoya en la utilización de una memoria caché y de un servidor de estadísticas diseñado en base de datos que tiene la siguiente estructura:

host_ name	md5	fifo_ create_ date	lu_ number_ of_uses	lru_ last_time_ referenced	lfu_ number_of_uses	lfu_ age

- **host_name:** nombre del host en el que se ha desplegado la imagen.
- **md5:** md5 de la imagen de la máquina virtual.
- **fifo_create_date**: nos proporciona la fecha en la que la imagen ha sido copiada en la memoria caché. Dicha información es de gran utilidad para la política de reemplazo FIFO (First Input First Output) que describiremos más adelante.
- **lu_number_of_uses**: número de usos que ha realizado dicho host con dicha imagen. Dicha estadística es de gran utilidad para la política de reemplazo LU (Less Uses).
- **lru_last_time_referenced**: nos muestra la fecha de la última vez que el host utilize dicha imagen. Dicha información es de gran es utilizada en la política de reemplazo de LRU (Least Recently Used).
- **lfu_number_of_uses**: número de usos que ha realizado dicho host con dicha imagen. Dicho elemento es gestionado por la política de reemplazo LFU (Least Frecuently Used). Se usa para actualizar el campo de edad, que usa esta política para determinar cuál es el elemento a candidato a ser sustituido. Concretamente, cuando se produce un acierto en caché, este valor se suma a todos los campos lfu_age, menos en el caso del elemento acertado.
- **lfu_age**: es un valor que se usa para determinar cuál es el item que ha sido usado más recientemente. Cuando se produce un acierto en caché, este campo se actualiza dividiendo el valor que tenía hasta el momento entre 2. A todos los elementos, menos al que ha sido referenciado, se le suma el campo de lfu_number_of_uses, para hacer que en esta política se tenga en cuenta tanto el número de usos, como el momento de tiempo.

Los atributos que identifican de forma única cada item de la base de datos (primary_key) son host_name + md5. Y por tanto existe la restricción de que no puede existir dos ítems iguales con el mismo host y el mismo md5 de la imagen.

5.2 Ficheros

Ofrecemos a continuación una relación breve de los directorios y ficheros que componen el driver, con una pequeña descripción de las funciones de cada uno.

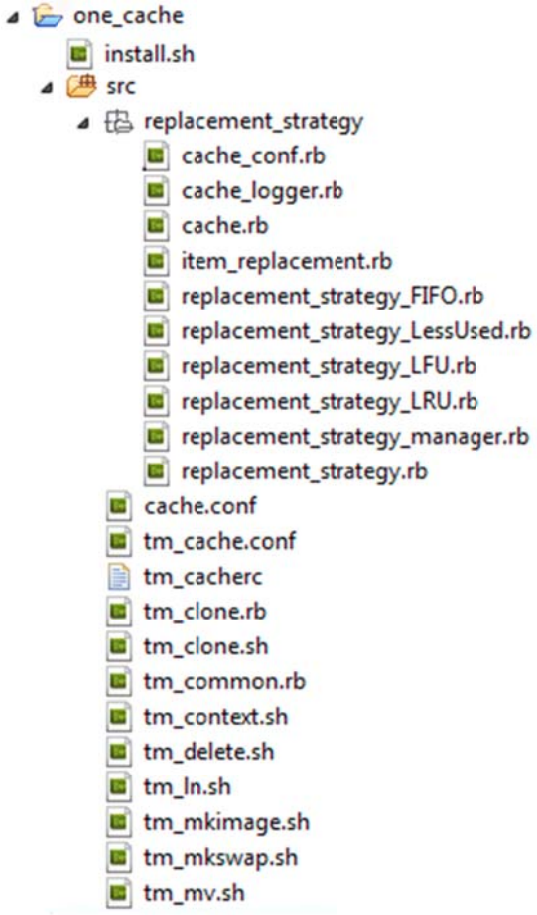


Fig 15. Estructura ficheros

- install.sh : shell que realiza la instalación de nuestro plugin.
- src: directorio fuente que contiene todo los ficheros necesarios para la instalación.
    - cache.conf : fichero de configuración donde se especifica el tamaño en KB de la memoria caché que se va a utilizar.
    - tm_cache.conf: fichero de configuración donde se especifica los ficheros que van a implementar las funcionalidades que necesita el Transfer Manager para la administración de las imágenes.
    - tm_cacherc: fichero de configuración donde se especifica si se desea habilitar o deshabilitar la inclusión de ficheros para otras imágenes de contexto que las que se especifica en tm_context.sh. Por defecto esta deshabilitado.

    - tm_clone.rb: script de ruby encargado de la implementación de las acciones sobre las copias y la gestión de las imágenes con la memoria caché.
      Este fichero contiene el grueso de nuestra implementación ya que posee las directrices para el manejo de la memoria caché. Implementa la memoria cache mediante la clase TmCache.

    - tm_clone.sh: shell que se encarga del copiado físico de la imagen del directorio ORIGEN al directorio DESTINO.

    - tm_common.rb: script de ruby que implementa las funciones comunes, es decir aquellas que pueden ser utilizadas de manera común en varias partes del driver.(Ej: TmCache).Implementa la clase TmCacheCommon.

    - tm_context.sh: Crea una ISO que contendrá todos los archivos pasados como argumentos.

    - tm_delete.sh: shell que se encarga del borrado físico de la imagen.

    - tm_ln.sh: shell que se encarga de crear un enlace simbólico en el DESTINO con punto en el ORIGEN.

    - tm_mkimage.sh: shell que crea una imagen en el DESTINO y la rellena con los ficheros que contiene el directorio ORIGEN.

    - tm_mkswap: shell que genera una imagen swap en el DESTINO. El tamaño de la imagen está especificado en el ORIGEN en MB.

    - tm_mv.sh: shell que mueve del ORIGEN al DESTINO.

- replacement_strategy: directorio donde se encuentran todos los ficheros para la implantación de las estrategias de reemplazamiento necesarias para la lógica de administración de la memoria caché.

    - cache.rb: script de Ruby que se encarga de la conexión y abstracción de las operaciones realizadas sobre la base de datos alojada en un servidor MySQL. Para la gestión con MySQL se ha utilizado el patrón de diseño de Active Record, siendo esta una capa de mapeo objeto-relacional (object-relational mapping layer) responsable tanto de la interoperabilidad entre la aplicación y la base de datos como de la abstracción de los datos.

    - cache_conf.rb: fichero de configuración de la memoria caché escrito en Ruby. A continuación hacemos un breve descripción de las variables a configurar en dicho fichero:

Variables a configurar para la base de datos.

- adapter: tipo de motor de la base de datos.
  Active Record soporta varios motores como DB2, Firebird, Frontbase, MySQL, Openbase, Oracle, Postgres, SQLite, SQL Server, and Sybase databases.
  En nuestro caso hemos desarrollado el servidor en MySQL y por tanto esta variable siempre tendrá el siguiente valor:
  "adapter" => "mysql"

- host: nombre del host que se encarga de la administración de la memoria caché.

- username: nombre de usuario que va a acceder a la base de datos. Hay que tener en cuenta que dicho usuario deberá tener los permisos correspondientes en MySQL.

- password: contraseña del usuario que accede a la base de datos.

- loggermode: indica el modo de autenticación que se va a usar para realizar la conexión al servidor de base de datos.

Variables a configurar para la configuración de la memoria caché.

- numcandidates: número de imágenes candidatas que a través de las políticas de reemplazo serán propuestas para abandonar la memoria caché.

- strategy<*Name*>active: configura si la estrategia con nombre <Name> se encuentra activa. En caso afirmativo tendrá el valor “Yes” sino “No”.

- strategy<Name>weight: variable que asigna el peso que tendrá la estrategia de reemplazamiento <Name> en el algoritmo de reemplazo de imágenes. Dicho valor está comprendido entre 0 y 1.

- cache_logger.rb: fichero de configuración escrito en ruby que implementa la clase CacheLogger que se encarga del log. Por defecto está configurado en modo DEBUG.

- item_replacement.rb: fichero escrito en Ruby que implementa la clase ItemReplacement la cual representa la imagen que va a ser reemplazada mediante el nombre del host que desplegó la imagen, el md5 de la imagen y la puntuación que tiene para ser eliminada de la caché.

- replacement_strategy.rb: fichero escrito en ruby que implementa la interface que deben de seguir todas las estrategias de reemplazo.
- replacement_strategy_FIFO.rb: fichero escrito en ruby que implementa la estrategia FIFO (First Input First Ouput) de reemplazamiento.

- replacement_strategy_LessUsed.rb: fichero escrito en ruby que implementa la estrategia Less Used de reemplazamiento.

- replacement_strategy_LFU.rb: fichero escrito en ruby que implementa la estrategia LFU (Less Frequently Used) de reemplazamiento.

- replacement_strategy_LRU.rb: fichero escrito en ruby que implementa la estrategia LRU (Least Recently Used) de reemplazamiento.

- replacement_strategy_manager.rb: fichero escrito en ruby que implementa la clase ReplacementStrategyManager encargada de la gestión y administración de las estrategias de reemplazo implementadas anteriormente.
  Dicha clase generará una lista de políticas de reemplazamiento, tomando aquellas que estén configuradas como activas en el fichero de configuración cache_conf.rb descrito anteriormente.

5.3 Políticas de reemplazamiento

Una política de reemplazo se encuentran en todas las formas de cachés existentes y este caso no es la excepción. Para obtener un buen desempeño se debe afinar la caché con alguna política de reemplazo que se ajuste a las necesidades del usuario final.

La función principal de una política de reemplazo se basa en la forma de determinar cuál es el objeto (en nuestro caso imagen) que será el siguiente en ser reemplazado o eliminado de la caché cuando se requiera.

Las políticas de reemplazo utilizan diferentes propiedades de los objetos para determinar cuál es el más apto para ser eliminado; algunas de las propiedades más utilizadas son edad (tiempo que ha permanecido en la caché), tamaño del objeto, cantidad de solicitudes y tipo de objeto entre otros.

El principal objetivo al utilizar una caché es minimizar dos parámetros: la latencia que tiene un objeto al ser presentado al usuario final (en este caso al host que desea desplegar la imagen) y reducir el tráfico de transferencia de imágenes.

La tasa de aciertos y el byte de tasa de aciertos son dos medidas muy utilizadas que describen la efectividad de una memoria caché. La tasa de acierto representa el porcentaje de solicitudes que fueron atendidas gracias a que en caché se encontraba una copia de la imagen solicitada. El byte de tasa de acierto es el porcentaje total de datos que fueron transmitidos a partir de la caché.

Si se quiere disminuir el número de peticiones a los servidores remotos (almacén de imágenes o datastore) , se debe incrementar la tasa de acierto, mientras que si se quiere disminuir el ancho de banda requerido para las peticiones entonces se debe aumentar el byte de tasa de acierto.

Hoy en día existe la tendencia en el comportamiento de una caché y es que tanto las medidas de la tasa de acierto como el byte de la tasa de acierto rondan un rango del 30% - 50%, según varios estudios realizados.

A continuación se presentan las políticas de reemplazo que se han desarrollado :

- FIFO : este algoritmo significa, por sus siglas en inglés, que el primer objeto en entrar en la caché será el primer objeto en salir de ella. Esta política toma como parámetro para eliminar un objeto la edad del objeto en la caché, mientras más tiempo permanezca en la caché, mayor es su probabilidad de ser eliminado.

  En nuestro caso concreto, dicho algoritmo selecciona una lista de imágenes ordenada de mayor a menor prioridad de ser eliminada de la caché.

