Localización y Seguimiento de Servicios Replicados en Sistemas Distribuidos

Pablo Pessolani, David Gabriel Harispe, Octavio Garcia Aguirre

Departamento de Ingeniería en Sistemas de Información

UTN- Facultad Regional Santa Fe

Santa Fe – Argentina

{ppessolani, dharispe, oaguirre}@frsf.utn.edu.ar

Resumen

Las aplicaciones para ejecución en la nube suelen factorizarse en múltiples componentes que localizan en diferentes computadores físicos o virtuales. Para alcanzar los niveles de disponibilidad, escalabilidad y robustez que éstas requieren, se deben realizar configuraciones más complejas y más costosas de implementar, operar y mantener.

Una forma de resolver este problema es haciendo uso de un Sistema de Virtualización Distribuido (DVS) que provee un mecanismo que permite comunicar entre sí los componentes de la aplicación en forma transparente a su localización, ocultando los problemas y complejidades añadidos por su ejecución distribuida.

En este artículo se detalla el desarrollo y funcionamiento de un servicio denominado RADAR, utilizado en el contexto de un DVS. Las tareas que desarrolla RADAR son: 1) la localización automática de los servicios ejecutando en el DVS, y 2) el seguimiento de la localización de esos servicios mediante la redirección automática de las comunicaciones cuando el servidor presenta algún fallo, o cuando éste ha migrado de nodo o cuando se ha producido una partición de red. Con el uso de RADAR en un DVS se pueden ejecutar múltiples instancias de servidores de forma transparente (no será necesario que los clientes conozcan de la existencia de las múltiples réplicas) resolviendo el problema de mantener una conexión cliente/servidor en presencia de fallos o ante la migración de procesos servidores.

1. Introducción

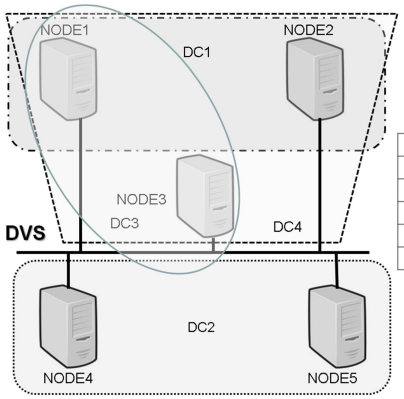
En la actualidad, las aplicaciones en la nube demandan cada vez más recursos, los cuales no pueden ser provistos por un único computador. Es por ello que las mismas se desarrollan para ejecutar en entornos distribuidos a fin de incrementar su poder de cómputo y almacenamiento, como también proveer alta disponibilidad y robustez con calidad de proveedor. Utilizando un sistema distribuido se pueden extender las capacidades de cómputo y almacenamiento a varios equipos físicos (nodos) distintos. Si bien existen diversas tecnologías de procesamiento distribuido, son muy valoradas aquellas que ofrecen más facilidades en cuanto a su implementación, operación y mantenimiento que redundará en menores costos. Son muy apreciadas aquellas tecnologías que brindan un Imagen de Sistema Único (SSI) y que ocultan tanto a programadores como a usuarios aspectos tales como la localización y migración de los procesos, direcciones IPs de uso interno, puertos TCP/UDP, etc. y sobre todo, cuando ocultan fallos mediante la utilización de mecanismos de replicación. Una tecnología que dispone de esas características es un Sistema de Virtualización Distribuido (DVS) [1].

Un DVS ofrece entornos de ejecución virtuales distribuidos en los cuales se permite ejecutar múltiples instancias aisladas de Sistemas Operativos Virtuales (VOS) [2]. Un VOS ofrece abstracciones y servicios a las aplicaciones de usuario, pero no gestiona hardware físico, sino dispositivos virtuales proporcionados por un Sistema Operativo (OS) real. Los recursos con los que cuenta el DVS están dispersos en diversos nodos de un cluster. Un DVS además, presenta capacidades de agregación (permite utilizar múltiples nodos de un cluster para el mismo VOS), y de particionado (permite ejecutar múltiples componentes de diferentes VOSs en un mismo nodo) en forma simultánea. Cada VOS distribuido se ejecuta dentro de un contexto de ejecución o dominio aislado denominado Contenedor Distribuido (DC). En la Fig. 1 se muestra un esquema topológico de un cluster DVS.

Un problema que debe considerarse cuando se utiliza un VOS distribuido está relacionado con la localización de un determinado servicio utilizado por algún cliente externo o por otro componente del mismo VOS. Una forma de resolver este problema por fuera del DVS sería utilizando protocolos de Internet ya existente. Con el protocolo DNS se puede localizar la dirección IP del servidor, y con ARP se puede localizar la dirección MAC del mismo dentro de una LAN. Sin embargo, una cuestión que debe considerarse al trabajar con un cluster, es que la red y sus nodos pueden fallar impidiendo la continuidad en la prestación de un servicio dado.

Para superar este tipo de contingencia se suelen utilizar mecanismos de replicación tal como Primary/Backup (P/B) [3]. En un esquema P/B un servidor de Backup sustituye a un servidor Primario cuando éste sufre un fallo, dando continuidad a la prestación de un servicio y

mejorando la disponibilidad y robustez del mismo. Este servidor de reemplazo, que puede ser temporario o no, tendrá una nueva dirección IP (y eventualmente dirección MAC), lo que implica que los demás componentes (o clientes) que requieren de sus servicios las conozcan.



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

Figura 1. Topología de un DVS.

