Un Modelo de Arquitectura para un Sistema de Virtualización Distribuido

Autor: Pablo Andrés Pessolani
Directores:
Dr. Toni Cortés – UPC – BSC- Barcelona – España.
Dr. Fernando G. Tinetti – UNLP, CIC Prov. de Bs. As. – La Plata – Argentina
Co-Director:
Dr. Silvio Gonnet – INGAR (CONICET - UTN FRSF) – Santa Fe - Argentina.

Resumen de Tesis presentada para obtener el grado de Doctor en Ciencias Informáticas

Universidad Nacional de La Plata – Facultad de Informática

Defensa de Tesis: 21 de Noviembre de 2018 – La Plata
Video Online: https://www.youtube.com/watch?v=Ha9Cx4sWimk
Calificación obtenida: 10 (diez).

Palabras claves: Virtualización, Contenedores, Sistemas Distribuidos, IaaS.

RESUMEN

Si bien los Sistemas Operativos disponen de características de seguridad, protección, gestión de recursos, etc., éstas parecen ser insuficientes para satisfacer los requerimientos de los sistemas informáticos que suelen estar permanente y globalmente conectados. Las actuales tecnologías de virtualización han sido, y continúan siendo masivamente adoptadas para cubrir esas necesidades de sistemas y aplicaciones por sus características de particionado de recursos, aislamiento, capacidad de consolidación, seguridad, soporte de aplicaciones heredadas, facilidades de administración, etc. Una de sus restricciones es que el poder de cómputo de una Máquina Virtual (o un Contenedor) está acotado al poder de cómputo de la máquina física que la contiene. Esta tesis propone superar esta restricción abordando esta problemática con el enfoque de un sistema distribuido.

Para poder alcanzar mayores niveles de rendimiento y escalabilidad, los programadores de aplicaciones nativas para la Nube deben partirlas (factorizarlas) en diferentes componentes distribuyendo su ejecución en varias Máquinas Virtuales (o Contenedores). Dichos componentes se comunican mediante interfaces bien definidas tales como las interfaces de Web Services. Las Máquinas Virtuales (o Contenedores) deben configurarse, asegurarse y desplegarse para poder ejecutar la aplicación. Esto se debe, en parte, a que los diferentes componentes no comparten la misma instancia de Sistema Operativo por lo que no comparten los mismos recursos abstractos tales como colas de mensajes, mutexes, archivos, pipes, etc. El defecto de esta modalidad de desarrollo de aplicaciones es que impide una visión integral y generalizada de los recursos. En ella, el programador debe planificar la asignación de recursos a cada componente de su aplicación y, por lo tanto, no solo debe programar su aplicación sino también gestionar la distribución de esos recursos.

En este trabajo se propone un modelo de arquitectura para un Sistema de Virtualización Distribuido (DVS) que permite expandir los límites de un dominio de ejecución más allá de una máquina física, explotando el poder de cómputo de un cluster de computadores. En un DVS se combinan e integran tecnologías de Virtualización, de Sistemas Operativos y de Sistemas Distribuidos, donde cada una de ellas le aporta sus mejores características. Este modelo de arquitectura, por ejemplo, le brinda al programador una visión integrada de los recursos distribuidos

que dispone para su aplicación relevándolo de la responsabilidad de gestionarlos. El modelo de DVS propuesto dispone de aquellas características que son requeridas por los proveedores de servicios de infraestructura en la Nube, tales como: mayor rendimiento, disponibilidad, escalabilidad, elasticidad, capacidad de replicación y migración de procesos, balanceo de carga, entre otras.

Las aplicaciones heredadas pueden migrarse más fácilmente, dado que es posible disponer de la misma instancia de un Sistema Operativo Virtual en cada nodo del cluster de virtualización. Las aplicaciones desarrolladas bajo las nuevas metodologías para el diseño y desarrollo de software para la Nube también se benefician adaptándose su utilización a un sistema que es inherentemente distribuido.

INTRODUCCIÓN

Las actuales tecnologías de virtualización han sido y son masivamente adoptadas para satisfacer aquellos requerimientos en que los Sistemas Operativos (OS: Operating System) han presentado debilidades, tales como el aislamiento de fallos, de seguridad y de rendimiento. Además, la virtualización incorpora nuevas facilidades tales como el particionado de recursos, la posibilidad de realizar la consolidación de servidores, el soporte para la ejecución de aplicaciones legadas, la disponibilidad de herramientas que facilitan su gestión, etc. Todas estas características resultan muy atractivas para las organizaciones en general y para los proveedores de Servicios en la Nube (Cloud Services) en particular.