  El tiempo que las imágenes permanecen en la caché se calculada calculando la diferencia entre la fecha en la que se solicita espacio en ese momento y la fecha de entrada en la memoria caché.

  Como hemos mencionado anteriormente dicha política se encuentra implementada en el fichero replacement_strategy_FIFO.rb

- Less Used (Menos Usado): esta política de reemplazamiento elimina el objeto que menos referencias a tenido en la memoria caché. A igual número de referencias se escogerá cualquiera.

  En nuestro plugin, existe un contador de referencias en el servidor MySQL que contabiliza el número de solicitudes que tiene cada imagen md5 en el host. Por tanto a menor número de referencias más probabilidades habrá de que abandone la caché y construyendo así el método de ordenación para una lista ordenada de imágenes para abandonar la caché. El atributo que recoge dicha información de cada imagen desplegada es lu_number_of_uses.

  Esta política está implementada en el fichero replacement_strategy_LessUsed.rb.

- LRU : la política de reemplazo LRU busca un mejor rendimiento de la caché, utilizando como parámetro de reemplazo el tiempo que ha permanecido un objeto sin haber sido solicitado de nuevo. Es decir, el objeto que menos ha sido utilizado recientemente es el candidato para ser reemplazado de la caché.

  lru_last_time_referenced recoge el última vez que una imagen ha sido referenciada por el host pertinente y será este atributo mediante la ordenación ascendente el que indique aquellas imágenes con más probabilidad a abandonar la caché.

  Esta política está implementada en el fichero replacement_strategy_LRU.rb.

- LFU: este algoritmo significa, por sus siglas en inglés, que el objeto que más probabilidad tiene de abandonar la caché es aquel que menos frecuentemente ha sido utilizado.

  Se utiliza un campo que almacena el número de usos y otro de edad. Estos campos se actualizan de la siguiente forma al tener un acierto:

  Para el elemento referenciado i tenemos que:

  $$\mathrm{edad(t)} = \frac{\mathrm{edad(t-1)}}{2}$$

  Y para todo elemento distinto de i:

  $$\mathrm{edad(t)} = \mathrm{R} + \frac{\mathrm{edad(t-1)}}{2}$$

  donde R es el número de usos de ese elemento.

  Es decir, el campo de edad disminuye para los elementos usados recientemente, y aumenta para el resto. Por eso los candidatos a ser sustituidos son aquellos elementos con más edad.
  En nuestro caso concreto el atributo del servidor que ordena la lista de imágenes por orden de probabilidad a abandonar la cache es el que mayor lfu_age posea.
  Esta política está implementada en el fichero replacement_strategy_LFU.rb.

Todas las políticas descritas anteriormente proponen una lista con las imágenes que deberían abandonar la caché. Dichas elecciones se pueden combinar configurando los pesos que van a tener cada política en la decisión final que tome el gestor de imágenes.
A continuación mostramos la estructura y organización que tiene las políticas de reemplazamiento en nuestro plugin.

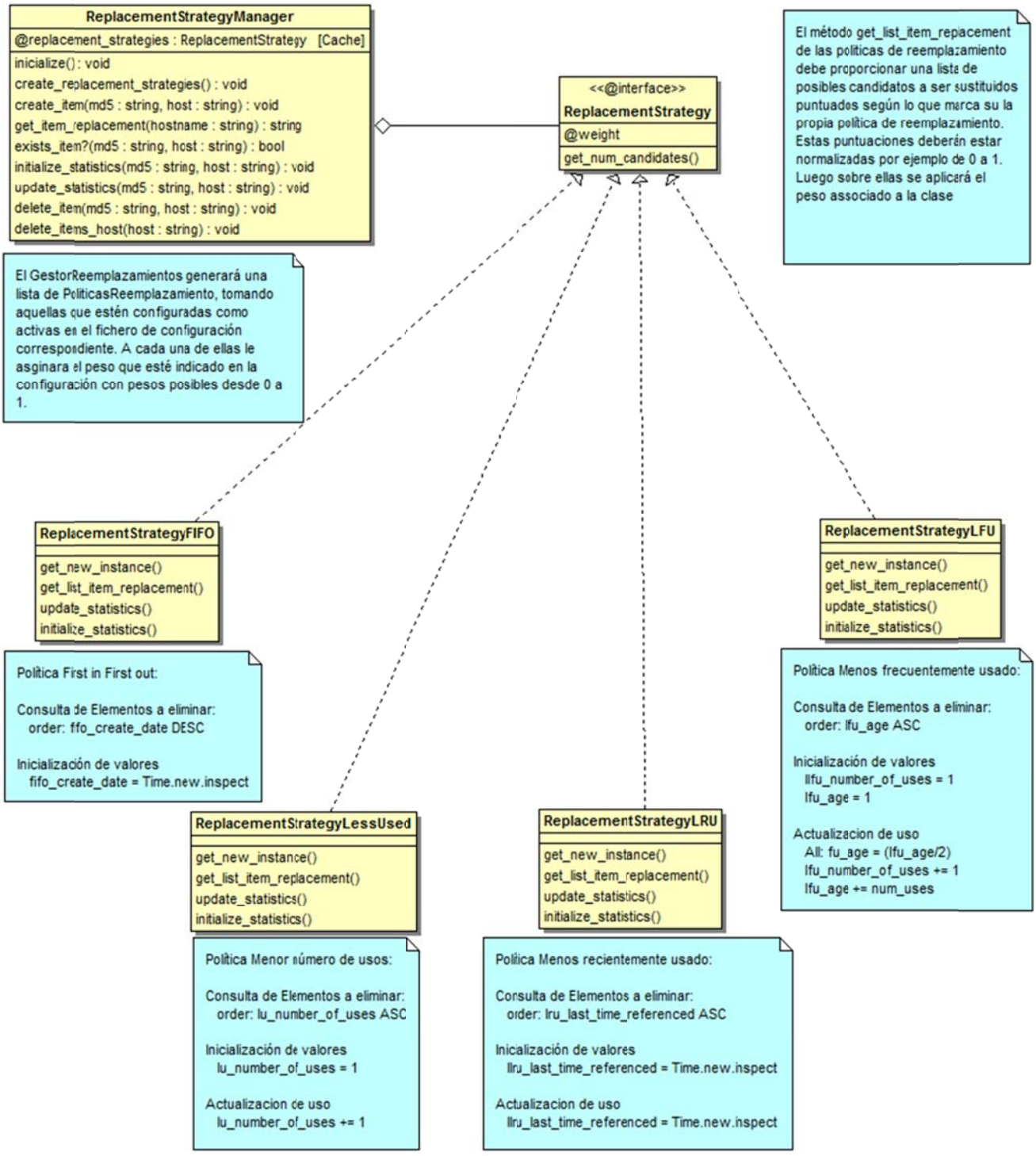


Fig 16. Estructura políticas de reemplazamiento

5.3.1 Como añadir nuevas políticas de reemplazamiento

Como podemos ver en el diagrama anterior se ha diseñado la estructura de las políticas de reemplazo para que sean lo más modulable y flexible posible, permitiendo en pocos pasos tanto su configuración, administración e incremento de forma clara y sencilla.

A continuación nombraremos los pasos necesarios para agregar y configurar nuevas políticas de reemplazamiento en nuestro plugin.

- La implementación de la nueva política se deberá generar en un archivo en el siguiente directorio del instalador:
  src / replacement_strategy

- Si fuera necesario, se debería modificar la estructura del servidor MySQL añadiendo aquellas columnas necesarias para albergar los nuevos parámetros que describen la política a utilizar.

- Deberá implementar la interface ReplacementStrategy e implementar los siguientes métodos:

  - get_new_instance(): mirará si en la configuración de la cache (cache_conf.rb), se encuentra activa dicha estrategia y su peso. Creará una nueva instancia.

  - get_list_item_replacement(hostname): proporcionará una lista de posibles candidatos a ser sustituidos puntuados según disponga el funcionamiento de la política.

  - update_statistics(md5, host): deberá implementar como se actualizan las estadísticas en el servidor.

  - initialize_statistics(md5, host): deberá implementar como se inician las estadísticas en el servidor.

- Activar la política de reemplazo en el fichero cache_conf.rb y asignarle un peso.

- Añadir la política de reemplazo en el contenedor de políticas del gestor, implementado en replacement_strategy_manager.rb en el método create_replacement_strategies.

- Añadir polítca al instalador install.sh en la variable CACHE_TM_COMMANDS_LIB_FILES_REP

CAPÍTULO 6: Evaluación

El driver “OneCaché” para OpenNebula consiste en una serie de guiones escritos prácticamente de manera íntegra en lenguaje Ruby y en menor medida, bash.

Como hemos visto en capítulos anteriores, OpenNebula ya ofrece diferentes formas de tratar las imágenes en los datastore mediante el Transfer Manager, y con este driver añadimos una posibilidad más orientada a la eficiencia.

Nuestra implementación para el manejo de imágenes destaca por la inclusión en los hosts de una memoria caché donde se guardarán las imágenes que han sido utilizadas, pudiendo ser desplegadas en futuros usos sin necesidad de volver a ser transferidas desde el servidor, y por tanto ahorrando tiempo al usuario final, evitando la sobrecarga en el propio servidor y la sobrecarga en el tráfico de la red.

Para que OpenNebula tenga conocimiento del estado de las diferentes memorias cachés de todos los host de la red, se ha desarrollado un servidor implementado con un modelo de datos en MySQL. Dicho servidor, facilitará información necesaria para la gestión de las memorias cache.

Para el mantenimiento de las memorias cachés se han establecido varias políticas de reemplazamiento las cuales según el uso que vayamos a realizar, podrán ser configuradas mediante su activación, desactivación o dotándolas de diferentes pesos.

Es de vital importancia para el rendimiento final, una exitosa configuración que maximice el número de tasas de acierto en la memoria caché.

6.1 Mejoras proporcionadas por la cache

Como cualquier incorporación de una memoria caché en un sistema de transferencia, lo que se persigue es una reducción de transferencias de ficheros. Este objetivo se conseguirá reduciendo la tasa de fallos en la memoria caché.

Con este objetivo, logramos tres puntos claves para un mejor rendimiento global y satisfacción de los usuarios que mencionamos a continuación:

- Una menor demora en la utilización de una máquina virtual desde que el usuario solicita su despliegue hasta que puede comenzar su uso.
- Una menor sobrecarga de trabajo en el servidor al no ser necesario transferir datos de gran tamaño.
- Una reducción en la sobrecarga de la red debido a la minimización de transferencias.

¿Pero cuál es la configuración óptima para aumentar la tasa de acierto? No existe una fórmula matemática que asegure una máxima reducción de tasa de fallos para todos los usos, ya que dependen de varios factores externos como el tamaño asignado a la memoria caché, tamaño de los archivos a transferir o la localidad de referencia por parte del sistema.

Sin embargo, podemos deducir que para mejorar el rendimiento de la caché, debemos actuar sobre la tasa de fallo, dando lugar a varias optimizaciones:

6.1.1 Reducción de la tasa de fallos.

La tasa de fallos podemos reducirla con las siguientes alternativas:

- Aumento del tamaño de directorio cache.
  Si disponemos de un tamaño de directorio cache mayor, el número de imágenes que pueden ser albergadas crecerá, de tal manera que el número de fallos disminuirá.
  Sin embargo, este método trae consigo una serie de desventajas. Con el aumento del tamaño de bloque, aumentaría también el tiempo de búsqueda en disco por parte del verificador de caché, además, también crecerá la ocupación del tamaño del disco.
  Por todo lo anterior, lo adecuado sería buscar un compromiso entre rendimiento, la tasa de fallos y la ocupación en disco.