Esto podría resolverse con protocolos como HSRP [4] o VRRP [5] que utilizan interfaces de LAN *fantasmas* que son las referidas por los clientes. El servidor Primario adquiere la dirección IP/MAC de la interfaz fantasma, y cuando éste falla, el Backup se convierte en nuevo Primario y la adquiere para sí. Los clientes, mientras tanto, no perciben esa modificación, salvo por algún pequeño retraso en la respuesta del servidor.

En el caso de servicios que utilicen el esquema de Máquinas de Estado Replicadas (RSM) [6], podrá ser el DNS (por ejemplo, con la utilización de Round Robin DNS) el que informa al cliente de la IP de un server.

Estas soluciones, si bien son factibles y muy utilizadas, requieren de numerosas configuraciones adicionales que inducen mayores costos en su implementación como en su operación. Por otro lado, los programadores deberán considerar en sus aplicaciones problemas tales como la localización de servidores después de fallos o migraciones. Por estos motivos, se propone aquí una solución más sencilla y transparente que mantiene el estado y configuración del cluster.

Un DVS mantiene como parte de su estado la identificación de los nodos que lo componen y de los nodos que componen cada DC. Sería una característica muy valorada en un DVS disponer de un mecanismo que realice el seguimiento en la localización de los servicios simplificando las aplicaciones y resolviendo los problemas de comunicación que puedan surgir entre un cliente y un servidor producto de cambios de localización de estos últimos. Los cambios de localización se suelen producir por fallos de procesos, de nodos o de red, o por la migración del servicio a otro nodo del cluster.

Cuando un servidor cambia de localización desde un nodo origen a un nodo destino, los procesos que se comunican con él deben mantener sus comunicaciones a pesar del cambio y en forma transparente a fin de simplificar la programación. Para resolver la problemática planteada en forma sencilla se desarrolló RADAR.

Haciendo uso de las características presentes en el DVS se desarrolló RADAR, que utilizando un Sistema de Comunicaciones de Grupos (GCS), detecta fallos en los nodos y procesos que componen el sistema distribuido. Luego de detectar un fallo en un nodo o en un proceso o en la red es capaz de efectuar la redirección automática de las comunicaciones de los clientes al servidor que brindará el servicio. En cada nodo del DVS se debe ejecutar una instancia de RADAR, las que se comunican entre sí utilizando el GCS.

Para el prototipo de RADAR, el GCS elegido fue Spread Toolkit [7], el cual detecta todos los tipos de fallos mencionados, pero podrían utilizarse otras herramientas equivalentes. Para realizar las pruebas de funcionamiento sobre el prototipo de RADAR se utilizó un servicio de disco replicado denominado RDISK [8], el cual utiliza un algoritmo capaz de sincronizar imágenes de disco entre un servidor Primario y uno o más servidores de Backup.

El funcionamiento de RADAR es totalmente transparente a la red subyacente debido a que no requiere de configuraciones de red adicionales a las ya establecidas al momento de configurar el DVS, facilitando su despliegue y mantenimiento.

El resto del artículo se organiza de la siguiente manera: en la Sección 2 se analizan trabajos relacionados y sus características principales. En la Sección 3 se presenta la base tecnológica en base a la que se desarrolló RADAR. En la Sección 4 se describe el diseño e implementación del prototipo de RADAR. Las pruebas realizadas y los resultados obtenidos se exponen en la Sección 5 y finalmente, las conclusiones y trabajos futuros se presentan en la Sección 6.

2. Trabajos Relacionados

La problemática planteada no es desconocida por la comunidad científica, por lo que previamente se han llevado a cabo diversos trabajos de investigación y desarrollo para darle solución. En esta sección se analizan las herramientas actuales más utilizadas.

Zookeeper [9] es una herramienta que provee un repositorio distribuido de clave/valor. Ofrece un servicio distribuido de configuración, un servicio de sincronización, y un registro de nombres para grandes sistemas distribuidos.

Zookeeper es un confiable porque soporta fallos entre los nodos. Si la conexión con el servidor se interrumpe, el cliente se conectará a un servidor diferente del grupo. Si bien *Zookeeper* es utilizado por grandes compañías, tiene algunos inconvenientes. Es de alta demanda computacional como consecuencia de que está desarrollado en Java y que tiene gran cantidad de dependencias. Para el registro y descubrimiento de servicios se requiere adicionar sistemas complementarios.

etcd [10] es otra herramienta que ofrece un repositorio distribuido clave/valor, consistente y confiable. Entre sus

características destacables cabe mencionar que brinda alta disponibilidad, rápidos tiempos de respuesta y transacciones seguras. Además, tolera la elección de líderes durante particiones de red y las fallas en nodos. Para que *etcd* pueda realizar tareas como el registro y el descubrimiento de servicios, es necesario complementarlo con otras herramientas.

Consul [11] es una herramienta distribuida de alta disponibilidad y consistencia utilizada para el descubrimiento de servicios, control de estado y como repositorio clave-valor. Al igual que *etcd,* utiliza el algoritmo de consenso distribuido *Raft* [12].

Consul está integrado por un sistema de descubrimiento y registro de servicios, por lo que no es necesario combinarlo con otras herramientas.

Kubernetes [13] es un gestor de contenedores desarrollado por Google. Permite automatizar la implementación, el escalado y la administración de aplicaciones en contenedores. *Kubernetes* ejecuta cargas de trabajo en unidades llamadas *pods*, para los que dispone de un servicio de localización. Se ofrecen 2 mecanismos básicos para localización de servicios: Variables de entorno y DNS.