Actualmente, existen varias tecnologías de virtualización que permiten brindar Infraestructura como Servicio (IaaS: Infrastructure as a Service) desplegadas sobre un cluster de servidores enlazados por redes de alta velocidad. Las redes de almacenamiento (SAN: Storage Area Networks), los dispositivos de seguridad (firewalls de red, firewalls de aplicación, Sistemas de Detección y Prevención de Intrusos, etc.), los dispositivos de red (switches, routers, balanceadores de carga, etc.), y un conjunto de sistemas de gestión y monitoreo complementan la infraestructura (informática) requerida para brindar este tipo de servicios a terceros en Nubes Públicas, o en la propia compañía (on-premise) para el caso de Nubes Privadas.

Para establecer los límites de las responsabilidades entre el proveedor de Servicios en la Nube y el cliente, parece adecuado adoptar la definición del NIST respecto a Infraestructura como Servicio (IaaS):

> *La capacidad de proveer al consumidor procesamiento, almacenamiento, redes y otros recursos informáticos fundamentales sobre los cuales el consumidor puede implementar y ejecutar software arbitrario, pudiendo incluir sistemas operativos y aplicaciones. El consumidor no administra ni controla la infraestructura subyacente de la Nube, pero tiene control sobre los sistemas operativos, el almacenamiento y otras aplicaciones implementadas; y posiblemente un control limitado de los componentes de red seleccionados (por ejemplo, firewalls) [1].*

Las tecnologías de virtualización más utilizadas son *Virtualización de Hardware*, *Paravirtualización* y *Virtualización basada en Sistema Operativo*. Cada una de ellas presenta diferentes características en cuanto a niveles de consolidación, rendimiento, escalabilidad, disponibilidad y aislamiento. Una *Máquina Virtual* (VM: Virtual Machine) es un entorno de ejecución aislado creado por el software encargado de gestionar todos los recursos del computador al que se lo conoce como *Hipervisor* o *Monitor de Máquina Virtual* (VMM: Virtual Machine Monitor). Generalmente, el término VM se relaciona a las tecnologías de Virtualización de Hardware y de Paravirtualización, aunque también es utilizado en la virtualización de procesos tales como la *Máquina Virtual Java* (JVM: Java Virtual Machine). Los entornos de ejecución aislados creados a nivel de OS se denominan *Contenedores, Prisiones* o *Zonas* (según el OS). A los efectos

de utilizar una única denominación en el resto del resumen, a estos entornos o dominios de ejecución se los referirá como *Contenedores*.

Sin considerar cuál es la denominación del *entorno de ejecución* creado por la virtualización según su tipo, queda claro que éste resulta del particionado de los recursos de un computador. Como consecuencia de esto, *el poder de cómputo y la utilización de recursos de esos entornos estarán acotados por el computador que lo contiene*. Al menos dos preguntas surgen al considerar esta limitación:

1. ***¿Cómo se pueden expandir el poder de cómputo y la utilización de recursos de una VM o Contenedor a varios computadores?***

   Se pretende disponer de un entorno de ejecución aislado equivalente a una VM o Contenedor, pero que su alcance no se limite a un único computador.

2. ***¿Cómo lograr mayores niveles de rendimiento y escalabilidad de las aplicaciones que se ejecutan en la Nube?***

   Las aplicaciones que ejecutan en la Nube demandan cada vez más potencia de cómputo y recursos (memoria, almacenamiento, red, etc.). El rendimiento y la escalabilidad requerida son imposibles de lograr con un único computador. Además, dada la criticidad de las aplicaciones, existen otros requerimientos que deben cumplirse de tal modo que se les permita a éstas expandirse o contraerse según la demanda (elasticidad), tolerar fallos, ahorrar energía y facilitar su despliegue y gestión.

Los diseñadores de aplicaciones para la Nube utilizan metodologías de desarrollo de software basada en la Arquitectura de Micro-Servicios (MSA: Micro Service Architecture) [2] o Arquitectura Orientada al Servicio (SOA: Service Oriented Architecture) [3]. Ambas arquitecturas proponen dividir las aplicaciones en múltiples servicios desplegados en múltiples VMs o Contenedores. Para realizar la configuración y el despliegue de los diferentes componentes se utilizan gestores de Contenedores tales como Docker [4], Mesos [5] o Kubernetes [6]. En el caso de Mesos, éste se anuncia como un DC/OS (Data Center Operating System), pero ésto resulta confuso ya que en realidad no es un OS, al menos en lo que a la definición tradicional de un OS refiere. Mesos es el software que permite gestionar todos los recursos de un cluster, tratando de integrar la aplicación a esos recursos.