- Utilización de una caché de víctimas.
  Se podría implementar una caché intermedia que contuviera los bloques descartados (sustituidos) por un fallo (víctimas).Ante un fallo OpenNebula comprobaría si la imagen había sido anteriormente descartada y por tanto aún se encuentra en la caché de víctimas antes de proceder a copiarla de nuevo a la memoria caché.
  Esta opción podría ser una mejora a añadir a nuestra implementación procediendo así a una caché de dos niveles.

- Asignación/configuración de buenas políticas de reemplazo
  Dependiendo del uso que se le va a dar a la virtualización de las imágenes de cada host, nos interesará asignar una u otra política de reemplazo.
  Por ejemplo si un host tienen pocas posibilidades de utilizar a corto plazo una imagen que ya ha sido desplegada por él, nos convendrá utilizar una política que seleccione como víctimas aquellas imágenes que llevan más tiempo en la caché.

6.1.2 Modelos de trabajo.

- Caso degenerado: Cualquier caso en el que un usuario utilice “n” imágenes, éstas se utilicen de manera consecutiva y el tamaño de la caché en el equipo disponga de un tamaño para n-1 imágenes, en este caso la caché no producirá ganancia, ya que deberá clonar y copiar la imagen cada vez que sea solicitada. En estos casos se producirá un retardo debido a que no sólo clona la imagen, sino que la copia de caché produciendo un retardo extra debido al fallo de caché.

- Si el tamaño de la memoria caché es menor que el de la imagen a descargar, la memoria caché no entrará en acción, por lo que su funcionalidad será anulada y procesará la petición como era el funcionamiento habitual de OpenNebula hasta ahora.

- Trabajo basado en roles: Para un modelo de trabajo basado en roles, la mejor opción es utilizar una política basada en frecuencia de uso.
  Los usuarios suelen tener un trabajo claro y concreto a desempeñar, esto provocará que las máquinas virtuales utilizadas tienden a ser las mismas cada día, por ello se deberán guardar en caché las más frecuentes.
  Es posible que temporalmente desempeñen un trabajo distinto del habitual debido a ausencias puntuales, por ello la máquina que tenga menos usos será eliminada, permaneciendo cuando se reincorpore a su trabajo habitual la máquina que utiliza por su desempeño.

- Por otro lado si un usuario, tiene claro que una vez que utilice una imagen pasará mucho tiempo sin utilizarla (no existe referencia temporal), la inclusión de una memoria caché con la incorporación de una política FIFO (first input first output) proporcionará entre las políticas de reemplazamiento una elección acertada no sólo por su comportamiento sino también por la sencillez y tiempo de cómputo en la actualización de estadísticas.

- Cambios en la reiteración de uso de imágenes: Un usuario puede variar la frecuencia con la cual utiliza distintas imágenes dependiendo de la fase en que se encuentre. Por tanto será necesario la utilización de distintas políticas de reemplazamiento que se fusionarán de distinto modo dependiendo de la importancia que le concedemos, según los pesos otorgados en la configuración inicial.
  La elección de los pesos y las políticas que intervendrán en la gestión de imágenes serán un paso clave para determinar el éxito o no de la utilización de la caché.

6.2 Experimentos

Para tomar las medidas seleccionamos una imagen de tamaño medio para realizar las pruebas. Concretamente el tamaño de la imagen es de 969 MB.

6.2.1 Tiempos globales

Tiempos en desplegar una imagen nueva.

Esta medida será el tiempo necesario en desplegar una imagen a un equipo. El driver tm_ssh desplegará la imagen desde el repositorio de imágenes al host y nuestro driver además creará una entrada en la caché para futuros usos de la imagen.

	TM_CACHE	TM_SSH
TIEMPO	real 3m38.670s user 0m13.153s sys 0m2.244s	real 2m56.436s user 0m0.196s sys 0m0.020s

Tiempos en desplegar una imagen utilizada con anterioridad.

Esta medida será el tiempo necesario en desplegar una imagen a un equipo. El driver tm_ssh, al igual que en el caso anterior desplegará la imagen desde el repositorio de imágenes al host. Por otro lado, nuestro driver simplemente creará una copia de la entrada disponible en la cache.

	TM_CACHE	TM_SSH
TIEMPO	real 2m7.041s user 0m12.873s sys 0m2.380s	real 2m56.436s user 0m0.196s sys 0m0.020s

Como podemos observar, el proceso de primer despliegue de nuestro driver será mínimamente más pesado en tiempo al de otro driver disponible, esto será debido a una primera penalización debido a la creación de la entrada en caché.

Si nos fijamos en los tiempos de la siguiente tabla, podremos deducir que a partir del segundo despliegue la primera penalización se ve compensada. Esta ganancia se observan en las figuras 17 y 18, donde se compara el rendimiento al utilizar la tm_cache frente al tm_ssh.

A continuación mostramos una tabla con los rendimientos en los 5 primeros despliegues teniendo en cuenta los tiempos medidos en un primer y segundo despliegue:

Num Solicitud	TM_CACHE $T_{ejecucion}$ / $T_{acumulado}$	TM_SSH $T_{ejecucion}$ / $T_{acumulado}$	GANANCIA
1	3m 38s / 3m 38s	2 m 56 s / 2 m 56 s	0.757
2	2 m 7 s / 5 m 45 s	2 m 56 s / 5 m 52 s	1.012
3	2 m 7 s / 7 m 52 s	2 m 56 s / 8 m 48 s	1.127
4	2 m 7 s / 9 m 59 s	2 m 56 s / 11 m 44 s	1.192
5	2 m 7 s / 12 m 06 s	2 m 56 s / 14 m 40 s	1.194

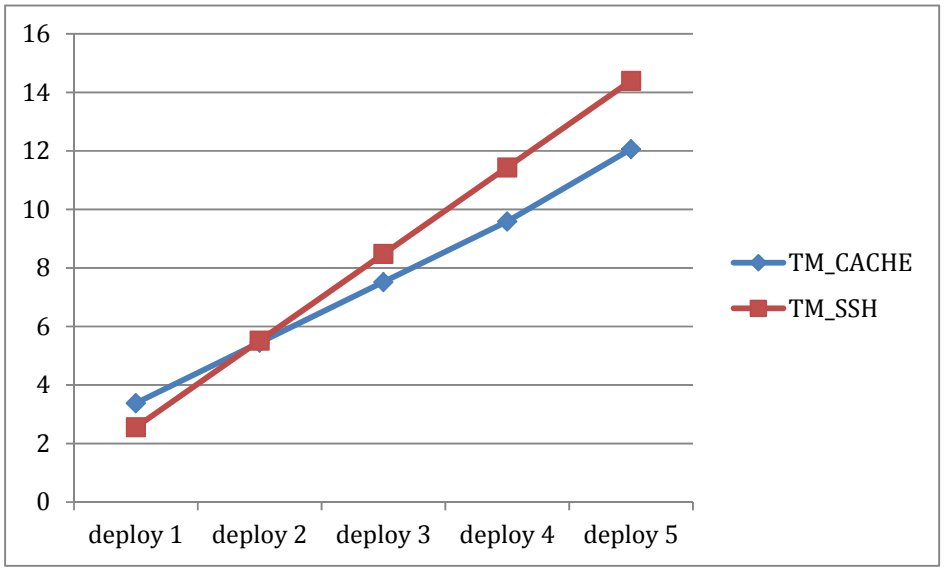


Fig 17. Tiempo de espera de usuario Final

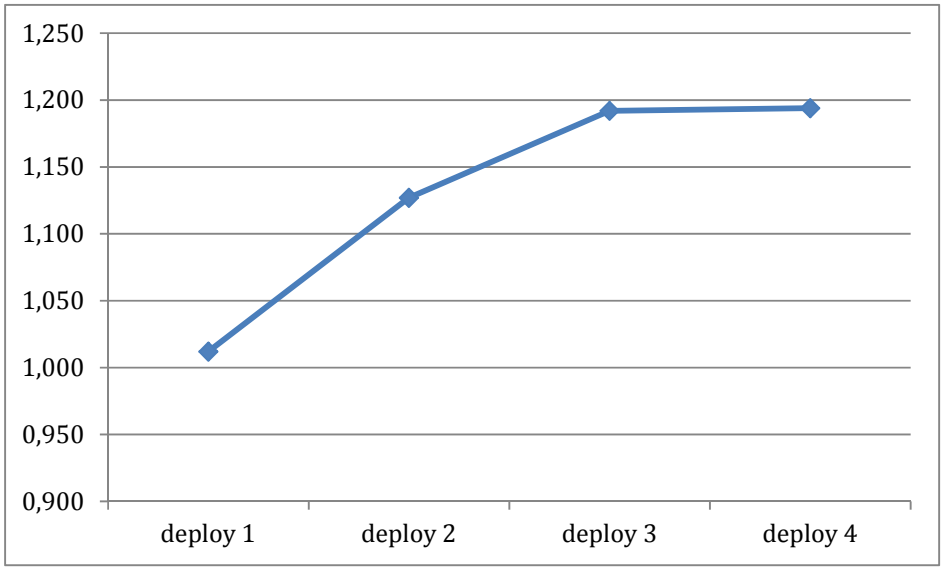


Fig 18. Ganancia de tm_cache frente a tm_ssh

Para poder calcular de forma cuantitativa el beneficio que ofrece el driver tm_cache respecto a otros drives de transferencia debemos poder compararlos con el cálculo de las siguientes variables:

Tiempo de despliegue en la primera solicitud:

Tiempo que transcurre desde la petición de la imagen por parte del host hasta que se puede utilizar dicha imagen desplegada por parte del usuario. Se calcula sumando los siguientes tiempos:

Ttotal = Tcons.bbdd + Tdisco + Tclonado + Tcopiado + Tact bbdd

- Tcons.bbdd: Tiempo dedicado a la consulta de la existencia en la bbdd de caché.
- Tdisco = Consulta de existencia de mv + Creación de estructura de ficheros + Dación de permisos en estructura.
- Tclonado: Tiempo que se tarda en clonar una imagen.
- Tcopiado: Tiempo en que se copia la imagen de la caché a su destino para ser desplegada.
- Tact bbdd: Tiempo en el que se actualizan las políticas y estadísticas de la mv desplegada.

Tiempo de despliegue en las solicitudes sucesivas:

Tiempo que tarda en estar otra vez operativa una imagen que ya había sido utilizada por el host.

Ttotal = Tcons.bbdd + Tdisco [+ Tclonado] + Tcopiado + Tact bbdd

- [+ Tclonado] : Si la máquina se ha eliminado de forma accidental, se contempla la opción de clonarla de nuevo.

- Tdisco = Consulta de existencia de mv + Creación de estructura de ficheros + Dación de permisos en estructura [+ (n * eliminado de mv por políticas)].

  n = número de imágenes necesarias a eliminar para que la nueva imagen se instancie en caché.

CAPÍTULO 7: Conclusiones

La realización de este proyecto nos ha permitido conocer más a fondo algunas tecnologías que desconocíamos hasta el momento, así como poner en práctica algunos de los conocimientos adquiridos durante estos años de estudio.

En primer lugar hemos conocido el gestor OpenNebula, que nos ha servido para comprender en profundidad un campo tan amplio como es el del Cloud Computing, que en estos momentos es fundamental en el mundo de las tecnologías de la información.