Docker [14] es otro gestor de contenedores ampliamente utilizado para el despliegue de aplicaciones en la nube. *Docker Cloud* crea una red privada superpuesta que conecta todos los Contenedores de los nodos contratados por el usuario otorgándole a cada Contenedor su propia dirección IP. Se dispone de dos mecanismos para localizar servicios: 1) utilizando el nombre del host y del servicio (algo similar al DNS) y 2) utilizando *Enlaces de Servicio,* los que crean variables de entorno que permiten la comunicación con otros Contenedores o con servicios externos.

Tanto *Kubernetes* como *Docker* utilizan direcciones IPs (o hostnames) y puertos TCP/UDP para identificar los servicios. Esto hace más complejo el despliegue y gestión de aplicaciones, como así también en la protección de las mismas, dado que se debería controlar el tráfico entre direcciones IP/puertos.

3. Bases y Fundamentos Tecnológicos

En esta sección se presentan aquellos desarrollos, productos y herramientas que se utilizaron como bases y fundamentos tecnológicos para el diseño de RADAR y para el desarrollo e implementación de su prototipo.

3.1. M3-IPC

Uno de los componentes de un DVS es el Kernel de Virtualización Distribuido (DVK) el cual se encuentra integrado al kernel de Linux como un módulo. El DVK provee a los programadores de un avanzado mecanismo de Comunicación entre Procesos denominado M3-IPC [15] que se instala en todos los nodos del cluster.

M3-IPC proporciona herramientas para llevar a cabo la comunicación transparente entre procesos localizados en el mismo o en diferentes nodos. M3-IPC utiliza procesos *Proxies de Comunicaciones* para enviar mensajes y datos entre nodos y que pueden utilizar diferentes protocolos de transporte/red. También dispone de las características requeridas por un DVS de clase-proveedor tales como, ocultar las interrupciones de las comunicaciones entre procesos replicados cuando se producen fallos en los servidores o cuando estos migran a otros nodos del cluster DVS.

M3-IPC soporta la replicación de servicios utilizando métodos de replicación Primary/Backup (P/B) [3] y de Máquina de Estados Replicada (RSM) [6]. En el primer caso, un proceso llamado Primario es el proceso activo brindando un servicio, y podría haber varios procesos pasivos o de Backup. Cuando el Primario falla, uno de los procesos de Backup se transforma en el nuevo Primario y mantiene activo el servicio. En el segundo caso (RSM), un grupo de procesos proporciona un servicio a los clientes como si fuese un único servidor. Todos los procesos están activos, y cuando uno o más de ellos fallan, los otros procesos continúan brindando el servicio.

M3-IPC también mantiene la continuidad de las comunicaciones ante la migración de procesos. Cuando un proceso migra de un nodo a otro, las comunicaciones se restablecen una vez que el proceso ha migrado o ha regresado a su nodo origen (migración fallida).

Los procesos M3-IPC son identificados por medio de un número entero que se denomina *endpoint*. Este *endpoint* no guarda relación alguna con la ubicación de cada proceso, éste se mantiene aún después de una migración del proceso. Esta característica se transforma en una importante propiedad que facilita la programación de aplicaciones, su despliegue y operación.

3.2. Spread Toolkit

Para intercambiar información entre los procesos que se ejecutan en los nodos de un cluster o en la nube se requiere de mecanismos de comunicación con características tales como confiabilidad, tolerancia a fallos y un alto rendimiento. Existen varios sistemas y herramientas que ofrecen disponen de estas características tales como JGroups [16], Appia [17] o Spread Toolkit [7]. Al desarrollar RADAR se seleccionó *Spread Toolkit* por haber sido utilizado como GCS en otros trabajos del grupo de investigación de los autores [8]. *Spread Toolkit* es un GCS de código abierto desarrollado en lenguaje C utilizado en sistemas o aplicaciones distribuidas en donde la confiabilidad, el rendimiento y la estabilidad de las comunicaciones son características requeridas.

Entre los servicios básicos que provee *Spread Toolkit* se encuentra la abstracción de grupo donde un nombre representa un conjunto de clientes. Dispone de APIs para la difusión (multicast) de mensajes a un grupo, la notificación de membresías a un grupo, la entrega confiable de mensajes (aún ante fallos de red o de miembros del grupo) y la detección de fallos de miembros

del grupo o la red. También soporta varias formas de orden en entrega de los mensajes a los miembros (FIFO, Causal, Atómico, etc.) convirtiéndola en una herramienta sumamente versátil para el desarrollo de sistemas distribuidos.

Spread Toolkit está basado en el modelo de membresía de grupo denominado *Extended Virtual Synchrony* (EVS) [18] tolerando fallos de partición de red y su reunión, fallos de nodos, de procesos y posterior reinicio.

4. Diseño e Implementación de RADAR

RADAR es un algoritmo distribuido que detecta cambios en la localización de *endpoints* tanto en esquemas de replicación P/B, RSM, como ante la migración de servicios. Utilizando al GCS, RADAR detecta fallos en los nodos, red y servicios que componen el sistema distribuido.

Ante un fallo (fallo de procesos y/o nodos, fallos de partición de red y reunión de red) o migración, RADAR re-configura el DVS de tal forma que los clientes mantienen su comunicación con (el ahora) nuevo servidor. Esto evita la necesidad de realizar cambios en la configuración de los mismos. El servidor se identifica sólo con su *endpoint* para el cual ya se estableció la nueva ubicación (a través de las APIs del DVK).