Por otro lado, cuando dos procesos componentes de la aplicación deben comunicarse, si los mismos se encuentran ejecutando en la misma VM o Contenedor (bajo el control del mismo OS) se pueden comunicar utilizando pipes o tuberías, memoria compartida, colas de mensajes, señales, semáforos, etc. Pero, cuando se comunican con procesos que se encuentran en otras VMs o Contenedores deben utilizar protocolos de red, sean éstos de alto nivel como HTTP, de nivel medio como RPC, o de bajo nivel como sockets TCP o UDP.

Si bien existen numerosas herramientas y una multiplicidad de software para desarrollar aplicaciones para la Nube, no conforman una solución transparente, consistente y homogénea como en un sistema centralizado. La complejidad recae entonces en el usuario, programador o administrador, el cual debe hacerse cargo de aspectos relativos al sistema y a los recursos distribuidos [7]. Esto se debe, a que todo el conjunto de recursos dispersos (de cómputo, de almacenamiento y de red) presentan una visión no homogénea que el usuario debe integrar, gestionar y mantener. Esto significa que, desde el punto de vista netamente práctico, se requiere realizar tareas tales como: asignar direcciones IP, puertos TCP/UDP, configurar reglas en los firewalls, establecer servidores de autenticación, mantener versiones de OS, mantener actualizadas bibliotecas y paquetes de software, y mantener archivos de configuración sincronizados entre VMs/Contenedores.

Cuando se utiliza IaaS para desplegar las aplicaciones de esta manera, no se puede asegurar que los Contenedores o las VMs contratadas se localicen en nodos diferentes con lo cual, muchos

supuestos realizados para distribuir cargas o para tolerar fallos a través de la redundancia dejan de tener sustento.

La tesis propone un modelo de arquitectura que se basa en este enfoque distribuyendo procesos, servicios y recursos para proveer un entorno de ejecución aislado basado en múltiples *Contenedores Distribuidos* (DC: Distributed Containers) y en la factorización de los OSs. El resultado es un Sistema de Virtualización Distribuido que combina tecnologías de OS y de virtualización de tal forma de que las aplicaciones puedan acceder a los recursos de su OS sin importar en el host donde se éstas se ejecutan. Esta característica simplifica el desarrollo de aplicaciones (costo y tiempo).

MOTIVACIÓN

Conforme una VM o Contenedor resulta de una partición de recursos de un computador, su potencia de cómputos y disponibilidad de recursos están limitadas al computador que lo contiene (las partes no pueden ser mayores que el todo). Esto nos permite realizar la siguiente afirmación:

En los sistemas de virtualización de la actualidad, los recursos y el poder de cómputo de una VM o de un Contenedor se encuentran limitados al computador donde residen.

En base a esta limitante, el modelo de arquitectura propuesto por la tesis permite que diferentes procesos que se ejecutan en diferentes nodos de un cluster pero que pertenecen al mismo entorno de ejecución (DC) utilicen recursos tal como si ejecutarían en un sistema centralizado (Imagen de Sistema Unico – SSI) constituyendo un nuevo modelo de arquitectura de virtualización.

Tesis: Se propone un modelo de Arquitectura de un Sistema de Virtualización Distribuido (DVS: Distributed Virtualization System), el cual permite construir dominios de ejecución denominados Contenedores Distribuidos (DC: Distributed Containers) que pueden extenderse más allá de los límites de una máquina física.

El objetivo es que las aplicaciones desplegadas en DCs dispongan de mayores cantidades de recursos y mayor poder de cómputo para poder brindar niveles de rendimiento, disponibilidad, escalabilidad y elasticidad más elevados. Los recursos (tanto físicos como abstractos) y procesos que constituyen un DC, pueden encontrase dispersos (y eventualmente replicados) en diferentes nodos de un cluster de virtualización.

Un DVS permite ocultar tanto a usuarios como a las aplicaciones cuáles son los nodos que componen un DC, cuántos de esos componentes se encuentran replicados, e incluso cuáles componentes se activan/desactivan cuando se producen fallos. Los usuarios y las aplicaciones de cada uno de los OSs que ejecutan en el entorno de un DC perciben un SSI.

El modelo propuesto conserva las características tan apreciadas de las tecnologías de virtualización actuales tales como confinamiento, consolidación y seguridad, y los beneficios de los DOSs tipo SSI, tales como la transparencia, mejor rendimiento, mayor elasticidad, disponibilidad y escalabilidad.

Tanto las aplicaciones legadas como las aplicaciones nativas para la Nube se benefician con sus características. En general, la migración de aplicaciones legadas que se ejecutan en servidores propios (on-premise) hacia aplicaciones desarrolladas para ejecutar en la Nube requiere de una importante inversión de dinero y tiempo. Si en la Nube estuviese disponible una interface estándar ofrecida por un VOS (tal como POSIX), pero sobre una infraestructura con capacidades extendidas a varios nodos de un cluster, las tareas de migración se simplificarían reduciendo los costos y tiempos.