Hemos podido comprender la gran importancia que juega una buena estructura en el desarrollo, gestión y evolución de un sistema. En nuestro caso se trata de desarrollar un driver para añadir una nueva funcionalidad al sistema en el manejo de imágenes virtuales.
Para dicho desarrollo, se han tenido en cuenta los requisitos básicos que el driver tenía que cumplir para encajar perfectamente con la estructura de OpenNebula. De esta forma hemos conseguido desarrollar un componente con un bajo nivel de acoplamiento con el gestor, lo cual hace que en cualquier momento se pueda sustituir por otro sin ningún problema.

Dado que el desarrollo se ha realizado mayoritariamente en Ruby, hemos podido conocer un nuevo lenguaje de programación, que tiene un enfoque distinto a lo que hasta ahora habíamos visto. También hemos podido usar el patrón Active Record, que nos ha permitido abstraer la gestión de la base de datos, centrándonos simplemente en el manejo de los datos como objetos.

Además, hemos comprobado el gran avance de la inclusión de una memoria caché en varios ámbitos, entre ellos en el que nos ocupa. Usando una caché en OpenNebula, se ha conseguido disminuir el tiempo de espera, tráfico de red, sobrecargas de trabajo al servidor, etc.

Esta incorporación, en el manejo y gestión de imágenes de un sistema de virtualización como OpenNebula, nos ha proporcionado ganancias en el tiempo de computación pero también nos ha mostrado un abanico de posibilidades para el desarrollo de nuevas mejoras.

Entre las mejoras a desarrollar en este driver, estaría la incorporación de un sistema de transferencia que haga uso del protocolo BitTorrent, que aliviaría aún más el trabajo del servidor. Asimismo se podría incluir un plugin para que el Transfer Manager caché pueda ser gestionado desde la interfaz web (Sunstone), dicho desarrollo aunque no interviniera en mejoras de computación sería un gran avance para simplificar el manejo y gestión de nuestro plugin.

CAPÍTULO 8: Glosario

Back-end: Comprende los componentes que procesan la salida del nodo frontal.

Cloud Computing: modelo que ofrece servicios de computación a través de Internet.

Cluster: Conjunto de máquinas cuyas componentes hardware son comunes a todas ellas y que en su globalidad actúan como si fuesen una única computadora.

Driver: Controlador de un dispositivo, que permite hacer una abstracción del hardware proporcionando una interfaz para hacer uso del recurso.

Host: Computadoras conectadas a una red, que proveen y utilizan servicios de ella.

Hypervisor: También llamado Virtual Machine Manager (VMM), es una de las muchas técnicas de virtualización hardware que permite que múltiples sistemas operativos coexistan de modo concurrente en un único host.

IaaS: Modelo de servicio de cloud que proporciona a los clientes la capacidad de utilizar los recursos de la infraestructura (computacionales, de red y de almacenamiento) como un servicio.

Máquina Virtual: Software que emula a una máquina sobre los cuales se pueden realizar las mismas operaciones que si fuese sobre una máquina real. Los procesos que ejecutan están limitados por los recursos proporcionados por ellas.

MD5: Algoritmo de reducción criptográfico de 128 bits.

MySQL: Sistema de gestión de bases de datos relacional, licenciado bajo la GPL de la GNU. Su diseño multihilo le permite soportar una gran carga de forma muy eficiente. MySQL fue creada por la empresa sueca MySQL AB.

Nodo frontal: La parte de un sistema de software que interactúa directamente con el usuario.

OpenNebula: Plataforma software que permite el desarrollo de cloud públicos, privados y mixtos de infraestructuras como servicio (IaaS). Se trata de un producto software bajo licencia Apache 2.0, desarrollado en la Universidad Complutense de Madrid.

Open Source: Código abierto.

Peers: Todos los usuarios que se encuentran dentro de una red BitTorrent

Repositorio de Imágenes o DataStore: se refiere a cualquier medio de almacenamiento local o remoto, que poseen las imágenes de las máquinas virtuales. Un repositorio de imágenes puede un servidor de archivos dedicado o una URL remota, ambos tienen que ser accesibles desde el nodo frontal de OpenNebula.

Ruby: Lenguaje de programación interpretado, reflexivo y orientado a objetos, creado por Yukihiro Matz. Su implementación oficial es distribuida bajo licencia de software libre.

Seeders: Usuarios de la red que poseen un archivo completo en redes BitTorrent. Comparten partes con los demás peers

SunStone: Interface de OpenNebula que pretende facilitar a los administradores la posibilidad de hacer operaciones sin la necesidad de utilizar el CLI.

Transfer Manager: Módulo de OpenNebula que realiza la gestión de las imágenes de las máquinas virtuales.

CAPÍTULO 9: Bibliografía

[1]. Página web de OpenNebula. http://www.opennebula.org

[2]. CESGA. Instalación y configuración de OpenNebula. Centro de supercomputación CESGA.

[3]. Documentación de API Ruby on Rails http://api.rubyonrails.org/classes/ActiveRecord/Base.html

[4]. Documentación específica de clase Active Record para Ruby http://ar.rubyonrails.org/

[5]. Página web de lenguaje de programación Ruby http://www.ruby-lang.org/en/

[6]. Presentación “Building Clouds with OpenNebula” http://opennebula.org/_media/community:01_-_constantino_vazquez_-_osdc_2012_-_introduction.pdf

[7]. Cloud Computing: Principles, Systems and Applications. Ed. Springer.

[8]. Páginas sobre las tecnologías de Virtualización http://ww.linux-kvm.org/page/Main_Page

[9]. Página de información sobre modelos de servicio Cloud Computing http://www.cloudcomputingla.com/2010/08/saas-paas-e-iaas.html

[10]. Página sobre tecnologías en Pyme http://www.tecnologiapyme.com/software/que-es-la-virtualizacion

Apéndice

Apéndice A: Instalación de OpenNebula

A.1 Prerrequisitos

	Prerrequisito	Descripción
1	SO Anfitrión	debian , openSUSE , Ubuntu instalado en la máquina anfitrión
2	Software OpenNebula	Código fuente o binarios compilados de OpenNebula

A.2 Usuarios y grupos

La instalación del paquete OpenNebula utiliza un nuevo usuario oneadmin en el nodo frontal. Esta cuenta se utilizará para ejecutar los servicios de OpenNebula y hacer la administración regular y tareas de mantenimiento. Eso significa que es necesario primero crear el usuario y loguearse como usuario oneadmin antes de iniciar OpenNebula.

- Crear grupo cloud
  ```
  $ groupadd cloud
  ```
- Crear usuario oneadmin especificando el directorio raíz y su grupo.
  ```
  $ useradd -d /srv/cloud/one -g cloud -m oneadmin
  ```
- Logearse como oneadmin
  ```
  $ sudo su oneadmin
  ```

A.3 Dependencias

OpenNebula contiene un script que instala las dependencias necesarias según la distribución elegida.

Ejecutar como usuario root:
```
$  /usr/share/one/install_gems
```

Dicho script detecta la distribución linux instalada e instala las librerías requeridas. Sin embargo si no encuentra los paquetes necesarios habrá que instalar a mano los paquetes (sqlite3,mysql client,curl, lbxml2, libxslt, ruby >= 1.8.7, gcc, g++, make)
En los host no será necesario instalar ningún componente de OpenNebula. Solo será necesario que tenga ruby >= 1.8.7 , que estén comunicados mediante ssh y que esté configurado correctamente según el hypervisor utilizado.

A.4 Instalación

OpenNebula incluye distribuciones oficiales para los sistemas de debian, openSUSE y Ubuntu sin embargo si queremos compilar e instalar OpenNebula desde el código fuente deberemos realizar los siguientes pasos:

1. Descargar y descomprimir el archivo tar de OpenNebula.
2. Ir a la carpeta creada y ejecutar el comando scons para compilar OpenNebula.
   Como queremos que tenga soporte para mysql ejecutamos el siguiente comando.
   ```
   $ scons sqlite = yes
   ```
3. OpenNebula puede ser instalado de dos modos: system-wide (en el todo el SO) o self-contaided directory (en un directorio autocontenido).
   Se recomienda realizar la instalación autocontenida ya que facilita la gestión de los ficheros al no instalar nada en la raíz del sistema anfitrión.

   Ejecutamos el script de instalación:
   ```
   $ ./install.sh <install_options>
   ```

   Donde <install_options> puede ser uno o más de las siguientes opciones:

OPCION	VALOR
-u	usuario que ejecutará OpenNebula, por defecto el usuario que ejecuta install.sh
-g	grupo que ejecutará OpenNebula, por defecto, el grupo del usuario que lo instala.
-k	mantener los ficheros de configuración de la instalación existente de OpenNebula, útil cuando se actualiza. Este flag no debería estar activo cuando se instala OpenNebula por primera vez.
-d	destino del directorio de instalación. Si se define se especifica la ruta de acceso para una instalación autocontenida. Si no se define será una instalación que se llevará a cabo en todo el sistema.
-r	eliminar OpenNebula, sólo es útil si -d no se ha especificado. Sino solo seria necesario borrar la carpeta que se especificó en -d . rm -rf $ONE_LOCATION
-h	muestra en la instalación.

En nuestro caso haremos la siguiente instalación:

```
$ sudo ./install.sh -u oneadmin -g cloud -d /srv/cloud/one
```

Donde le decimos que instale los archivos poniendo como propietario a oneadmin y como grupo a cloud y que el directorio que va a contener el código fuente de OpenNebula es /srv/cloud/one

A.5 Instalación del Nodo frontal

Después de instalar OpenNebula, se creará la siguiente estructura de directorios:

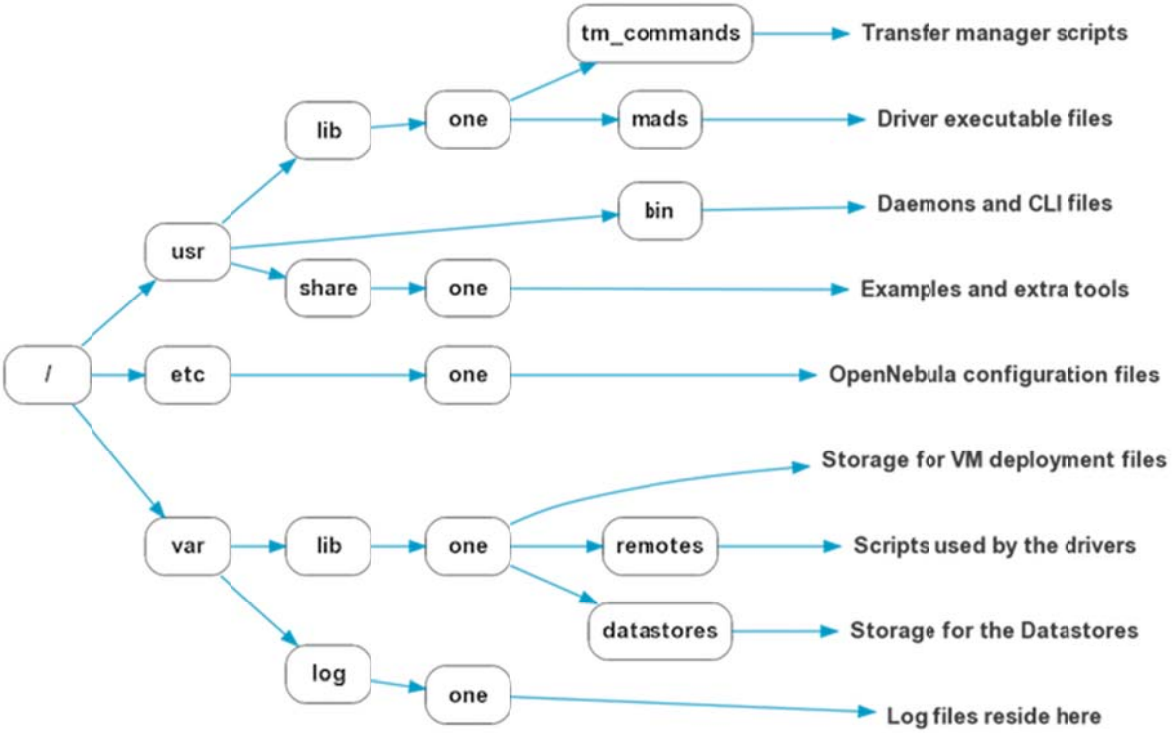


Fig 19. Estructura de directorios de OpenNebula

Como nuestro plugin hace uso de MySQL necesitamos dar permisos al usuario oneadmin para el manejo de tablas en la base de datos de OpenNebula.