Si el proceso Cliente de un servicio no contara con RADAR para localizar la nueva ubicación de un servidor, no podría mantener la continuidad en las comunicaciones con él. El cliente podría quedar a la espera de la respuesta de un servidor fallido o localizado en otro nodo.

Si se dispone de RADAR, el seguimiento por cambio de ubicación del servidor es transparente y automático. El proceso Cliente es capaz de mantener su comunicación (con un pequeño retardo inicial) dado que es el DVK quien establece dónde y cómo alcanzar la nueva localización del servidor a partir de la información suministrada por RADAR.

4.1. Funcionamiento de RADAR

RADAR implementó como múltiples Máquinas de Estados Finitos (FSM), una por cada servicio a monitorear. La Fig. 2 presenta el diagrama de flujo de la función principal de RADAR.

Los pasos que se siguen son:

1. Se lee el archivo de configuración, que recibe como argumento. En éste se especifican parámetros tales como los *endpoints* a controlar, los nombres de los servicios a monitorear con sus identificadores de DC (DCID), etc.
2. Se inicializan el driver del DVK y las librerías de Spread y del DVS.
3. Se verifica si cada servicio a monitorear tiene registrado su *endpoint* en el DVK local; si no lo está, inicializa las variables de lo contrario, actualiza las mismas en forma similar a cuando lo hace cuando se ha producido un fallo del servicio.
4. Por cada servicio a monitorear se crea un hilo (thread) como FSM.
5. Cada hilo se conecta al grupo Spread del servicio a monitorear e inicia la FSM con las siguientes operaciones en ciclo:
   a. Espera recibir un mensaje.
   b. Verificar el tipo de mensaje.
   c. Según el tipo de mensaje, ejecutar la función correspondiente, luego volver al punto a.

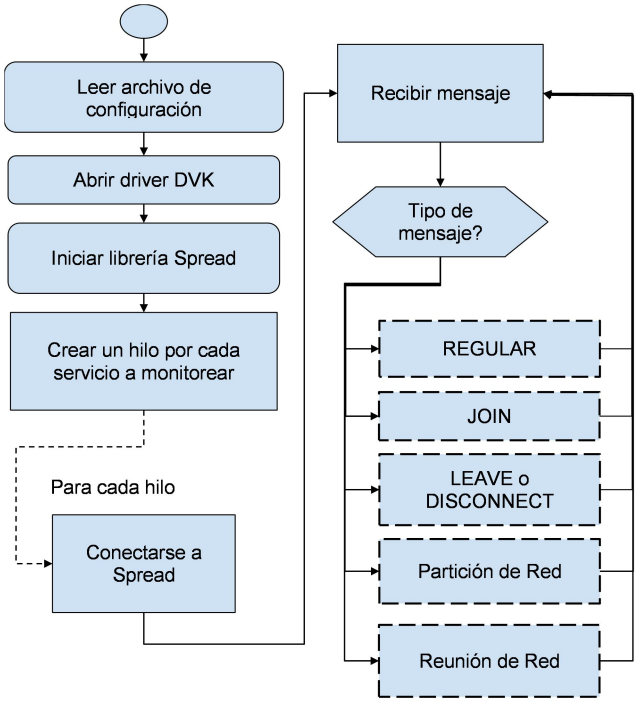


Figura 2.Función principal de RADAR

Los mensajes con los que opera RADAR pueden provenir de los siguientes orígenes:

- *De Spread*: Cuando se ha producido un cambio de membresía o de red en el cluster o en los procesos que pertenecen al grupo. JOIN, indica que un nuevo integrante se ha unido al grupo. *LEAVE*/*DISCONNECT* indican que un miembro del grupo ha finalizado explícita/implícitamente. En tanto que mensajes de tipo *NETWORK* indica alguno de los siguientes eventos:
  a. Un nodo ha fallado,
  b. Un nodo se ha desconectado de la red
  c. Se produjo una partición de red
  d. Se produjo una re-unión de una red partida.
- *Del Servidor Primario del servicio monitoreado*: Este informa que él es el Primario y en el nodo donde se localiza. Si bien el mecanismo RSM no distingue a uno de sus procesos como Primario, a los efectos de la utilización de RADAR, éste se distingue como aquel servidor que le informa de cambios.

A continuación, se detalla el tratamiento que RADAR da a los diferentes tipos de mensajes.

4.2. Tratamiento de Mensajes

4.2.1. Mensajes Regulares de los miembros del Grupo del Servicio.

Los mensajes de tipo *Regular* son enviados por los miembros del grupo que brinda el servicio de forma explícita (en general lo hace el Primario). En el desarrollo de este artículo se contempla que los nodos que brindan el servicio envíen a RADAR su información local sobre cuáles nodos están sincronizados (es decir, que pueden asumir la función de activos respecto al servicio). Si bien esta información no es utilizada por RADAR, podría ser utilizada por otras herramientas de gestión del DVS.

Para disponer de información actualizada, cada vez que hay un cambio de configuración del grupo que conforma el servicio, el servidor Primario envía un mensaje Regular de tipo *MC_RADAR_INFO* al grupo (utilizando un servicio de multicast del GCS) con el resultado de esos cambios. Si el estado del RADAR en un nodo es "*Sin Primario*" o con el "*Primario Fallido*" o "*Desconectado*", se utiliza la información del mensaje *MC_RADAR_INFO* para determinar qué acción realizar. A partir del mensaje recibido se obtiene la información sobre cuáles nodos están sincronizados y se establece la localización (nodo) del servidor Primario (*Commit*) o se vuelve atrás la misma (*Rollback*). Esta acción dependerá de si el servicio se ha relocalizado o si el mismo se ha reanudado en el nodo original tal como estaba antes del evento que originó el mensaje.