Las aplicaciones desarrolladas bajo el modelo de MSA [2] o SOA [3] también se verían beneficiadas con un DVS. Un DVS es inherentemente distribuido, por lo que dispone de herramientas de comunicación que facilitan el intercambio de datos y la sincronización entre

servicios o micro-servicios. Un DVS presenta una vista única integral del conjunto de componentes de la aplicación que facilitan su despliegue, mantenimiento y operación.

Desde los inicios del proyecto, se comenzó con el desarrollo de un prototipo de DVS que permitiera validar diferentes aspectos considerados en el modelo, detectar errores de diseño y proponer mejoras. Se describen en la tesis los detalles de implementación de los diferentes componentes del prototipo y se presentan resultados de las mediciones previas a las optimizaciones. El prototipo demuestra la factibilidad de implementar un DVS, mediante el desarrollo módulos de kernel, bibliotecas, programas de pruebas, etc. para Linux. También se han convertido dos OSs a Virtual OSs (VOS) y se han utilizado herramientas y facilidades ya existentes en Linux.

CONTRIBUCIÓN

La contribución esperada de la tesis es un modelo de arquitectura de un DVS que permite conformar entornos de ejecución aislados (DCs) que aprovechan de la agregación de recursos computacionales de los nodos de un cluster. Como se pueden adoptar múltiples estrategias, enfoques, mecanismos y variantes para el modelo del DVS, la tesis se limita en su alcance a desarrollar aquellos componentes considerados centrales en importancia. Otras características y cualidades potenciales del modelo se proponen para futuros trabajos de investigación.

Además del modelo de arquitectura de DVS, en la tesis se realizan otras contribuciones a saber:

- ✓Un mecanismo de IPC que es independiente de la localización de los procesos (se encuentren los mismos en el mismo o en diferentes nodos), que soporta en forma transparente la migración de procesos, la distribución de cargas y la utilización de mecanismos de replicación de tipo Primario-Respaldo o tipo Máquina de Estado.
- ✓Un algoritmo distribuido tolerante a fallos de asignación de recursos (utilizado en el VOS distribuido del prototipo), que permite asignar recursos y mantener la consistencia y gran parte de la disponibilidad tanto en fallos de tipo de Detención Total o de Partición de Red.
- ✓Un VOS distribuido compuesto de varios procesos servidores dispersos en varios nodos del cluster, algunos de ellos replicados.
- ✓Un VOS tipo unikernel que contiene un servidor web, un sistema de archivos y una pila de protocolos TCP/IP del cual se pueden separar varios componentes hacia otros nodos, tales como el servidor de archivos o el servidor de almacenamiento en disco.

Más allá de prototipo construido, para el lector debe quedar claro que el modelo propuesto es de carácter teórico y genérico, el cual puede extenderse, modificarse o simplemente ser inspirador para otros tipos de enfoques.

ALCANCE DE LA TESIS

El alcance de la tesis abarca la descripción del modelo de arquitectura de un DVS que se propone como una nueva tecnología de virtualización, la descripción de sus componentes, el análisis de los beneficios que se obtienen con este sistema, la descripción del prototipo de DVS y una evaluación de rendimiento del mismo.

Aspectos relacionados a su integración con arquitecturas de gestión de servicios en la Nube quedan más allá del alcance. Estos y otros muchos tópicos relacionados quedan abiertos para futuros trabajos de investigación.

TECNOLOGIAS RELACIONADAS

El modelo de arquitectura de un DVS surge a partir de integrar múltiples tecnologías de virtualización, de OSs y de sistemas distribuidos en general. Todas ellas fueron fuentes de inspiración para construir el modelo propuesto.

1) *Sistemas Operativos Multiservidor o de Microkernel* [8]: A diferencia de los OS monolíticos, los OS multiservidor basados en microkernel descomponen al sistema en múltiples capas y múltiples procesos aislados que ejecutan en modo usuario. Estos procesos se comunican entre sí mediante mecanismos de IPC que provee el microkernel. El microkernel brinda servicios básicos a las capas superiores y es el único componente que ejecuta en modo kernel o modo supervisor.

2) *Sistemas Operativos Distribuidos con Imagen de Sistema Unica [9]*: Estos OSs distribuyen la ejecución de procesos en diferentes nodos de un cluster ofreciendo al usuario una imagen única de sistema. Algunos de estos sistemas permiten la migración de procesos para distribución y/o balanceo de cargas.