Seguiremos los siguientes pasos:

1. Accedemos a MySQL.

   ```
   $ mysql -u root –p
   ```

2. Creamos el usuario oneadmin en MySQL.

   ```
   mysql> GRANT ALL PRIVILEGES ON opennebula.* TO 'oneadmin'IDENTIFIED BY 'oneadmin';
   ```

3. Salimos de MySQL.

   ```
   mysql> exit
   ```

Antes de iniciar OpenNebula por primera vez debemos fijar las credenciales oneadmin en OpenNebula. Hay que realizar los siguientes pasos:

1. Crear carpeta .one

   ```
   $ mkdir .one
   ```

2. Crear archivo de autenticación

   ```
   $ vi .one/one_auth
   ```

3. Establecer usuario y contraseña

   El archivo one_auth debe contener una única línea con el siguiente formato:

   username: password (Ej: oneadmin:oneadmin)

4. Establecer variables de entorno para el usuario oneadmin

   ```
   $ cd ~
   $ vi .basrc
   ```

5. Definir variables de entorno:

   - ONE_LOCATION: Especifica el directorio en el que se encuentra instalado OpenNebula.
   - ONEAUTH: Especifica dónde se encuentra el archivo de autenticación “one_auth”.
   - PATH: Especifica en qué directorio se encuentran las aplicaciones de OpenNebula.

   Ejemplo de archivo .bashrc

   ```
   export ONE_LOCATION=/srv/cloud/one
   export ONE_AUTH=$ONE_LOCATION/one_auth
   export PATH=$PATH:$ONE_LOCATION/bin
   ```

Finalmente instalamos el servidor ssh en el nodo frontal.

Es necesario crear las claves SSH para el usuario oneadmin y configurar los equipos host para que pueda conectarse a ellos mediante ssh sin necesidad de una contraseña.

Ejecutamos las siguientes instrucciones:

Descripción	Comando
Crear par RSA para usuario oneadmin	ssh-keygen -t rsa
Permitir acceso ssh al usuario oneadmin en localhost	cat ~/.ssh/id_rsa.pub >> ~/.ssh/authorized_keys
Crear archivo de configuración del cliente ssh	vi ~/.ssh/config
Establecer la agregación automática de hosts a known_hosts (Archivo ~/.ssh/config)	Host * StrictHostKeyChecking no

Apéndice B: Instalación del driver “OneCache”

B.1 Dependencias

Antes de instalar nuestro plugin, el sistema debe contener los siguientes paquetes:

- gem install activerecord
- gem install composite_primary_keys

Ambos paquetes son necesarios para la utilización de ActiveRecord.

ActiveRecord es un patrón de diseño el cual permite crear un objeto que "envuelve" una tabla SQL, agregándole la lógica del modelo y el control de acceso. Este patrón de diseño permite unir el mundo de la programación orientada a objetos (POO), con el mundo de los datos relacionales (SQL), y por tanto nos permitirá gestionar de una forma más sencilla la operativa de nuestro servidor SQL.

B.2 Instalación

1. Ejecutar el script mysql (install.sql) para generar la estructura de base de datos.
   Para ello nos conectamos como root y ejecutamos el siguiente comando:

   ```
   $ mysql -u “USUARIOBBDD” –password=”PASSBBDD” < ARCHIVOSQL
   ```

   Donde:
   - **USUARIOBBDD:**Usuario para la conexión con la Base de Datos, en este caso oneadmin.
   - **PASSBBDD:**Clave para la conexión con la Base de Datos del usuario oneadmin.
   - **ARCHIVOSQL.sql:** Archivo SQL que será ejecutado (install.sql).

2. Ejecutar el instalador install.sh
   El instalador tiene la posibilidad de realizar enlaces simbólicos o copias, e incluso de desinstalar, al estilo del instalador del propio OpenNebula.

B.3 Configuración

El fichero cache.conf contiene el tamaño del directorio de caché. Por tanto es fundamental modificar el tamaño del directorio, para adaptarlo a las necesidades del usuario. El tamaño definido en el archivo será en Kilobytes.

```
CACHE_SIZE=90000
```

Por otro lado, en el fichero /replacement_strategy/cache_conf.rb, se encuentra la configuración de la gestión de las políticas de reemplazamiento, pudiendo activar/desactivar cada una de las políticas, modificar el peso que toma cada una, etc.

A continuación se muestra un ejemplo de configuración:

```
"numcandidates"            => "5",
"strategyLessUsedactive"   => "Yes",
"strategyLessUsedweight"   => "0.1",
"strategyFIFOactive"       => "Yes",
"strategyFIFOweight"       => "0.1",
"strategyLFUactive"        => "Yes",
"strategyLFUweight"        => "0.5",
"strategyLRUactive"        => "Yes",
"strategyLRUweight"        => "0.3"
```

Apéndice C: Configuración OpenNebula

Para el manejo del plugin “oneCache” necesitamos activar el soporte MySQL en OpenNebula. Antes de ejecutar OpenNebula, es necesario configurar el fichero oned.conf para establecer los detalles de la conexión y la base de datos.

Es necesario efectuar en el archivo de configuración de OpenNebula los cambios que se señalan a continuación:

1. Desactivar el soporte de SQLite, activado por defecto:
   Es necesario comentar la siguiente línea:

   DB = [backend = “sqlite”]

2. Activar soporte MySQL añadiendo:

   DB = [backend = "mysql",
   server = "localhost",
   user = "oneadmin",
   passwd = "oneadmin",
   db_name = "opennebula"]

   Campos:

   - server: indica el servidor de OpenNebula.
   - user: indica el usuario de acceso a la base de datos (usuario MySQL).
   - passwd: indica la contraseña de acceso a la base de datos.
   - db_name: indica el nombre de la base de datos a emplear.

Por otro lado debemos indicar a OpenNebula los drivers que se van a encargar de la gestión de las imágenes en el módulo de Transfer Manager. Es decir debemos indicar que para la gestión de imágenes utilizaremos el driver “oneCache”.
Para ello realizaremos las siguientes modificaciones en el fichero de configuración (oned.conf).

Apéndice D: Configuración de “Transfer Driver”

1. Configurar Transfer Driver en oned.conf

```
#-------------------------------------------------------------------------------
# CACHE Transfer Manager Driver Configuration
#-------------------------------------------------------------------------------
       TM_MAD = [
              name        = "tm_cache",
              executable  = "one_tm",
              arguments   = "tm_cache/tm_cache.conf" ]
#-------------------------------------------------------------------------------
```

   Campos:

   - name: Nombre del transfer driver.
   - executable: Ruta del ejecutable del transfer driver.
   - arguments: Argumentos adicionales para el driver.

D.1 Configuración del instalador de “OneCache”

El fichero de instalación puede realizar enlaces simbólicos o copias de los ficheros necesarios para su funcionamiento con OpenNebula.

Por defecto viene para realizar enlaces simbólicos. Si queremos realizar una copia de los ficheros tendremos que modificar la siguiente variable:

LINK = “no”

Por otro lado para realizar la desinstalación del driver, modificaremos en el mismo fichero la variable UNINSTALL = “yes”

D.2 Iniciar OpenNebula

Una vez configurado correctamente el fichero de configuración de OpenNebula, podemos inciar OpenNebula mediante los siguientes pasos:

1. Inicie sesión como usuario oneadmin.

2. Arranca el demonio de OpenNebula.

   ```
   $ one start
   ```

D.3 Verificar la instalación

Después de que OpenNebula se inicie por primera vez, debemos comprobar que los comandos pueden conectar con OpenNebula. Para ello en el nodo frontal, debemos ejecutar el siguiente comando:

```
$ onevm list
```

Si en lugar de una lista vacía de máquinas virtuales, aparece un error, entonces el demonio de OpenNebula no se pudo iniciar correctamente. Y habrá que comprobar los mensajes de error en los log de OpenNebula.

Apéndice E: Ejemplos de uso

En este apartado mostramos al usuario las herramientas y uso principal del driver “oneCache” con las imágenes explicativas de lo acontecido.

- Demostración de la primera utilización.

  En este apartado, mostraremos al usuario los pasos necesarios para crear un host y desplegar en él una máquina virtual.

  Una vez que tengamos instalado y configurado OpenNebula con el driver “oneCache” iniciaremos los siguientes pasos:

  1. Creación de un host y visualización de sus características.

     Para añadir el host es necesario utilizar la consola de OpenNebula empleando el usuario oneadmin

     onehost create <hostname> <im_mad> <vmm_mad> <tm_mad>

     <hostname> : Identifica al nodo mediante su nombre o dirección IP.

     <im_mad> : Identifica el driver de información a emplear.

     <vmm_mad>: Identifica el driver de virtualización a emplear.

     <tm_mad>: Identifica el driver de transferencia a emplear. En nuestro caso será el desarrollado para el driver “oneCache” : tm_cache

     En nuestro caso crearemos un host KVM con el driver de transferencia tm_cache.

     ```
     $ onehost create 192.168.1.33 im_kvm vmm_kvm tm_cache
     ```

2. Mostraremos la información del host para verificar que se ha creado con el tm especificado.

   onehost show <host_id>

   <host_id> : identificador del host. Se obtiene con el comando onehost list.