4.2.2. Tratamiento de Mensajes *JOIN*

La instancia de RADAR de cada nodo solo actualiza la información del conjunto de nodos que componen el grupo que brinda el servicio acorde a lo informado por el GCS en los campos de información del mensaje. Si el emisor que ingresaba al grupo era un miembro RADAR, se registra también el cambio en un mapa de bits de miembros RADAR para ese servicio.

4.2.3. Mensajes *LEAVE* o *DISCONNECT*

Si el mensaje recibido de tipo *LEAVE* o *DISCONNECT* es consecuencia de la finalización de un miembro RADAR, se actualiza la información de nodos y miembros RADAR (en su mapa de bits) acorde a lo informado por GCS en el mensaje. Si el miembro que finalizó o abandonó el grupo era el servidor Primario, el miembro RADAR local de cada nodo cambia su estado a *NO_PRIMARY_DEAD* y notifica al DVK que el *endpoint* no se encuentra disponible. Luego, se detienen todas las transferencias de datos y mensajes con ese *endpoint*.

4.2.4. Mensajes *NETWORK* de partición o de re-unión de la Red

El GCS envía este tipo de mensajes cuando un nodo se desconecta, o cuando se ha producido una Partición o Re-unión de la red del cluster DVS. Cuando ocurre una partición de red, algunos nodos quedan accesibles al miembro RADAR del nodo local y otros no. El grupo original de nodos se subdivide y cada subgrupo queda sin conexión con el resto. Cuando esto sucede se necesita recrear la información del grupo, y ver cuáles nodos están en la partición del nodo local donde ejecuta la instancia de RADAR.

Si el nodo Primario original se encuentra en el mismo grupo que la instancia de RADAR, no se realiza ninguna acción. En cambio, si el nodo Primario ya no se encuentra disponible se inicia la preparación para que un nodo Backup pueda continuar el servicio como nuevo Primario. Además, se detienen todas las transferencias de datos y mensajes con *endpoint* del Primario a ser sustituido.

Cuando no se encuentran nodos que dispongan de procesos de Backup del servicio monitoreado, el sistema pasa al estado *NO_PRIMARY_NET*, deteniendo las comunicaciones con el *endpoint* monitoreado hasta que un nuevo Primario se registre en la misma partición o se realice una Re-unión de la red partida.

4.2.5. Mensajes *NETWORK* de Fallo de Desconexión del Servidor Primario

Ante un fallo de desconexión del servidor Primario se inicia el seguimiento del servicio deteniendo las comunicaciones con ese *endpoint*. Luego, se elimina al Primario de la lista de nodos, se almacena la información del nodo donde se localizaba el Primario anterior y se establece un estado para RADAR que puede ser *NO_PRIMARY_DEAD* que indican fallo de finalización del servidor Primario, o *NO_PRIMARY_NET* que indica una partición de red que impide alcanzar el nodo del servidor.

4.3. Estados de RADAR

De acuerdo a los distintos mensajes recibidos y situaciones detectadas, RADAR puede estar en los siguientes estados:

- *Sin Primario al Inicio (NO_PRIMARY_BIND)*: Cuando la aplicación inicia por primera vez se encuentra en este estado hasta que se logra registrar el *endpoint* del Primario al DVK del nodo local. Si por algún motivo no lo logra (por ejemplo, el servicio no se está ejecutando), pasa al estado *NO_PRIMARY_DEAD*.
- *Con Primario*: Se ha localizado el nodo donde se encuentra registrado el *endpoint* del servidor Primario.
- *Sin Primario por Fallo (NO_PRIMARY_DEAD):* Ante un fallo del proceso servidor Primario, el GCS genera un evento de *LEAVE* o *DISCONNECT*. Se detienen las comunicaciones de los procesos locales con el *endpoint* del servidor Primario.
- *Sin Primario por Partición de Red (NO_PRIMARY_NET):* Ante una partición de la red donde el nodo donde ejecuta el servidor Primario no se encuentre en la partición del nodo de RADAR local, se detienen las comunicaciones de los procesos locales con el *endpoint* del servidor Primario.

Es importante señalar, que si bien en el esquema de replicación RSM no existe el concepto de servidor Primario, para el proceso Cliente su servidor "*Primario*" será aquel que tiene asignado para realizar sus peticiones.

5. Evaluación

En esta sección se describen el escenario utilizado y la batería de pruebas realizadas sobre el algoritmo RADAR, y se analizan los resultados de cada una de ellas.

5.1. Escenarios de Pruebas

La batería de pruebas a las que se sometió el algoritmo RADAR fue diseñada para simular diferentes fallos de entorno reales (Fig. 3). Para realizarlas se utilizó un banco de pruebas en un entorno virtual. Se crearon Máquinas Virtuales (VMs) conectadas entre sí mediante una red virtual que simulaban un cluster DVS. La virtualización se realizó utilizando VMware Workstation y se configuró un cluster de 3 nodos: *Node0*, *Node1* y *Node2*. El proceso a monitorizar fue un servidor de almacenamiento (disco remoto) denominado RDISK. Cliente y Servidor se ejecutaron en el entorno de un Contenedor Distribuido (DC0).