3) *Virtualización de Sistema Operativo - Sistemas Operativos Virtuales [10]*: Un OS ofrece servicios y abstracciones a las aplicaciones a través de su interface superior, y se comunica con el hardware de la plataforma a través de su interface inferior. Al igual que un OS, un VOS ofrece servicios y abstracciones a las aplicaciones a través de su interface superior, pero utiliza los servicios de un OS subyacente a través de su interface inferior. El funcionamiento es similar al de un OS ejecutando sobre un sistema paravirtualizado, excepto que utiliza Llamadas al Sistema (syscalls) en vez de Llamadas al Hipervisor (hypercalls).

4) *Virtualización basada en Sistema Operativo con Contenedores [11]:* El kernel del OS-Host particiona sus recursos en instancias aisladas en espacio de usuario. El sistema de virtualización construye un Contenedor que confina a un conjunto de aplicaciones y a su entorno de ejecución. Esta tecnología se basa en la virtualización de Llamadas al Sistema. Si bien todas estas instancias se ejecutan en sus propios entornos, todas comparten el mismo OS.

5) *Sistema de Comunicaciones Grupales (GCS) [12]:* Como el DVS es un sistema distribuido, se requiere de mecanismos de comunicación para sus diferentes componentes, y una alternativa es utilizar un GCS. Dentro de las posibilidades que ofrecen los sistemas distribuidos se encuentra la replicación; al utilizar GCS ésta debe ofrecer diferentes opciones de entrega de mensajes tales como FIFO, Causal, Orden Total, etc. Por otro lado, como los sistemas distribuidos que ejecutan en un cluster, la falla de uno de los nodos del mismo puede provocar una falla completa del sistema. Para reducir esta posibilidad, el sistema de GCS debe ofrecer a las aplicaciones mecanismos de tolerancia a fallos.

La descomposición (factorización) en múltiples tareas y servidores comunicándose mediante mecanismos de IPC como proponen los OS de microkernel brinda el aislamiento requerido por un sistema de virtualización y permite la distribución de los mismos en diferentes computadores siempre y cuando se puedan comunicar mediante IPC y GCS.

Actualmente los servicios de infraestructura proveen recursos tales como CPUs, memoria, disco y red, pero el usuario carece de una visión integrada tal como la que ofrece un OS multikernel debiendo gestionar explícitamente sus recursos en la Nube. El modelo de arquitectura de DVS propone una gestión integrada de recursos y transparencia para las aplicaciones. Estas características, simplifican la programación, el despliegue y mantenimiento de aplicaciones en la Nube.

MODELO DE ARQUITECTURA PROPUESTO

TOPOLOGÍA DE UN CLUSTER DVS

El modelo de arquitectura de DVS cumple con los requerimientos de los proveedores de Servicios en la Nube tales como alta disponibilidad, replicación, elasticidad, balanceo de carga, gestión de recursos y migración de procesos entre otros. Su objetivo principal es aumentar el rendimiento asignando recursos de un subconjunto de nodos a cada instancia de un VOS (agregación de recursos), pero permitiendo que múltiples VOS compartan los mismos nodos

(partición de recursos). Cada VOS se ejecuta en un entorno aislado denominado Contenedor Distribuido (DC), el cual puede abarcar desde uno a todos los nodos del cluster DVS. Un DC es un entorno de ejecución aislado y distribuido que ofrece recursos computacionales, virtuales y abstractos a un conjunto de procesos. En la Fig. 1, se presenta un ejemplo de topología de un cluster DVS que cuenta con cinco nodos unidos por una red de alta velocidad.

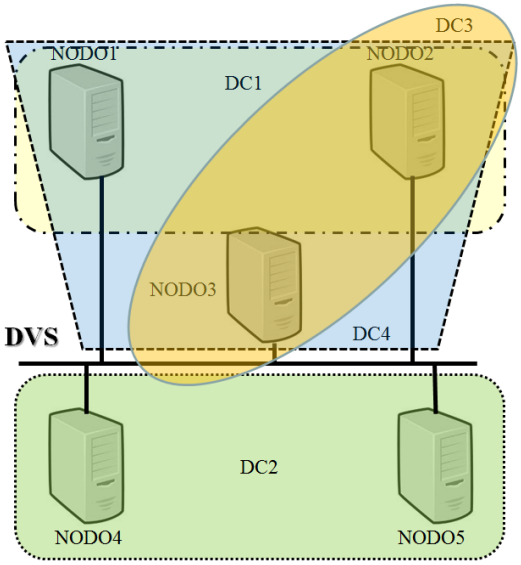

Figura 1.Ejemplo de Topología de un cluster DVS.