   $ onehost show <ID_host>

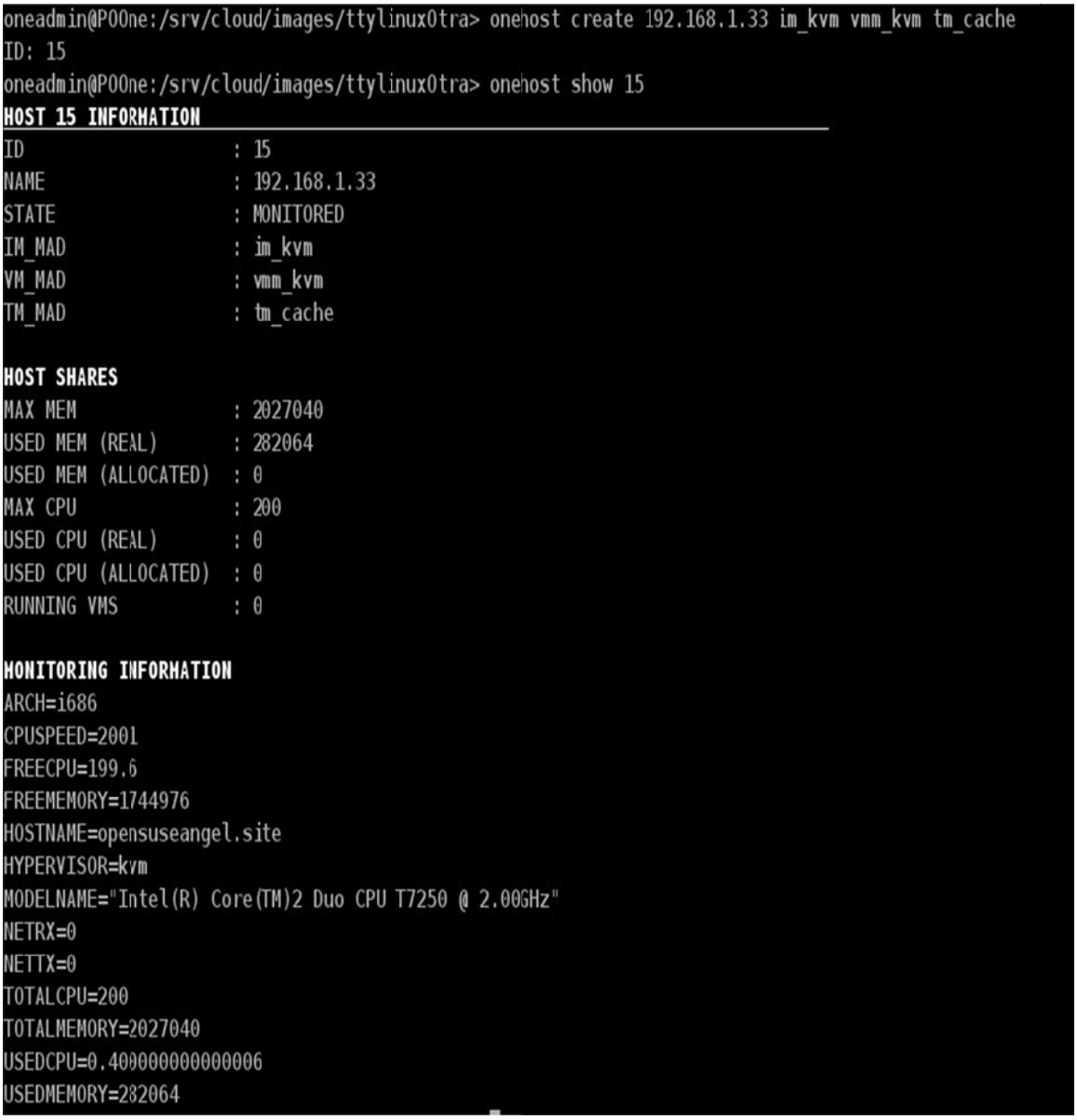


Fig 20. Creación de un host y visualización de sus características.

3. Creación de una máquina virtual.

   onevm create <template>

   <template> : Plantilla que permite desplegar la máquina virtual para una imagen de disco disponible en el repositorio.

   En nuestro caso realizaremos el despliegue de la siguiente imagen:

   ```
   $ onevm create ttylinuxkvmvm.one
   ```

4. Mostramos la información de la máquina virtual.

   onevm show <vm_id>

   <vm_id>: Identificador de la máquina virtual. Se obtiene con el comando onevm list.

   ```
   $ onevm show 171
   ```

   En la Fig. 21 se observa la creación y características de una imagen virtual.

5. Mostramos la listas de máquinas virtuales de forma continua con el siguiente comando:

   onevm top

   Con este comando podremos ver el estado en que se encuentra el despliegue de nuestra máquina virtual.
   Una vez que se haya clonado la imagen a caché y posteriormente se realice un copiado al directorio imagen, la imagen virtual pasará al estado de runn.

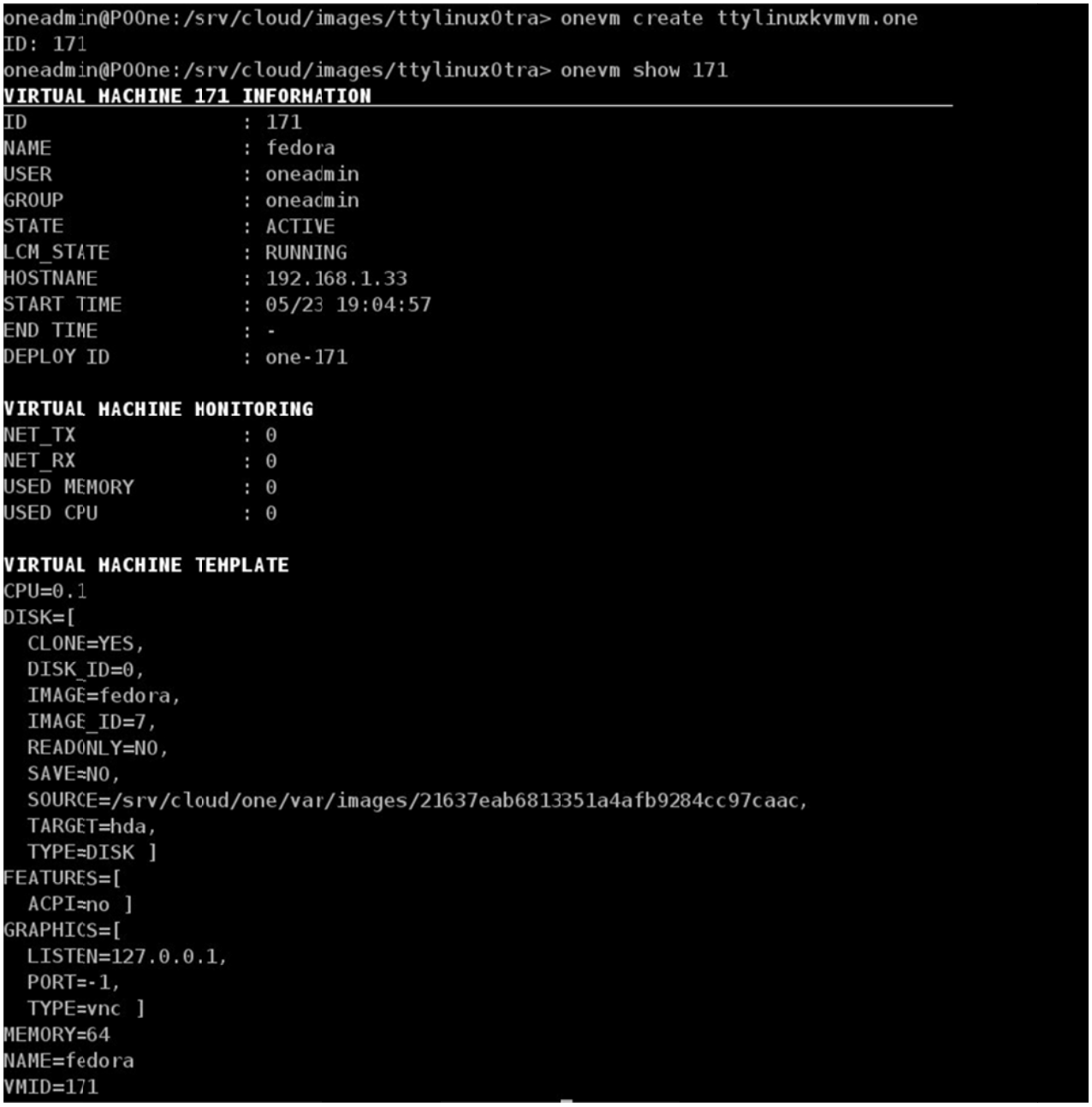


Fig 21. Creación de una máquina virtual y visualización de sus características.

6. Observamos la inicialización de las estadísticas en MySQL con la nueva imagen desplegada. En este caso concreto hemos seleccionado como políticas de reemplazamiento activas FIFO y LFU con distintos pesos para su aplicación.

#	host_name	md5	fifo_create_date
1	192.168.1.33	dab9b4c29da42aceadb5b52f3f2ed756	2012-05-23 19:05:26

lu_number_of_uses	lru_last_time_referenced	lfu_number_of_uses	lfu_age
0	NULL	1	1.000000000000000000000000000000

Fig 22. Estadísticas primer uso.

7. Log del primer despliegue de una imagen. Podemos observar en el log que en primer lugar se registra la imagen mediante su clonado a cache, y posteriormente la imagen es cogida desde caché para su uso.

```
tm_clone.rb: Creating directory /srv/cloud/one/var/cache
tm_clone.rb: Executed "ssh 192.168.1.33 mkdir -p /srv/cloud/one/var/cache".
tm_clone.rb: Creating directory /srv/cloud/one/var//171/images
tm_clone.rb: Executed "ssh 192.168.1.33 mkdir -p /srv/cloud/one/var//171/images".
tm_clone.rb: Max cache size 10000000000
tm_clone.rb: Currently cache size 4
tm_clone.rb: Image size 26696
tm_clone.rb: Check exists: dab9b4c29da42aceadb5b52f3f2ed756 192.168.1.33
tm_clone.rb: FIFO: Initialize statistics correctly for: md5: dab9b4c29da42aceadb5b52f3f2ed
tm_clone.rb: LFU: Initialize statistics correctly for: md5: dab9b4c29da42aceadb5b52f3f2ed7
tm_clone.rb: Initialize stats for md5: dab9b4c29da42aceadb5b52f3f2ed756 in the host: 192.1
tm_clone.rb: New item created: dab9b4c29da42aceadb5b52f3f2ed756 192.168.1.33
tm_clone.rb: Cloning POOne.one:/srv/cloud/one/var/images/21637eab6813351a4afb9284cc97caac
tm_clone.rb: Executed "scp POOne.one:/srv/cloud/one/var/images/21637eab6813351a4afb9284cc9
tm_clone.rb: FIFO: Initialize statistics correctly for: md5: dab9b4c29da42aceadb5b52f3f2ed
tm_clone.rb: LFU: Initialize statistics correctly for: md5: dab9b4c29da42aceadb5b52f3f2ed7
tm_clone.rb: Initialize stats for md5: dab9b4c29da42aceadb5b52f3f2ed756 in the host: 192.1
tm_clone.rb: Copying /srv/cloud/one/var/cache/dab9b4c29da42aceadb5b52f3f2ed756
tm_clone.rb: Executed "ssh 192.168.1.33 cp /srv/cloud/one/var/cache/dab9b4c29da42aceadb5b5
tm_clone.rb: Update statistics: dab9b4c29da42aceadb5b52f3f2ed756 192.168.1.33
tm_clone.rb: Executed "ssh 192.168.1.33 chmod a+rw /srv/cloud/one/var/cache".
tm_clone.rb: Executed "ssh 192.168.1.33 chmod a+rw /srv/cloud/one/var//171/images/disk.0".
```

Fig 23. Fragmento del log en el despliegue de la primera imagen.

- Despliegue de una imagen que ya se encuentra en cache.

  En este segundo caso, la imagen ya se encuentra en cache y por tanto no tendremos que realizar el clonado a la misma ahorrando tiempo en el despliegue y reduciendo de forma considerable el tráfico en la red.

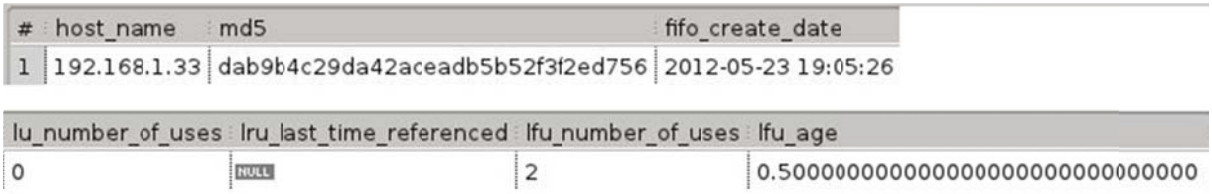
  1. Realizamos los mismos pasos anteriores en la creación de una imagen virtual (la misma que ya se encuentra en cache) y su posterior despliegue.