En los nodos *Node0* y *Node1* se ejecutaron instancias de RDISK [8] que serían Primario/Backup, en tanto en Node2 se ejecutó el proceso Cliente de RDISK el cual requería el seguimiento del servicio. Por supuesto, se ejecutó una instancia local de RADAR en *Node2*.

La ejecución de las distintas pruebas y sus resultados se pueden observar en la Fig. 3. En ella se muestra el estado de los procesos registrados en el DVK de cada nodo según el contenido (dinámico) del archivo */proc/dvs/DC0/procs* en los momentos de importancia en cada prueba. Este archivo contiene la lista de procesos en ejecución en el DC0 y los estados de los mismos. Se puede observar, entre otras cosas el número de nodo donde se encuentra ejecutando un proceso determinado (columna *nd*) y un código (columna *flag*) que indica su estado. Los estados que se identifican en estas pruebas son: 8 (proceso Local), 1000 (proceso Remoto alcanzable) y 1800 (proceso Remoto no alcanzable).

A continuación, se detallan las pruebas realizadas.

5.1.1. Fallo y restitución de la conexión del servidor Primario sin Backup

En esta prueba (Fig. 3-A) solo se ejecutó una instancia de RDISK en *Node0*, de manera que solo existe un servidor Primario y no se dispone de servidores de Backup.

Inicialmente, RADAR asoció el proceso RDISK al *Node0*, estableciendo en consecuencia a *nd=0* y *flag=1000* en *Node2*.

Cuando se provocó un fallo de red en *Node0*, el estado del proceso RDISK en *Node2* cambio al valor *flag=1800* para indicar que el proceso no estaba activo o no era alcanzable.

Una vez que se restableció la conexión de red en *Node0*, se volvió a detectar la disponibilidad del proceso RDISK y su estado cambió a *flag=1000*, sin modificarse el valor de *nd*.

5.1.2. Fallo de la conexión del servidor Primario con Backup

Para esta prueba (Fig. 3-B) se incorporó un servidor de RDISK de Backup (*Node1*) del Primario (*Node0*). Se puede observar que en el archivo del directorio */procs* del servidor Backup, el proceso RDISK posee un estado *flag=1000* con *nd=0* indicando que el *endpoint* es Remoto localizado en Node0.

Cuando se finaliza abruptamente el proceso de RDISK en *Node0* (mediante un comando *kill*), el GCS detecta el cambio e informa a RADAR que comienza la secuencia de seguimiento para asociar RDISK a otro nodo. La instancia de RDISK del *Node1*, también detecta el fallo del Primario, ejecuta un algoritmo de elección y como único miembro sincronizado del grupo se proclama como nuevo Primario. Es entonces cuando realiza un multicast de un mensaje de tipo *MC_RADAR_INFO*. Todas las instancias de RADAR de los diferentes nodos actualizan la información del nodo donde se está ejecutando el nuevo Primario, en este caso *Node1*.

Como resultado se puede observar que RDISK en el archivo del directorio */procs* de *Node2* posee un estado de *flag=1000* (Remoto activo/alcanzable) y ubicado en *Node1* (*nd=1*). Además, se observa que en el archivo del directorio */procs* de *Node1* ha cambiado e indica que RDISK es un proceso local (*flag=8*). Esto siempre es así con el proceso RDISK en el nodo que ejecuta como servidor Primario.

5.1.3. Fallo de la conexión del nuevo servidor Primario y vuelta al servidor Primario Anterior

A partir del estado alcanzado en la prueba anterior, se reinició la instancia de RDISK del *Node0*, para disponer así de un servidor que realice la función de Backup.

Luego, se inicia en *Node2* un Cliente de RDISK, que enviará y recibirá mensajes a/desde el servidor RDISK (Fig. 3-C).

Una vez ejecutado el Cliente, los mensajes fueron enviados y recibidos a/desde el servidor RDISK (*Node1*). A continuación, se causó un fallo de red de *Node1*, que fue detectado por el GCS y reportado tanto a los miembros RADAR como RDISK.