MODELO DE ARQUITECTURA DE DVS

El modelo de arquitectura de DVS está conformado por tres capas (Fig. 2). En su capa inferior se encuentran el hardware, con su OS-host que tiene embebido un módulo que conforma uno de los componentes del DVK.

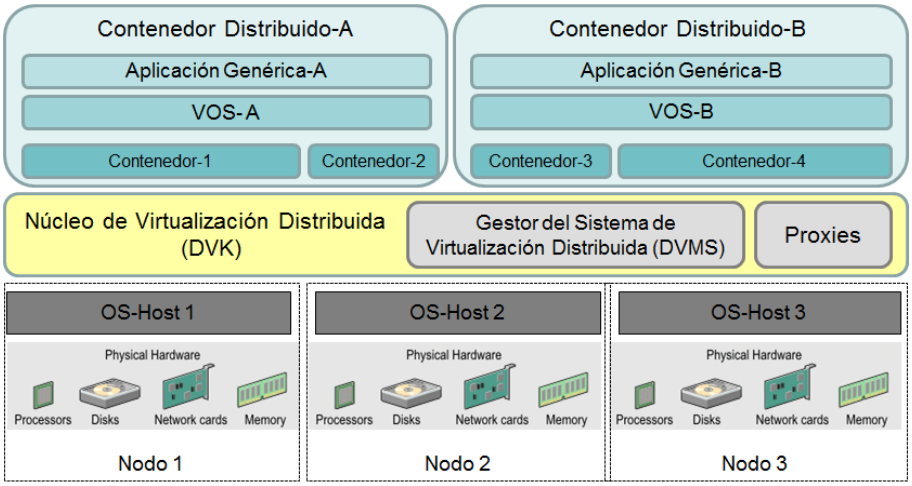

Figura 2.Modelo de Arquitectura del ***DVS***.

La capa intermedia la conforman el DVK, el Sistema de Gestión de Virtualización Distribuida (DVMS: Distributed Virtualization Management System) y los proxies. El DVK está compuesto por una serie de servicios que se ofrecen a la capa superior a través de sus APIs (Fig. 3), y los cuales utilizan servicios de la capa inferior a través del módulo embebido. El DVMS dispone de las APIs, comandos, bibliotecas, interfaces gráficas o interfaces web para gestionar el DVS y todos sus componentes. Cada servicio del DVK que se ofrece a las capas superiores (GCS, exclusión mutua distribuida, consenso distribuido, etc.) debe tener en consideración que los

procesos que involucran deben pertenecer al mismo DC, tal como lo hace el mecanismo de IPC. Es función del DVK controlar el aislamiento entre DCs para todos estos servicios.

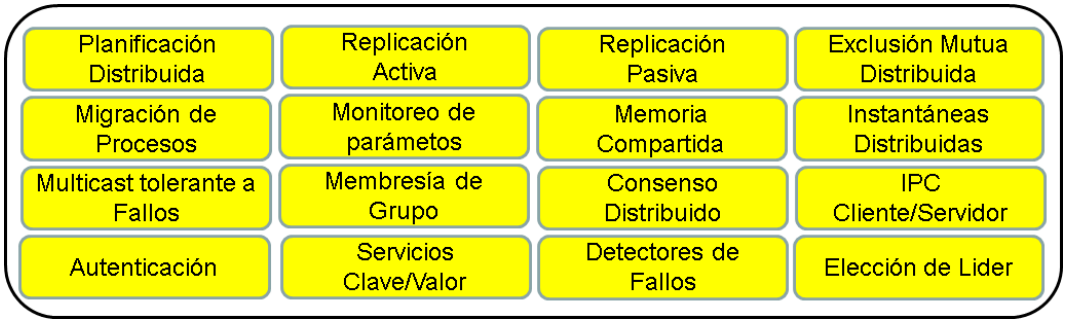

Figura 3. Servicios del Distributed Virtualization Kernel (*DVK*).

Los proxies son los procesos que transfieren mensajes y datos entre los nodos del cluster del DVS. Finalmente, la capa superior la conforman los DCs, los Contenedores, los VOS y las aplicaciones.

Núcleo de Virtualización Distribuida (DVK)

El DVK es la capa de software principal que integra los recursos del cluster, gestiona y limita los recursos asignados a cada DC. Ofrece a las capas superiores las interfaces necesarias para utilizar los protocolos y servicios de bajo nivel (Fig. 3), los cuales pueden ser utilizados por los VOS, tales como Planificación Distribuida, Replicación Activa/Pasiva, Exclusión Mutua Distribuida, mecanismos de migración de procesos, monitoreo de parámetros de rendimiento, Memoria Compartida Distribuida, Instantáneas Distribuidas, Multicast tolerante a Fallos, Membresía de Grupos, Consenso Distribuido, IPC Cliente/Servidor, servicios de clave/valor, autenticación, detección de fallos, elección de Lider, etc.