     $onevm create ttylinuxkvmvm.one

     $onevm top

     Se reducirá de forma considerable el tiempo de despliegue.

  2. Actualización de estadísticas en el segundo uso de una imagen en cache.



#	host_name	md5	fifo_create_date
1	192.168.1.33	dab9b4c29da42aceadb5b52f3f2ed756	2012-05-23 19:05:26

lu_number_of_uses	lru_last_time_referenced	lfu_number_of_uses	lfu_age
0	NULL	2	0.50000000000000000000000000000

Fig 24 Estadísticas segundo uso.

Como observamos en la imagen, ahora el número de usos de lfu_number_of_uses ha aumentado en uno a diferencia de la Fig. 23 y la lfu_age se ha reducido a la mitad.

  3. Log del despliegue de una imagen que ya se encontraba en cache.

```
tm_clone.rb: Creating directory /srv/cloud/one/var/cache
tm_clone.rb: Executed "ssh 192.168.1.33 mkdir -p /srv/cloud/one/var/cache".
tm_clone.rb: Creating directory /srv/cloud/one/var//172/images
tm_clone.rb: Executed "ssh 192.168.1.33 mkdir -p /srv/cloud/one/var//172/images".
tm_clone.rb: Copying /srv/cloud/one/var/cache/dab9b4c29da42aceadb5b52f3f2ed756
tm_clone.rb: Executed "ssh 192.168.1.33 cp /srv/cloud/one/var/cache/dab9b4c29da42aceadb5b5
tm_clone.rb: Update statistics: dab9b4c29da42aceadb5b52f3f2ed756 192.168.1.33
tm_clone.rb: Executed "ssh 192.168.1.33 chmod a+rw /srv/cloud/one/var/cache".
tm_clone.rb: Executed "ssh 192.168.1.33 chmod a+rw /srv/cloud/one/var//172/images/disk.0".
```

Fig 25. Fragmento del log en el segundo despliegue de una imagen en cache.

- Despliegue de una imagen que no se encuentra en cache con el tamaño de cache completo.

  Para este caso de uso vamos desplegar una imagen en un host que nunca ha utilizado dicha imagen. Además se cumple la condición que dispone de una cache usada de manera completa.

  Las imágenes que se encuentran en cache son de al menos la mitad de tamaño de la imagen a desplegar, por lo que será necesario eliminar al menos dos según la política de reemplazo configurada, que en este caso será de FIFO (0.7) y LFU (0.3)

  1. Realizamos los mismos pasos que en los casos anteriores para desplegar la máquina virtual al host. Antes de su despliegue las estadísticas en el sistema se pueden observar en la fig. 26

#	host_name	md5	fifo_create_date	lu_number_of_uses	lru_last_time_referenced	lfu_number_of_uses	lfu_age
1	192.168.1.33	7fc2b85df1c3818cbd9011d95f27d6f7	2012-05-23 12:47:40	0	NULL	3	5.9960818290710449218750
2	192.168.1.33	dab9b4c29da42aceadb5b52f3f2ed756	2012-05-23 12:47:11	0	NULL	9	1.2788388729095458984375
3	192.168.1.33	0aff11ae5bbe3f6e052d2cd3f85a6e77	2012-05-23 12:56:12	0	NULL	3	5.9550781250000000000000
4	192.168.1.33	771520caf85703272e4aab00441b8610	2012-05-23 12:48:42	0	NULL	7	12.466222763061523437500

Fig 26. Estadísticas de uso en cache para el equipo 192.168.1.33

  2. Una vez procedemos a desplegar la imágenes, el gestor deberá eliminar dos imágenes en caché, debido a requisitos de espacio en cache para poder obtener espacio para la nueva imagen a desplegar.

     a. En primer lugar se calculan los valores de probabilidad que tiene cada imagen albergada en cache para ser sustituida.

        En el log mostrado en la fig. 27 podemos observar que para todas las imágenes albergadas en caché o el número configurado en el fichero cache_conf.rb se calcula el valor de la probabilidad normalizada según los pesos configurados.

```
tm_clone.rb: Creating directory /srv/cloud/one/var/cache
tm_clone.rb: Executed "ssh 192.168.1.33 mkdir -p /srv/cloud/one/var/cache".
tm_clone.rb: Creating directory /srv/cloud/one/var//177/images
tm_clone.rb: Executed "ssh 192.168.1.33 mkdir -p /srv/cloud/one/var//177/images".
tm_clone.rb: Max cache size 95000
tm_clone.rb: Currently cache size 90004
tm_clone.rb: Image size 40960
tm_clone.rb: md5: dab9b4c29da42aceadb5b52f3f2ed756  fifo_create_date: Wed May 23 12:47:11 +0200 2012
tm_clone.rb: md5: 7fc2b85df1c3818cbd9011d95f27d6f7  fifo_create_date: Wed May 23 12:47:40 +0200 2012
tm_clone.rb: md5: 771520caf85703272e4aab00441b8610  fifo_create_date: Wed May 23 12:48:42 +0200 2012
tm_clone.rb: md5: 0aff11ae5bbe3f6e052d2cd3f85e6e77  fifo_create_date: Wed May 23 12:56:12 +0200 2012
tm_clone.rb: md5: 771520caf85703272e4aab00441b8610  lfu_number_of_uses: 7 lfu_age: 12.4662227630615234375
tm_clone.rb: md5: 7fc2b85df1c3818cbd9011d95f27d6f7  lfu_number_of_uses: 3 lfu_age: 5.996081829071044921875
tm_clone.rb: md5: 0aff11ae5bbe3f6e052d2cd3f85e6e77  lfu_number_of_uses: 3 lfu_age: 5.955078125
tm_clone.rb: md5: dab9b4c29da42aceadb5b52f3f2ed756  lfu_number_of_uses: 9 lfu_age: 1.2788388729095458984375
tm_clone.rb: key: 7fc2b85df1c3818cbd9011d95f27d6f7
tm_clone.rb: value: 0.9
tm_clone.rb: ----------------------
tm_clone.rb: key: 771520caf85703272e4aab00441b8610
tm_clone.rb: value: 0.94
tm_clone.rb: ----------------------
tm_clone.rb: key: dab9b4c29da42aceadb5b52f3f2ed756
tm_clone.rb: value: 0.79
tm_clone.rb: ----------------------
tm_clone.rb: key: 0aff11ae5bbe3f6e052d2cd3f85e6e77
tm_clone.rb: value: 0.77
tm_clone.rb: ----------------------
tm_clone.rb: Deleting because it applies to replacement policy /srv/cloud/one/var/cache/771520caf85703272e4aab00441b8610
tm_clone.rb: Executed "ssh 192.168.1.33 rm /srv/cloud/one/var/cache/771520caf85703272e4aab00441b8610".
tm_clone.rb: Delete item: 771520caf85703272e4aab00441b8610 192.168.1.33
```

Fig 27. Log de sustitución y cálculo de pesos para primer reemplazo de imágenes

b. Posteriormente se vuelve a verificar si ya se puede registrar la imagen en cache o es necesario seguir aplicando política de reemplazo. Para el caso de uso demostrado es necesario una segunda aplicación de la política de reemplazo para desplegar la máquina virtual como queda visible en la fig. 28

```
tm_clone.rb: Max cache size 95000
tm_clone.rb: Currently cache size 68896
tm_clone.rb: Image size 40960
tm_clone.rb: md5: dab9b4c29da42aceadb5b52f3f2ed756  fifo_create_date: Wed May 23 12:47:11 +0200 2012
tm_clone.rb: md5: 7fc2b85df1c3818cbd9011d95f27d6f7  fifo_create_date: Wed May 23 12:47:40 +0200 2012
tm_clone.rb: md5: 0aff11ae5bbe3f6e052d2cd3f85e6e77  fifo_create_date: Wed May 23 12:56:12 +0200 2012
tm_clone.rb: md5: 7fc2b85df1c3818cbd9011d95f27d6f7  lfu_number_of_uses: 3 lfu_age: 5.996081829071044921875
tm_clone.rb: md5: 0aff11ae5bbe3f6e052d2cd3f85e6e77  lfu_number_of_uses: 3 lfu_age: 5.955078125
tm_clone.rb: md5: dab9b4c29da42aceadb5b52f3f2ed756  lfu_number_of_uses: 9 lfu_age: 1.2788388729095458984375
tm_clone.rb: key: 7fc2b85df1c3818cbd9011d95f27d6f7
tm_clone.rb: value: 0.97
tm_clone.rb: ----------------------
tm_clone.rb: key: dab9b4c29da42aceadb5b52f3f2ed756
tm_clone.rb: value: 0.86
tm_clone.rb: ----------------------
tm_clone.rb: key: 0aff11ae5bbe3f6e052d2cd3f85e6e77
tm_clone.rb: value: 0.87
tm_clone.rb: ----------------------
tm_clone.rb: Deleting because it applies to replacement policy /srv/cloud/one/var/cache/7fc2b85df1c3818cbd9011d95f27d6f7
tm_clone.rb: Executed "ssh 192.168.1.33 rm /srv/cloud/one/var/cache/7fc2b85df1c3818cbd9011d95f27d6f7".
tm_clone.rb: Delete item: 7fc2b85df1c3818cbd9011d95f27d6f7 192.168.1.33
```

Fig 28. Log de sustitución y cálculo de pesos para segundo reemplazo de imágenes

c. Por último, una vez verificado que existe espacio en cache se procede a clonar la imagen como si se tratara del primer despliegue de la misma. La verificación de que los pasos se han realizado correctamente se pueden observar en la fig 29

```
tm_clone.rb: Max cache size 95000
tm_clone.rb: Currently cache size 47792
tm_clone.rb: Image size 40960
tm_clone.rb: Check exists: 04c7d00e88fa66d9aaa34d9cf8ad6aaa 192.168.1.33
tm_clone.rb: FIFO: Initialize statistics correctly for: md5: 04c7d00e88fa66d9aaa34d9cf8ad6aaa
tm_clone.rb: LFU: Initialize statistics correctly for: md5: 04c7d00e88fa66d9aaa34d9cf8ad6aaa
tm_clone.rb: Initialize stats for md5: 04c7d00e88fa66d9aaa34d9cf8ad6aaa in the host: 192.168.1.33 correctly
tm_clone.rb: New item created: 04c7d00e88fa66d9aaa34d9cf8ad6aaa 192.168.1.33
tm_clone.rb: Cloning POOne.one:/srv/cloud/one/var/images/f90748b6bd9ad4e1f98268b4bfc2693f
tm_clone.rb: Executed "scp POOne.one:/srv/cloud/one/var/images/f90748b6bd9ad4e1f98268b4bfc2693f 192.168.1.33:/srv/cloud/one/var/cache/04c7d00e88fa66d9aaa34d9cf8ad6aaa".
tm_clone.rb: FIFO: Initialize statistics correctly for: md5: 04c7d00e88fa66d9aaa34d9cf8ad6aaa
tm_clone.rb: LFU: Initialize statistics correctly for: md5: 04c7d00e88fa66d9aaa34d9cf8ad6aaa
tm_clone.rb: Initialize stats for md5: 04c7d00e88fa66d9aaa34d9cf8ad6aaa in the host: 192.168.1.33 correctly
tm_clone.rb: Copying /srv/cloud/one/var/cache/04c7d00e88fa66d9aaa34d9cf8ad6aaa
tm_clone.rb: Executed "ssh 192.168.1.33 cp /srv/cloud/one/var/cache/04c7d00e88fa66d9aaa34d9cf8ad6aaa /srv/cloud/one/var//177/images/disk.0".
tm_clone.rb: Update statistics: 04c7d00e88fa66d9aaa34d9cf8ad6aaa 192.168.1.33
tm_clone.rb: Executed "ssh 192.168.1.33 chmod a+rw /srv/cloud/one/var/cache".
tm_clone.rb: Executed "ssh 192.168.1.33 chmod a+rw /srv/cloud/one/var//177/images/disk.0".
```

Fig 29. Log de clonado de imagen a cache

3. Los pasos finales de este caso de uso será la actualización de las estadísticas en la base de datos, realizando la eliminación de las imágenes que han sido eliminadas de cache y registrando la nueva imagen que se desea copiar como se puede observar en la fig 30

#	host_name	md5	fifo_create_date	lu_number_of_uses	lru_last_time_referenced	lfu_number_of_uses	lfu_age
1	192.168.1.33	04c7d00e88fa66d9aaa34d9cf8ad6aaa	2012-05-23 19:33:30	0	NULL	1	1.000000000000000000000000
2	192.168.1.33	0aff11ae5bbe3f6e052d2cd3f85e6e77	2012-05-23 12:56:12	0	NULL	3	5.977539062500000000000000
3	192.168.1.33	dab9b4c29da42aceadb5b52f3f2ed756	2012-05-23 12:47:11	0	NULL	9	9.639419436454772949218750

Fig 30. Estadísticas de imágenes en equipo tras sustitución

- Prueba de robustez.