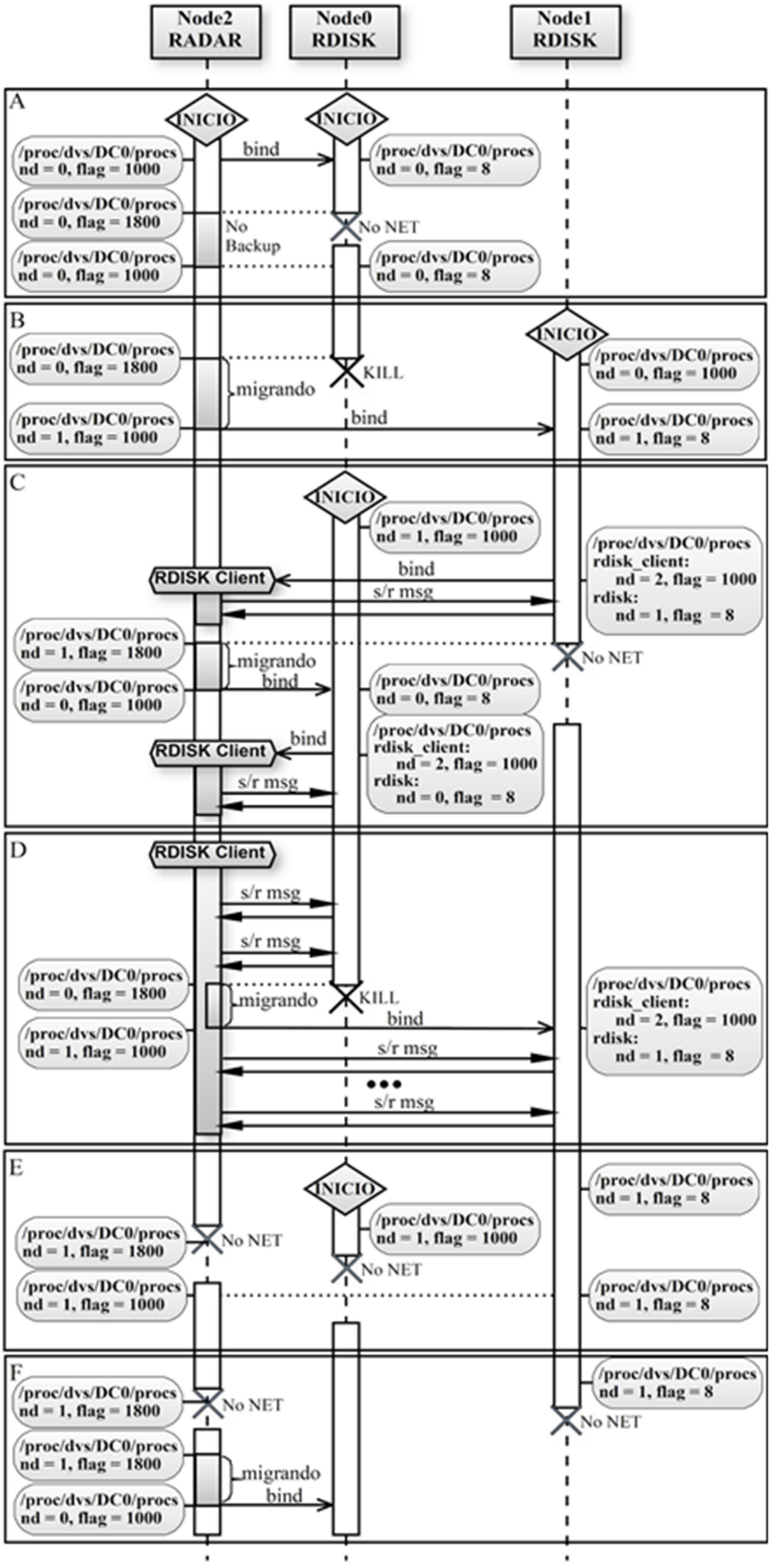


Figura 3.Batería de pruebas realizadas sobre RADAR.

La instancia de RDISK del *Node0* se proclama nuevo Primario, y el miembro RADAR de *Node2* modifica la configuración, primero para indicar que el proceso no es más alcanzable en *Node1* (*flag=1800*) y luego para que el *endpoint* del servicio indique a *nd=0*, como alcanzable (*flag=1000*).

El Cliente de RDISK funcionó sin problemas nuevamente, pero ahora enviando los mensajes y recibiendo a/desde *Node0*.

Se puede observar que antes y después del fallo, quien realiza las tareas de servidor Primario de RDISK asoció el cliente de RDISK en su lista de procesos del DC0. Este Cliente tiene valores de *nd=2* y *flag=1000* porque el proceso es Remoto y se ubica en *Node2*.

5.1.4. Fallo del proceso del servidor Primario con Backup

En esta prueba (Fig. 3-D), se utiliza a *Node0* como servidor Primario y a *Node1* como Backup. También se inicia el cliente de RDISK en *Node2*. Sin embargo, ahora se provoca la interrupción de servicio del Primario durante el envío de varios mensajes sucesivos (en lugar de un fallo de red), deteniendo al proceso RDISK en *Node0* en forma abrupta mediante el comando *kill*.

Cuando RADAR recibe un mensaje del GCS que da cuenta de una falla del servidor Primario, este comienza el proceso de relocalización para asociar el *endpoint* de RDISK a otro nodo. En este proceso, los valores *nd* y *flags* en el archivo del directorio */procs* pasan de valer *0→1* y *1800→1000* respectivamente.

A pesar de que el fallo se produce durante el envío de mensajes, el cliente no percibe ningún problema al enviar/ recibir mensajes y no requiere de ningún cambio de configuración, demostrando transparencia total al fallo.

De manera similar, se realizó una prueba con un escenario donde se finaliza abruptamente la VM (tal como un apagado o fallo eléctrico en un computador) en la que ejecuta el servidor Primario y los resultados fueron los mismos. Este escenario no se describe por ser prácticamente idéntico.

5.1.5. Fallo de conexión del nodo del servidor Backup y del nodo del Cliente.

A partir del estado alcanzado en la prueba anterior, se reinició la instancia de RDISK del *Node0*, para disponer así de un servidor que realice la función de Backup. A continuación se provocaron (artificialmente) fallas sobre el nodo donde se ejecutaba RADAR (*Node2*) y sobre el servidor de Backup (Fig. 3-E).

El resultado es que RADAR detecta la pérdida de conexión o de disponibilidad del servidor Primario, pero al no poder alcanzar al servidor Backup, la asociación del proceso de RDISK permanece en *Node1,* solo que su estado pasa a ser no alcanzable (*flag=1800*). Permaneció en ese estado hasta que se restableció la conexión del *Node2* al grupo, momento en el cual el estado del *endpoint* RDISK volvió a ser *flag=1000*.