Representantes de Comunicaciones de Nodos Remotos (Proxies)

El cluster donde se ejecutará el DVS se define básicamente utilizando nodos y proxies. Los proxies de comunicaciones son los procesos responsables de transferir mensajes y bloques de datos entre un nodo Local y uno o más nodos remotos. Lo existencia de proxies se oculta al resto de los componentes del DVS tales como DCs, Contenedores y procesos registrados.

Contenedores

Los Contenedores son entornos de ejecución aislados que brinda el OS. Entre sus características significativas para su utilización en el DVS se destaca el aislamiento y el espacio de nombres (*namespace*) propio. El aislamiento de rendimiento se logra limitando ciertos recursos del OS asignados a cada Contenedor tales como tiempo de CPU, buffers, caches, reserva de ancho de banda de la red, E/S de dispositivos, memoria, etc.

Contenedores Distribuidos

Un DC también es un conjunto de Contenedores dispersos en diferentes nodos de un cluster, pero que permite comunicar los procesos de un Contenedor con los procesos de otro Contenedor en diferente nodo utilizando un IPC transparente a la localización. El mecanismo de IPC solo admite la comunicación entre procesos que se ejecutan en un mismo DC, estén estos localizados en el mismo o en diferentes nodos del cluster de virtualización.

Sistema Operativo Virtual

Se necesita un VOS para que las aplicaciones no pueden realizar Llamadas al Sistema al OS-host en forma directa, y para que puedan ejecutarse diferentes VOS sobre el mismo OS-host. Se admiten

diferentes tipos de VOS tales como monolíticos, Unikernels o Distribuidos los cuales pueden ejecutar concurrentemente en diferentes DCs.

PROTOTIPO

Prácticamente, en forma simultánea al diseño del modelo de arquitectura de DVS, se comenzó con el desarrollo de un prototipo con el fin de confirmar la factibilidad de la propuesta, detectar errores de diseño y proponer mejoras. Los componentes que lo constituyen son:

- *DVK*: En él se implementan, por un lado, los mecanismos de IPC (y sus APIs) extendidos para transferir mensajes y datos entre nodos de un cluster, para soportar migración y esquemas de replicación. Por otro lado, se implementaron los mecanismos de gestión del DVS tales como iniciar y detener el DVS, configurar, iniciar y detener DCs, registrar y de-registrar nodos y procesos proxies, registrar y de-registrar procesos ordinarios dentro de un DC, obtener información de estado y estadística de cada componente del DVS.
- *GCS*: Por cuestiones prácticas, en el DVK no se incorporan comunicaciones grupales, aquellas aplicaciones que las requerían utilizan Spread Toolkit [13].
- *VOS-multiservidor*: Como se disponía de la base del desarrollo de MoL (Minix Over Linux) se convirtieron sus componentes en los componentes de un VOS-multikernel,
- *VOS-unikernel:* Como base para el desarrollo de un VOS-unikernel se tomó el software LwIP [14], el cual dispone de una pila de protocolo TCP/IP completa, un servidor web básico con página estática en RAM, e interfaces de LAN virtual
- *Gestión por Webmin*: Si bien el prototipo puede ser gestionado mediante Interface de Línea de Comandos (CLI: Command Line Interface), o utilizando las APIs del DVMS que forman parte del DVK, se desarrolló un módulo Webmin que permite gestionar el DVS.

CONCLUSIONES

Los proveedores de IaaS siempre requieren de altos niveles de rendimiento, escalabilidad y disponibilidad para brindar sus servicios de virtualización. Estos requerimientos se pueden satisfacer con una tecnología de virtualización distribuida. El modelo de arquitectura de DVS propuesto combina e integra las tecnologías de virtualización y de los Sistemas Distribuidos para proporcionar los beneficios de ambos mundos, lo que lo hace adecuado para ofrecer Servicios en la Nube con nivel de proveedor. La arquitectura se basa en poner a disposición de los VOS y aplicaciones mecanismos de IPC, GCS y servicios autónomos, distribuidos y débilmente acoplados otorgándoles flexibilidad para los desarrollos de nuevos servicios y aplicaciones, pero manteniendo las características de aislamiento requeridas.

Un DVS es que facilita la migración/portación de aplicaciones heredadas desde servidores locales (on-premise) a la Nube. Con un VOS que suministre las API POSIX y RPC permitiría la reutilización de gran parte del código de esas aplicaciones, mejorando los costos y tiempos de implementación al reducir el esfuerzo de codificación. Por otro lado, también se adapta perfectamente a las aplicaciones basadas en el MSA o SOA, éstas pueden ejecutar un conjunto de microservicios en varios nodos del clúster mediante el uso de IPC/RPC propio del DVS para las comunicaciones.