1. Imagen md5 corrupta en caché.

   Cabe la posibilidad que la imagen que se desea desplegar ya exista en la cache pero por algún motivo dicha imagen se encuentra corrupta siendo diferente el md5 de la imagen en cache y el de la imagen que se quiere desplegar.
   El plugin de “OneCache” detectará que dicha imagen no es válida y procederá a su borrado para liberar espacio en cache y posteriormente volverá a clonarla para su correcto funcionamiento.

   En el siguiente log se muestra dicha operativa:

```
tm_clone.rb: Creating directory /srv/cloud/one/var/cache
tm_clone.rb: Executed "ssh 192.168.1.33 mkdir -p /srv/cloud/one/var/cache".
tm_clone.rb: Creating directory /srv/cloud/one/var//179/images
tm_clone.rb: Executed "ssh 192.168.1.33 mkdir -p /srv/cloud/one/var//179/images".
tm_clone.rb: Deleting because md5 is different /srv/cloud/one/var/cache/04c7d00e88fa66d9aaa34d9cf8ad6aaa
tm_clone.rb: Executed "ssh 192.168.1.33 rm /srv/cloud/one/var/cache/04c7d00e88fa66d9aaa34d9cf8ad6aaa'.
tm_clone.rb: Max cache size 95000
tm_clone.rb: Currently cache size 4
tm_clone.rb: Image size 40960
tm_clone.rb: Check exists: 04c7d00e88fa66d9aaa34d9cf8ad6aaa 192.168.1.33
tm_clone.rb: Cloning POOne.one:/srv/cloud/one/var/images/f90748b6bd9ad4e1f98268b4bfc2693f
tm_clone.rb: Executed "scp POOne.one:/srv/cloud/one/var/images/f90748b6bd9ad4e1f98268b4bfc2693f 192.168.1
tm_clone.rb: FIFO: Initialize statistics correctly for: md5: 04c7d00e88fa66d9aaa34d9cf8ad6aaa
tm_clone.rb: LFU: Initialize statistics correctly for: md5: 04c7d00e88fa66d9aaa34d9cf8ad6aaa
tm_clone.rb: Initialize stats for md5: 04c7d00e88fa66d9aaa34d9cf8ad6aaa in the host: 192.168.1.33 correct
tm_clone.rb: Copying /srv/cloud/one/var/cache/04c7d00e88fa66d9aaa34d9cf8ad6aaa
tm_clone.rb: Executed "ssh 192.168.1.33 cp /srv/cloud/one/var/cache/04c7d00e88fa66d9aaa34d9cf8ad6aaa /srv
tm_clone.rb: Update statistics: 04c7d00e88fa66d9aaa34d9cf8ad6aaa 192.168.1.33
tm_clone.rb: Executed "ssh 192.168.1.33 chmod a+rw /srv/cloud/one/var/cache".
tm_clone.rb: Executed "ssh 192.168.1.33 chmod a+rw /srv/cloud/one/var//179/images/disk.0".
```

Fig 31. Fragmento del log md5 corrupto.

Como se puede observar el log informa del borrado del fichero encontrado en cache debido a que el fichero se encuentra corrupto detectado gracias a la comprobación del md5.

2. No se encuentra la imagen en caché.

   Puede que aunque el servidor de estadísticas MySQL indique que existe en un host X una imagen Y en cache, en realidad dicha imagen no se encuentre en la caché.

   Por tanto una vez que indiquemos a OpenNebula que queremos desplegar dicha imagen Y en el host X, comprobará si realmente se encuentra en cache y detectará que no es así, por lo que se realizará de nuevo el clonado de la imagen como si se tratara de una primera solicitud.

   El siguiente log muestra dicho caso:

```
tm_clone.rb: Creating directory /srv/cloud/one/var/cache
tm_clone.rb: Executed "ssh 192.168.1.33 mkdir -p /srv/cloud/one/var/cache".
tm_clone.rb: Creating directory /srv/cloud/one/var//180/images
tm_clone.rb: Executed "ssh 192.168.1.33 mkdir -p /srv/cloud/one/var//180/images".
tm_clone.rb: Max cache size 95000
tm_clone.rb: Currently cache size 4
tm_clone.rb: Image size 40960
tm_clone.rb: Check exists: 04c7d00e88fa66d9aaa34d9cf8ad6aaa 192.168.1.33
tm_clone.rb: Cloning POOne.one:/srv/cloud/one/var/images/f90748b6bd9ad4e1f98268b4bfc2693f
tm_clone.rb: Executed "scp POOne.one:/srv/cloud/one/var/images/f90748b6bd9ad4e1f98268b4bfc2693f 192.168.1.33:
tm_clone.rb: FIFO: Initialize statistics correctly for: md5: 04c7d00e88fa66d9aaa34d9cf8ad6aaa
tm_clone.rb: LFU: Initialize statistics correctly for: md5: 04c7d00e88fa66d9aaa34d9cf8ad6aaa
tm_clone.rb: Initialize stats for md5: 04c7d00e88fa66d9aaa34d9cf8ad6aaa in the host: 192.168.1.33 correctly
tm_clone.rb: Copying /srv/cloud/one/var/cache/04c7d00e88fa66d9aaa34d9cf8ad6aaa
tm_clone.rb: Executed "ssh 192.168.1.33 cp /srv/cloud/one/var/cache/04c7d00e88fa66d9aaa34d9cf8ad6aaa /srv/clo
tm_clone.rb: Update statistics: 04c7d00e88fa66d9aaa34d9cf8ad6aaa 192.168.1.33
tm_clone.rb: Executed "ssh 192.168.1.33 chmod a+rw /srv/cloud/one/var/cache".
tm_clone.rb: Executed "ssh 192.168.1.33 chmod a+rw /srv/cloud/one/var//180/images/disk.0".
```

Fig 32. Fragmento del log md5 inexistente.

Apéndice F: Pruebas realizadas

CASO 1: El registro existe en la bbdd de cache

CASO 1.1 El fichero existe en el destino cache

CASO 1.1.1 El fichero tiene igual MD5 que la imagen a copiar

☑Comprobar que se crea la estructura de fichero de máquina virtual con el fichero de imagen copia de cache.

☑Comprobar que se actualizan estadísticas en la bbdd.

CASO 1.1.2 El fichero tiene distinto MD5 que la imagen a copiar: Para comprobarlo, se creará un fichero en la caché con el nombre del MD5 de la imagen origen y sin contenido.

☑Comprobar que se elimina fichero de caché incorrecto.

☑Comprobar que se clona fichero de imagen correcto.

☑Comprobar que se crea la estructura de fichero de máquina virtual con el fichero de imagen copia de cache.

☑Comprobar que se inicializan estadísticas en la bbdd.

CASO 1.2 El fichero no existe en el destino cache: Una vez registrada una imagen a un host en la bbdd se eliminará el fichero de la cache simulando su pérdida o su eliminado manual.

CASO 1.2.1 El valor máximo de la variable CACHE_SIZE es menor que el tamaño de la imagen: Definir un tamaño de CACHE_SIZE menor que el tamaño de la imagen a copiar

☑Comprobar que el log muestra la información relativa a esta situación.

☑Comprobar que no realiza la copia en caché pero permite desplegar la máquina

CASO 1.2.2 El valor máximo de la variable CACHE_SIZE es mayor que el tamaño de la imagen

CASO 1.2.2.1 El valor de la variable CACHE_SIZE es mayor que el tamaño actual del directorio + Tamaño de imagen a copiar

☑Comprobar que se clona fichero de imagen correcto.

☑Comprobar que se crea la estructura de fichero de máquina virtual con el fichero de imagen copia de cache.

☑Comprobar que se inicializan estadísticas en la bbdd.

☑Comprobar que se las anteriores pruebas se cumplen con al menos 2 máquinas.

CASO 1.2.2.2 El valor de la variable CACHE_SIZE es menor que el tamaño actual del directorio + Tamaño de imagen a copiar

☑Comprobar que se elimina de bbdd el registro correcto según la política de reemplazo.

☑Comprobar que se elimina el fichero de caché correcto según la política de reemplazo

☑Comprobar que se clona fichero de imagen correcto.

☑Comprobar que se crea la estructura de fichero de máquina virtual con el fichero de imagen copia de cache.

☑Comprobar que se inicializan estadísticas en la bbdd.

☑Comprobar que el juego de pruebas anterior se cumple cuando se debe eliminar al menos dos imágenes.

CASO 2: El registro no existe en la bbdd cache

CASO 2.1 El fichero existe en el destino cache: Para comprobar esto se copiará la imagen a cache sin haber sido registrada previamente en la bbdd.

☑Comprobar que se crea el registro en bbdd.

Se ejecutará el juego de pruebas del caso 1.1

CASO 2.2 El fichero no existe en el destino cache: Este es el funcionamiento normal

☑Comprobar que se crea el registro en bbdd.

Se ejecutará el juego de pruebas del caso 1.2

Apéndice G: Posible extensión del proyecto.

Siguiendo con la dinámica de poder descargar al servidor en sus tareas y que la espera del usuario sea menor, como extensión a este proyecto cabe la posibilidad de realizar transferencias mediante protocolo de red BitTorrent diseñado para el intercambio de archivos peer-to-peer.

Este protocolo utiliza un servidor dedicado llamado tracker que contiene la información necesaria para que los peers se conecten unos con otros, pero como nuestra intención es no sobrecargar el servidor, se debería utilizar una mejora al protocolo.

La principal mejora del protocolo importante para nuestro driver se basaría en el uso de sistemas trackerless que se basa de un sistema que permite intercambiar entre clientes de BitTorrent ficheros sin necesidad de disponer de un servidor tracker dedicado.

Para este uso los clientes necesitarían tener activada la opción DHT (Distributed Hash Table) dentro de los programas de intercambio BitTorrent.

Uno de los programas compatibles con la opción DHT indicada y con la mayoría de distribuciones Linux es la última versión de Transmission.