5.1.6. Fallo de conexión del nodo Cliente

En esta prueba (Fig. 3-F), luego de ser reestablecida la conexión del *Node0* (que es ahora servidor Backup), se provocó un corte de conexión en *Node2* y poco después en *Node1*. En un primer momento, el estado del proceso RDISK paso a ser *flag=1800* porque ya no era alcanzable. Luego, cuando se restableció la conexión de *Node2*, RADAR cambio la asociación del proceso RDISK al *Node0* (*nd=0*), donde se ejecuta el nuevo servidor Primario (antes Backup).

El conjunto de pruebas realizadas contempló situaciones representativas de las funciones especificadas como requerimientos de diseño de RADAR. En ellas se consideraron tanto fallos de procesos servidores Primarios, servidores Backups y clientes, como así también la detención explícita o abrupta de nodos, y las interrupciones o particiones de la red del cluster. Todas las pruebas arrojaron los resultados esperados y demostraron la principal característica de RADAR de permitir que los clientes de un servicio operen sin interrupciones y con total transparencia ante fallos de servidores replicados en un cluster DVS.

6. Conclusiones y Trabajos Futuros

Existen diversas soluciones para resolver el problema de la localización y seguimiento de servicios ante fallos. Estas soluciones presentan características deseadas en cuanto a escalabilidad y disponibilidad se refiere pero, carecen de sencillez en la implementación y gestión de los mecanismos a los que recurre para lograrlo. Es necesario tratar con varias configuraciones y esto debe tenerse en cuenta al desarrollar aplicaciones distribuidas.

RADAR, además de brindar escalabilidad, confiabilidad y disponibilidad, es sencillo de implementar y configurar; y ligero en términos de requerimientos, lo que facilita su despliegue, mantenimiento y operación reduciendo los costos asociados. Todo esto se logra utilizando la infraestructura provista por el DVS y al GCS como parte de esa infraestructura.

RADAR permite que los clientes continúen operando normalmente, brindando transparencia total ante fallos de los servidores replicados que monitorea.

Como trabajo a futuro se plantea permitir el funcionamiento de RADAR con configuraciones dinámicas utilizando servidores de tipo Clave-Valor, ya que en este prototipo se utilizaron archivos de configuración.

Finalmente, como el prototipo de RADAR se desarrolló con soporte para esquemas de replicación P/B y

RSM, se planean realizar las modificaciones necesarias para utilizar el esquema Chain Replication [19].

Referencias

[1] P. Pessolani, T. Cortes, F. Tinetti, S. Gonnet; "*An Architecture Model for a Distributed Virtualization System*"; Cloud Computing 2018; The Ninth International Conference on Cloud Computing, GRIDs, and Virtualization; Barcelona, España.2018.

[2] R. Nikolaev, G. Back; "*VirtuOS: An operating system with kernel virtualization*"; Proceedings of the Twenty-Fourth ACM Symposium on Operating Systems Principles; 2013.

[3] N. Budhiraja, K. Marzullo, F. B. Schneider, S.Toueg, "*The primary-backup approach*", in Distributed systems (2nd Ed.), Sape Mullender (Ed.). ACM Press/Addison-Wesley Publishing Co., New York, NY, USA 199-216, 1993.

[4] T. Li, B.Cole, P. Morton, D. Li, "*RFC 2281 - Cisco Hot Standby Router Protocol (HSRP)*", Network Working Group IETF, March 1998.

[5] Nadas, S., Ed., "*RFC 5798 - Virtual Router Redundancy Protocol (VRRP) Version 3 for IPv4 and IPv6*", Network Working Group IETF, March 2010.

[6] Schneider, F., "*Implementing Fault-Tolerant Services Using the State Machine Approach: A Tutorial*", ACM Computing Surveys. vol. 22, no. 4, pp. 299–319, 1990.

[7] The Spread Toolkit. http://www.spread.org, accedido Mayo 2019.

[8] M. Alemandi, O. Jara, "*Un driver de disco tolerante a fallos*", Jornada de Jóvenes Investigadores Tecnológicos (JIT 2015), Rosario, 2015.

[9] Apache ZooKeeper, https://zookeeper.apache.org/, accedido Julio 2019.

[10] Etcd, https://github.com/etcd-io/etcd, accedido Julio 2019.

[11] Consul, https://www.consul.io/, accedido Julio 2019.

[12] Ongaro, D & Ousterhout, J. "*In search of an understandable consensus algorithm*". USENIX, 2014.

[13] Kubernetes, https://kubernetes.io/es/, accedido Julio 2019.

[14] Docker, https://www.docker.com/, accedido Julio 2019.

[15] P. Pessolani, T. Cortes, F. G. Tinetti, and S. Gonnet, "*An IPC Software Layer for Building a Distributed Virtualization System*", Congreso Argentino de Ciencias de la Computación (CACIC 2017) La Plata, Argentina, October 9-13, 2017.

[16] JGroups, http://www.jgroups.org/, accedido Julio 2019.

[17] Miranda, Hugo & Pinto, Alexandre & Rodrigues, Luís. "*Appia, a Flexible Protocol Kernel Supporting Multiple Coordinated Channels*", 2001.

[18] L. E.Moser , Y. Amir, P. M. Melliar-Smith, D. A. Agarwal, "*Extended Virtual Synchrony*", in Proceedings of the IEEE 14th International Conference on Distributed Computing Systems (Poznan, Poland, June), 1994.

[19] R.Van Renesse, F. B. Schneider, "*Chain Replication for Supporting High Throughput and Availability*", 6th OSDI 2004: San Francisco, California, USA December 2004.