Con un DVS se da respuesta a las dos preguntas planteadas en la Introducción: 1) Los límites de un entorno aislado para la ejecución de aplicaciones (DC) pueden expandirse a todos los nodos del clúster (agregación) mejorando sus características de rendimiento, escalabilidad y disponibilidad. Además, los nodos de un cluster pueden compartirse entre varios DCs (partición), mejorando así la utilización de la infraestructura de hardware. 2) Un DVS soporta un VOS distribuido ejecutando dentro de un DC el cual le aporta una visión integrada de las aplicaciones equivalente a un sistema centralizado. Esta visión integrada facilita la gestión reduciendo los tiempos y los costos de implementación, operación y mantenimiento de aplicaciones en la Nube.

El modelo de arquitectura propuesto abre la posibilidad de investigar y desarrollar varios aspectos relacionados tanto al DVS propiamente dicho, como así también acerca de algunos de sus componentes, tales como los VOSs.

TRABAJOS FUTUROS

Para los próximos años se planea trabajar en los siguientes proyectos de investigación y desarrollo (los cuales están actualmente en curso bajo el PID 5162 – UTN - FRSF) que pueden producir un gran impacto en la adopción del modelo de DVS:

1) Incorporar estándares de gestión tales como Openstack[15],

2) Incorporar a UML (User Mode Linux) como VOS,

3) Incorporar un VOS Unikernel generado con Rumpkernel [16] con soporte POSIX.

4) Desarrollar un Router Virtual Distribuido basado en Contrail vRouter [17] que utilice Virtual Network Interface Cards (vNICs) localizadas en diferentes nodos del cluster DVS.

El éxito en estos proyectos aumentaría la confianza para la adopción del modelo propuesto.

REFERENCIAS

[1] P. M. Mell, T. Grance, "*The NIST Definition of Cloud Computing*", Technical Report, NIST, SP 800-145, 2011.

[2] C. Pautasso, O. Zimmermann, M. Amundsen, J. Lewis, and N. Josuttis, "*Microservices in Practice, Part 1: Reality Check and Service Design*", IEEE Softw. 34, pp 91-98, Jan. 2017, 2017.

[3] N. Bieberstein et al., "*Service-Oriented Architecture Compass*", Pearson, ISBN 0-13-187002-5, 2006.

[4] J. Turnbull, "*The Docker Book*", 2014, Available online at: https://www.dockerbook.com/, accedido en Enero 2018.

[5] B. Hindman, A. Konwinski, M. Zaharia, A. Ghodsi, A. D. Joseph, R. Katz, S. Shenker, and Ion Stoica, "*Mesos: a platform for fine-grained resource sharing in the data center*", Proc. of the 8th USENIX conference on Networked systems design and implementation (NSDI'11), USENIX Association, Berkeley, CA, USA, pp. 295-308, 2011.

[6] N. Poulton, "*The Kubernetes Book*", ISBN-13: 978-1521823637, ISBN-10: 1521823634 ,2017.

[7] D. Wentzlaff, C. Gruenwald III, N. Beckmann, K. Modzelewski, A. Belay, L. Youseff, J. Miller, A. Agarwal, "*An operating system for multicore and clouds: mechanisms and implementation*", in Proceedings of the 1st ACM symposium on Cloud computing (SoCC '10). ACM, New York, NY, USA, 3-14, 2010.

[8] A. Tanenbaum, et al, "*Minix 3: Status Report and Current Research*", login: The USENIX Magazine, 2010.

[9] OpenSSI (Single System Image) Clusters for Linux, http://www.openssi.org/cgi-bin/view?page=openssi.html , accedido Enero 2018.

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

[11] G. Banga, P. Druschel, J. C. Mogul, "*Resource Containers: A New Facility for Resource Management in Server Systems*", in Operating Systems Design and Implementation, 1999.

[12] K. P. Birman, "*The process group approach to reliable distributed computing*", Commun. ACM 36, Dec. 1993, pp. 37-53, doi:http://dx.doi.org/10.1145/163298.163303.

[13] The Spread Toolkit. http://www.spread.org , accedido Enero 2018.

[14] LwIP, http://savannah.nongnu.org/projects/lwip/ , accedido Enero 2018.

[15] Openstack, https://www.openstack.org/ , accedido Enero 2018.

[16] Rumpkernel, http://rumpkernel.org/ , accedido Enero 2018.

[17] vRouter Contrail, https://github.com/Juniper/contrail-vrouter, accedido Enero 2018.